

DevOps on Z Proof of Technology (PoT)

Exercises Using zDT on Cloud

Updated September 10 2021

Contents

OVERVIEW.....	9
LAB 1 - WORKING WITH MAINFRAME USING COBOL AND DB2 (90 - 120 MINUTES).....	10
SECTION 1 – CONNECT TO A Z/OS SYSTEM.....	11
____ 1.1 WORKING WITH THE IDZ WORKSPACE.....	11
____ 1.2 CONNECTING TO THE Z/OS REMOTE SYSTEM.....	13
SECTION 2 EXECUTE THE DB2/COBOL BATCH PROGRAM AND VERIFY THE ABEND.....	14
____ 2.1 SUBMIT A COBOL/DB2 BATCH TO EXECUTE.....	15
____ 2.2 ADD Z/OS RESOURCES TO THE MVS SUBPROJECT	17
____ 2.3 SUBMIT A JCL TO EXECUTE THE COBOL PROGRAM.....	18
SECTION 3. USE FAULT ANALYZER TO IDENTIFY THE CAUSE OF THE ABEND.....	20
____ 3.1 USING FAULT ANALYZER PERSPECTIVE.....	21
SECTION 4. USING THE IBM Z/OS DEBUGGER FOR A TEMPORARY FIX	26
____ 4.0 BE SURE THAT IDZ CLIENT IS LISTENING ON PORT 8001.....	26
____ 4.1 SUBMIT THE JCL TO INVOKE THE DEBUG	26
____ 4.2 (OPTIONAL) USING THE VISUAL DEBUGGER FOR STACK PATTERN BREAKPOINTS.....	34
____ 4.3 ACCESSING THE Z/OS JES	37
SECTION 5. MODIFY THE COBOL CODE TO FIX THE BUG.....	39
____ 5.1 USING THE EDITOR WITH Z/OS COBOL PROGRAMS.....	39
____ 5.2 FIXING THE PROGRAM REGI0B.....	45
____ 5.3 COMPILE AND LINK REGI0B.	47
____ 5.4 EXECUTE THE COBOL/DB2 PROGRAM	51
SECTION 6. USING THE CODE COVERAGE.....	53
____ 6.1 CREATING A JCL TO RUN THE CODE COVERAGE	53
____ 6.2 SUBMIT THE JCL FOR Z/OS EXECUTION.....	55
SECTION 7. (OPTIONAL) EXECUTING SQL STATEMENTS WHEN EDITING THE PROGRAM.....	58
____ 7.0 STARTING IDZ VERSION 14 AND CONNECT TO Z/OS.....	58
____ 7.1 CREATING A CONNECTION TO THE DB2 ON Z/OS.....	59
____ 7.2 RUNNING A SQL QUERY FROM COBOL PROGRAM.....	60
____ 7.3 USING THE SQL OUTLINE VIEW.....	64
____ 7.4 USING SQL VISUAL EXPLAIN	64
____ 7.5 REVERSE ENGINEER THE QUERY.....	65
____ 7.6 DISPLAYING DB2 TABLE CONTENT.....	67
SECTION 8. (OPTIONAL) USING FILE MANAGER	69
____ 8.1 VERIFY THAT YOU ARE CONNECTED TO THE PDTTOOLS COMMON COMPONENTS	69
____ 8.2 USING VIEW LOAD MODULE UTILITY	69
____ 8.3 WORKING WITH Z/OS DATA SETS.....	72
____ 8.4 WORKING WITH TEMPLATES	82
LAB 2D -(OPTIONAL) Z/OS CONNECT EE TOOLKIT : CREATE AN API FROM CATALOG	
MANAGER APPLICATION (EGUI).....	88
SECTION 1 – CREATE AND CONFIGURE A Z/OS CONNECT SERVICE PROJECT	89
____ 1.1 CREATE THE Z/OS CONNECT SERVICE PROJECT.....	89
____ 1.2 SERVICE CONFIGURATION.....	91
____ 1.3 CONNECTING TO A Z/OS CONNECT EE SERVER.....	101
____ 1.4 DEPLOYING THE SERVICE TO Z/OS CONNECT	102
SECTION 2 – CREATE AND CONFIGURE A Z/OS CONNECT API PROJECT	104
____ 2.1 CREATE THE Z/OS CONNECT API PROJECT AND IMPORT THE SERVICE	104
____ 2.2 MAPPING THE API TO THE SERVICE.....	106
SECTION 3 – DEPLOY THE API	109
____ 3.1 DEPLOY API TO Z/OS	109
SECTION 4 – TESTING DEPLOYED API USING SWAGGER.....	111
____ 4.1 INVOKING SWAGGER TO TEST THE API.....	111
SECTION 5 – (OPTIONAL) IMPORT THE SOLUTION	113

LAB 5 (OPTIONAL) CREATE URBancode DEPLOY INFRASTRUCTURE AND DEPLOY TO Z/OS.....	115
LAB ARCHITECTURE AND PROPOSED SCENARIO	115
PART 1 – CREATE THE URBancode APPLICATION INFRASTRUCTURE	116
TASK 2 – CREATE A UCD COMPONENT	119
TASK 3 – CREATE A SHIP LIST FILE AND Z/OS COMPONENT VERSION	121
TASK 4 - CREATE THE UCD COMPONENT VERSION FROM JCL.....	126
TASK 5 - CREATE UCD RESOURCES.....	132
TASK 6 - CREATE AN URBancode APPLICATION	136
TASK 7 - CREATE ENVIRONMENT	137
PART 2 - CREATE THE URBancode DEPLOYMENT PROCESSES.....	140
TASK 8 - CREATE A COMPONENTS PROCESS.....	140
TASK 9 - CREATE AN APPLICATION PROCESS	155
TASK 10 – ENVIRONMENT PROPERTIES CONFIGURATION.....	158
PART 3 – DEPLOY THE APPLICATION TO Z/OS CICS	162
TASK 11 – RUN AN APPLICATION PROCESS	162
TASK 12 – VERIFYING THE DEPLOY RESULTS AT Z/OS CICS.....	165
LAB 6B – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING REMOTE ASSETS (60 MINUTES)	167
OVERVIEW OF DEVELOPMENT TASKS.....	167
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	168
___ 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL	168
___ 1.2 RUN CICS TRANSACTION HCAZ	170
SECTION 2 – IMPORT A Z/OS PROJECT	171
___ 2.1 IMPORTING THE LAB6B Z/OS PROJECT	172
SECTION 3 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	174
___ 3.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE.....	174
___ 3.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	175
SECTION 4. GENERATE, BUILD AND RUN THE UNIT TEST.....	183
___ 4.1 GENERATING, BUILDING AND RUNNING THE TEST CASE PROGRAM.....	183
___ 4.2 RUNNING ZUNIT TEST CASE WITH CODE COVERAGE.....	186
SECTION 5. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.....	189
___ 5.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	189
___ 5.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS	190
___ 5.3 RERUNNING THE TEST CASE.....	191
___ 5.4 RUNNING ZUNIT TEST CASE WITH DEBUGGING	193
SECTION 6. RUN THE UNIT TEST FROM A BATCH JCL.....	197
___ 6.1 RUNNING THE UNIT TEST FROM A BATCH JCL	197
LAB 6C – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING LOCAL ASSETS (60 MINUTES)	200
OVERVIEW OF DEVELOPMENT TASKS	200
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	201
___ 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL	201
___ 1.2 RUN CICS TRANSACTION HCAZ	203
SECTION 2 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	204
___ 2.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE.....	204
___ 2.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	205
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST.....	214
___ 3.1 GENERATING THE TEST CASE PROGRAM.....	214
___ 3.2 BUILDING THE GENERATED TEST CASE PROGRAMS USING IBM DBB	215
___ 3.3 RUNNING THE TEST CASE,.....	220
___ 3.4 VERIFY THE JCL SUBMITTED	223
___ 3.5 RUNNING ZUNIT TEST CASE CODE COVERAGE.....	223
SECTION 4. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.....	227

— 4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	227
— 4.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS USING DBB.....	228
— 4.3 RUNNING THE TEST CASE AGAIN	230
— 4.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	233
SECTION 5. (OPTIONAL) RUN THE UNIT TEST FROM A BATCH JCL.	237
— 5.1 RUNNING THE UNIT TEST FROM A BATCH JCL	237

**LAB 6D – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL/DB2 BATCH PROGRAM
(60 MINUTES).....** **240**

OVERVIEW OF DEVELOPMENT TASKS	240
PART #1 – UNIT TEST ON PROGRAM DB2BATCH AND INTRODUCE A BUG	241
SECTION 1. RUN THE COBOL/DB2 BATCH PROGRAM USING JCL	241
— 1.0 CONNECT TO z/OS USING IDz.....	241
— 1.1 SUBMIT A PROVIDED JCL FOR EXECUTION.....	242
SECTION 2 – USE zUNIT TO RECORD THE BATCH EXECUTION.	245
— 2.1 UNDERSTANDING THE MAIN COBOL PROGRAM THAT READS FROM DB2 TABLE	245
— 2.2 RECORDING THE BATCH JCL EXECUTION	246
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.	252
— 3.1 GENERATING THE COBOL TEST CASE PROGRAMS.....	252
— 3.2 GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.....	253
— 3.3 RUNNING THE TEST CASE,.....	256
— 3.4 VERIFY THE JCL SUBMITTED THAT RUNS THE TEST CASE	258
SECTION 4. MODIFY THE COBOL/DB2 PROGRAM (INTRODUCE A BUG) AND RERUN THE UNIT TEST	259
— 4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	259
— 4.2 RE-COMPILe AND LINK THE CHANGED PROGRAM.....	260
— 4.3 RUNNING THE TEST CASE AGAIN	262
SECTION 5. RUN THE BATCH PROGRAM AND VERIFY THE BUG	264
— 5.1 VERIFY THE BUG ON THE PRINTED REPORT.....	264
SECTION 6. USE IDz TO FIX THE BUG, RECOMPILE THE COBOL/DB2 PROGRAM.	266
— 6.1 MODIFYING THE PROGRAM TO ELIMINATE THE BUG.....	266
— 6.2 RE-COMPILe AND LINK THE CHANGED PROGRAM.....	267
SECTION 7. RERUN THE zUNIT AND VERIFY THAT THE BUG IS ELIMINATED.	269
— 7.1 RUNNING THE TEST CASE AGAIN	269

**LAB 7 – USING IBM DEPENDENCY BASED BUILD WITH GIT, JENKINS AND UCD ON Z/OS
(60 MINUTES).....** **271**

SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	273
— 1.2 RUN CICS TRANSACTION J05P	275
SECTION 2 – LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE	277
— 2.1 CLONING THE GIT REPOSITORY	277
— 2.2 VERIFY THE CODE CLONED USING z/OS PROJECTS PERSPECTIVE	282
SECTION 3 –MODIFY THE COBOL CODE USING IDz.	283
— 3.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE	283
SECTION 4. USE IDz DBB USER BUILD TO COMPILE/BIND AND PERFORM PERSONAL TESTS.	285
— 4.1 USING DEPENDENCY BASED BUILDING OPTION	285
— 4.2 VERIFY THE DBB USER BUILD RESULTS	289
— 4.3 ISSUING A CICS NEW COPY	290
— 4.4 RUNNING J05P CICS TRANSACTION	293
SECTION 5. PUSH AND COMMIT THE CHANGED CODE TO Git.	294
— 5.1 USING IDz WITH THE GIT PLUGIN TO COMPARE THE CHANGE VERSUS ORIGINAL.....	294
— 5.2 PUSH AND COMMIT THE CHANGES TO Git.....	295
SECTION 6. USE JENKINS WITH GIT PLUGIN TO BUILD ALL THE MODIFIED CODE COMMITTED TO Git.	297
— 6.1 LOGON TO JENKINS USING A WEB BROWSER AND START THE z/OS AGENT.....	297
— 6.2 STARTING THE JENKINS PIPELINE	298
— 6.3 CHECKING THE RESULTS	300
— 6.4 UNDERSTAND THE RESULTS.....	302

SECTION 7. USE JENKINS AND UCD PLUGIN TO DEPLOY RESULTS AND TEST THE CICS TRANSACTION AGAIN	305
— 7.1 CHECKING THE DEPLOY RESULTS (SECOND STAGE)	305
— 7.2 CHECKING URBANCODE DEPLOY LOGS	306
— 7.3 USING JENKINS BLUE OCEAN PLUGIN.....	309
— 7.3 TESTING THE CICS CODE DEPLOYED TO DEV ENVIRONMENT	310
SECTION 8. (OPTIONAL) UNDERSTANDING DBB BUILD REPORTS	312
— 8.1 LOGIN TO THE DBB SERVER USING THE BROWSER	312
— 8.2 UNDERSTAND THE DBB COLLECTIONS	313
— 8.3 UNDERSTAND THE BUILD RESULTS.....	314
LAB 8 – (OPTIONAL) USING APPLICATION PERFORMANCE ANALYZER (APA) (60 MINUTES).....	316
SECTION 1. CREATE A NEW OBSERVATION REQUEST FOR A JOB THAT IS NOT RUNNING YET.....	317
— 1.1 CONNECT TO THE ADFz COMMON COMPONENTS	317
— 1.2 BEGIN A NEW APA OBSERVATION SESSION	319
SECTION 2 – RUN A SAMPLE BATCH JOB TO COLLECT PERFORMANCE DATA.....	321
— 2.1 CONNECT TO Z/OS	321
— 2.2 FIND THE JCL TO BE SUBMITTED.....	322
— 2.3 SUBMIT THE JCL TO EXECUTE THE COBOL/DB2 BATCH PROGRAM.....	323
SECTION 3 – REVIEW SOME OF THE REPORTS CREATED.....	324
— 3.1 DOWNLOAD THE MOST RECENT APA OBSERVATION REPORTS.....	325
— 3.2 ANALYZING THE APA MEASUREMENT PROFILE REPORT (S01)	327
— 3.3 ANALYZING THE APA LOAD MODULE SUMMARY REPORT (S03).....	330
— 3.4 MEASUREMENT ANALYSIS REPORT (S09).....	331
— 3.5 ANALYZING THE APA CPU USAGE BY CATEGORY REPORT (C01)	332
— 3.6 ANALYZING THE APA DB2 MEASUREMENT PROFILE REPORT (F01)	334
— 3.7 ANALYZING THE APA DB2 ACTIVITY BY STATEMENT REPORT (F04)	335
LAB 9 – (OPTIONAL) USING IBM Z VTP TO TEST A COBOL /CICS / DB2 TRANSACTION (60 MINUTES).....	337
OVERVIEW OF DEVELOPMENT TASKS	337
SECTION 1. USE VTP TO RECORD THE CICS/DB2 APPLICATION IN ACTION	338
— 1.1 EMULATE A 3270 TERMINAL AND CONNECT TO CICS VERSION 5.4	338
— 1.2 USING VTP TO RECORD THE HEALTH CARE APPLICATION TRANSACTION SEQUENCE	339
SECTION 2 – RUN A JCL TO EXECUTE THE RECORDED REPLAY SEQUENCE.....	344
— 2.1 CONNECT TO Z/OS USING IDz	344
— 2.2 SUBMIT A PROVIDED VTP JCL FOR EXECUTION	345
SECTION 3 – MODIFY ONE PROGRAM (INTRODUCE A BUG) AND RERUN THE VTP JCL	348
— 3.1 MODIFYING ONE COBOL PROGRAM INTRODUCING A BUG	348
— 3.2 BUILDING THE MODIFIED PROGRAM USING DBB.....	349
— 3.3 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM	351
SECTION 4. RUN THE CICS TRANSACTION AND VERIFY THE BUG	354
— 4.1 ISSUING A CICS NEW COPY USING 3270 EMULATION TERMINAL	354
— 4.2 EXECUTING HCAZ TRANSACTION	356
SECTION 5 USE IDz TO FIX THE BUG AND RECOMPILE/BIND THE PROGRAM.....	357
— 5.1 MODIFYING ONE COBOL PROGRAM AGAIN TO REMOVE THE BUG	357
— 5.2 REBUILDING THE FIXED PROGRAM USING DBB	358
SECTION 6 RERUN THE VTP JCL AND VERIFY THAT THE BUG IS ELIMINATED.....	359
— 6.1 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM	359
LAB 10 – (OPTIONAL) DEPLOYING COBOL/CICS/DB2 APPLICATION USING A GITLAB CI PIPELINE (60 MINUTES).....	363
SECTION 1. REVIEW THE GITLAB ISSUE AND VERIFY THE BUG USING THE 3270 TERMINAL.....	365
— 1.0 ACCESS GitLAB AND VERIFY THE ISSUE.....	365
— 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL	366
— 1.2 RUN CICS TRANSACTION SSC1	367

SECTION 2 – LOAD THE SOURCE CODE FROM GITLAB TO THE LOCAL IDZ WORKSPACE.....	369
— 2.1 CLONING THE GIT REPOSITORY	369
— 2.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	374
SECTION 3 – USE IDZ TO RUN THE ZUNIT TEST CASE AND VERIFY THE ERROR.....	375
— 3.1 RUNNING THE ZUNIT IN BATCH	375
— 3.2 VERIFYING THE ZUNIT RESULTS	378
SECTION 4. MODIFY THE COBOL/CICS/DB2 PROGRAM THAT HAS THE BUG USING IDz.....	379
— 4.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE	379
SECTION 5. USE IDz DBB USER BUILD TO COMPILE/LINK AND RUN THE ZUNIT.....	380
— 5.1 USING DEPENDENCY BASED BUILDING OPTION	380
— 5.2 VERIFY THE DBB USER BUILD RESULTS	385
— 5.3 RUNNING THE ZUNIT AGAIN.....	386
SECTION 6. PUSH AND COMMIT THE CHANGED CODE TO GITLAB.....	388
— 6.1 USING IDz AND GIT PERSPECTIVE TO COMPARE THE CHANGE VERSUS ORIGINAL	388
— 6.2 PUSH AND COMMIT THE CHANGES TO GITLAB	388
SECTION 7. VERIFY THE GITLAB CI BUILD EXECUTION.....	391
— 7.1 VERIFYING THE BUILD.....	391
SECTION 8. VERIFY THE GITLAB CI PACKAGE AND DEPLOY TO UCD AND TEST THE CICS TRANSACTION AGAIN USING 3270	393
— 8.1 CHECKING THE PACKAGING RESULTS (SECOND STAGE)	393
— 8.2 CHECKING URBancode CREATED VERSION	394
— 8.3 CHECKING THE DEPLOYMENT RESULTS (THIRD STAGE)	396
— 8.4 CHECKING URBancode DEPLOY RESULTS	397
— 8.5 TESTING THE CICS CODE DEPLOYED TO DEV ENVIRONMENT	399
LAB 11 – (OPTIONAL) MIGRATING PARTITIONED DATA SETS (PDS) TO A DISTRIBUTED GIT REPOSITORY. (50 MINUTES).....	402
SECTION 1. LOGON TO IDz AND PREPARE THE NECESSARY DIRECTORIES AND FILES ON ZFS	404
— 1.1 CONNECT TO Z/OS USING IDz.....	404
— 1.2 VERIFY THE Z/OS DATASETS TO BE MIGRATED.....	405
— 1.3 CREATE DIRECTORIES AND SCRIPT MIGRATION FILE AT ZFS	406
SECTION 2. MIGRATING USING DBB MIGRATION TOOL TO A LOCAL ROCKET GIT REPOSITORY ON ZFS	409
— 2.1 SETTING UP A LOCAL ROCKET GIT REPOSITORY ON ZFS	410
— 2.2 RUN MIGRATION USING A MAPPING FILE.....	411
— 2.3 SETTING THE FILE TAG AFTER THE MIGRATION	413
— 2.4 FINISHING MIGRATION PROCESS	415
SECTION 3. PUSH THE MIGRATED FILES TO GIT SERVER (GITLAB).....	416
— 3.1 CREATING A NEW GITLAB REPOSITORY	416
— 3.2 CONNECT THE LOCAL ROCKET GIT REPOSITORY WITH THE GITLAB REPOSITORY	419
— 3.3 PUSH THE EXISTING ROCKET GIT CLIENT REPOSITORY TO THE GIT SERVER REPOSITORY	419
— 3.4 ACCESS GITLAB REPOSITORY TO VERIFY THE CODE MIGRATED.....	419
SECTION 4. LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE AND START WORKING WITH THE MIGRATED ASSETS.....	422
— 4.1 CLONING THE GIT SERVER REPOSITORY USING IDz	422
— 4.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	427
— 4.3 COMMIT THE UPDATE TO THE GIT SERVER REPOSITORY	431
LAB 12 - (OPTIONAL) APPLICATION DISCOVERY : FIND A CANDIDATE FUNCTION IN A CICS APPLICATION IN ORDER TO CREATE AN API.....	434
OVERVIEW OF DEVELOPMENT TASKS.....	434
SECTION 1 REQUEST THE ADDI Z TRIAL LAB (ASSUMED THAT YOU HAVE COMPLETED THIS ON DAY 1)	435
SECTION 2 – LAB : DISCOVER A CANDIDATE API AND FIND ITS INTERFACE	437
— 2.1 OPEN YOUR ADDI zTRIAL.....	438
— 2.2 THE DISCOVER SCENARIO WILL OPEN THE WORKSHOP INSTRUCTIONS AND START IDz IN THE ADDI PERSPECTIVE.....	439

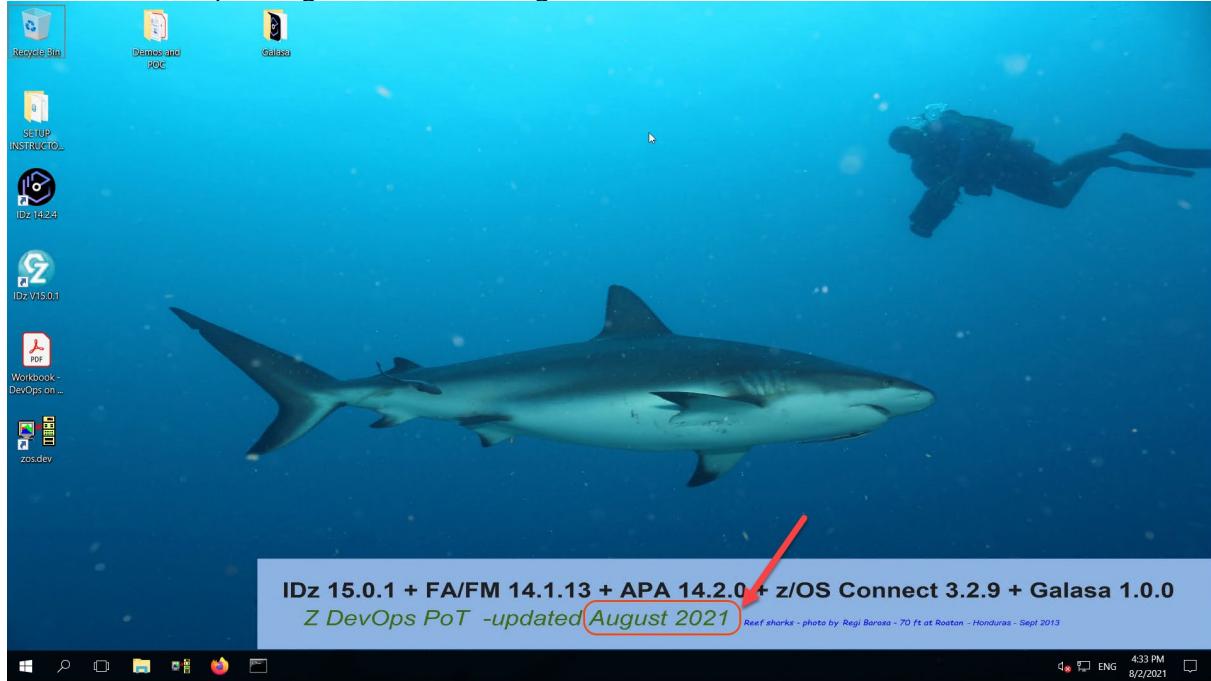
LAB 13 – (OPTIONAL) RUN INTEGRATION TESTS USING GALASA (60 MINUTES).....	441
OVERVIEW OF DEVELOPMENT TASKS.....	442
SECTION 1. START THE SAMPLE PROVIDED <i>SIMBANK</i> APPLICATION.....	443
SECTION 2. USE IBM PERSONAL COMMUNICATIONS (PCOM) TO LOG ONTO THE SAMPLE APPLICATION.....	445
SECTION 3. RUN THE IVT TEST TO VALIDATE THAT GALASA CAN CONNECT TO THE <i>SIMBANK</i> APPLICATION.....	448
SECTION 4. OPEN THE TEST RESULT EDITOR.....	449
SECTION 5. EXECUTE A MIXED WEB SERVICE AND 3270 TEST WITHIN THE GALASA FRAMEWORK	450
SECTION 6. EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS	452
SECTION 7. EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS	454
SECTION 8. CHANGE THE TEST TO LOG THE REQUEST AND RESPONSE FOR THE WEB SERVICE.....	457
SECTION 9. EXECUTE A BATCH TEST WITHIN THE GALASA FRAMEWORK	461
SECTION 10 EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS.....	463
SECTION 11 EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS	464

Overview

- Integrated application development and problem analysis ([ADFz](#))
- Managing your source code using modern tools (including [Git](#))
- Building automation ([DBB](#)) for COBOL or PL/I 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 01, 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

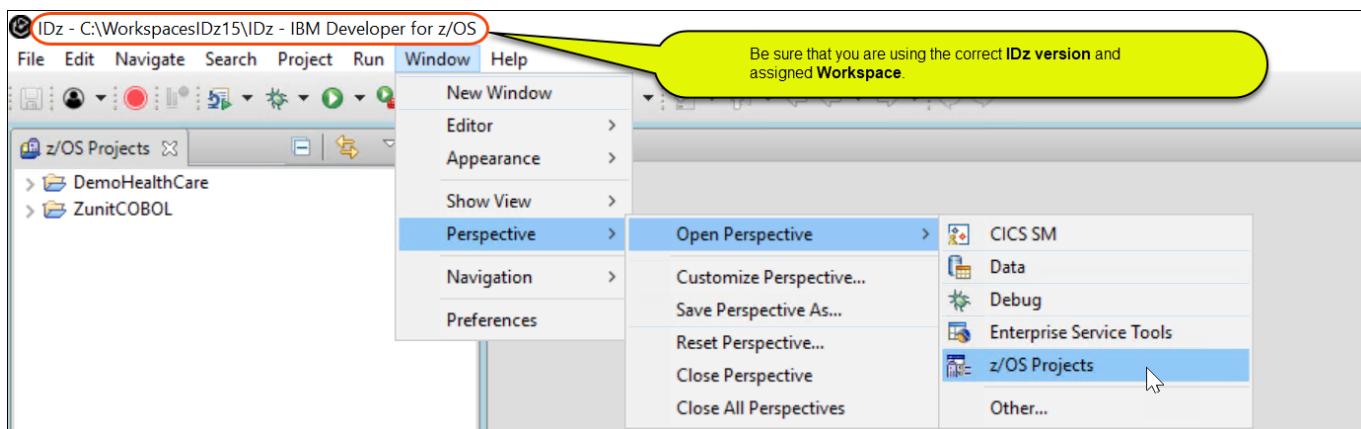
1.1.1 Start IBM Developer for z Systems version 15

- ▶▶ 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.

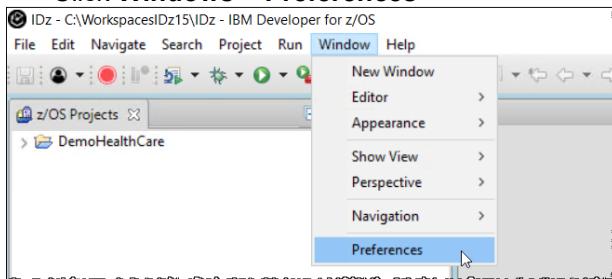
To open one of the views that were closed:

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.

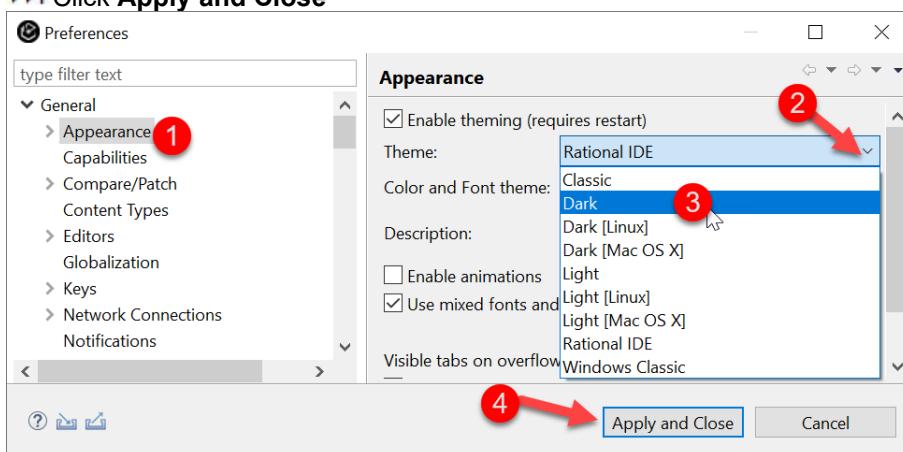
1.1.3 Starting on version 15 the developer can change the workspace appearance. If you want to try that

▶▶ Click Windows > Preferences



1.1.4 ▶▶ Under General select Appearance and on Theme select Dark.

▶▶ Click Apply and Close

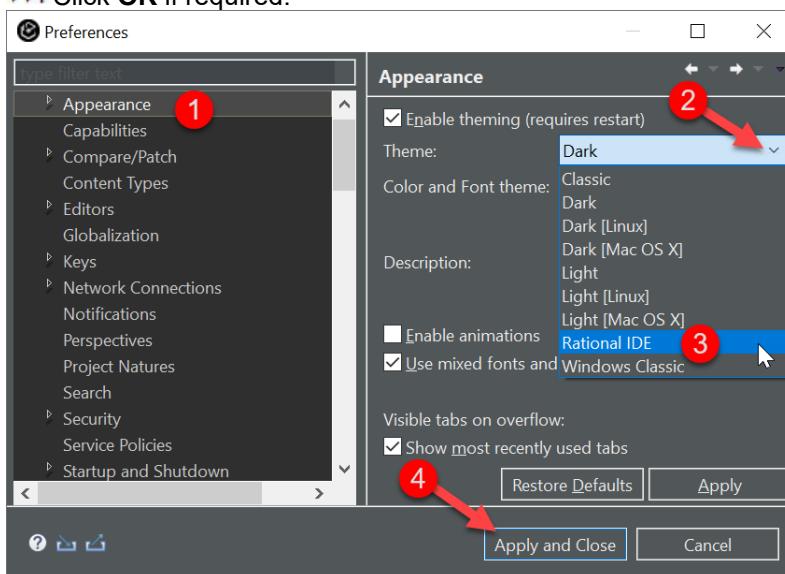


1.1.5 Since all picture here were done using "Rational IDE" theme we recommend to return to the default.

▶▶ Again, click Windows > Preferences

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

▶▶ Click OK if required.



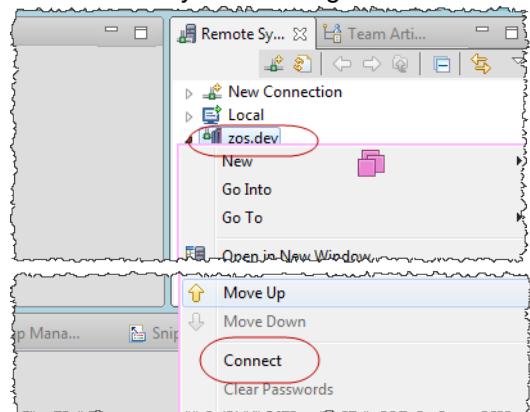
1.2 Connecting to the z/OS Remote system

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 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.



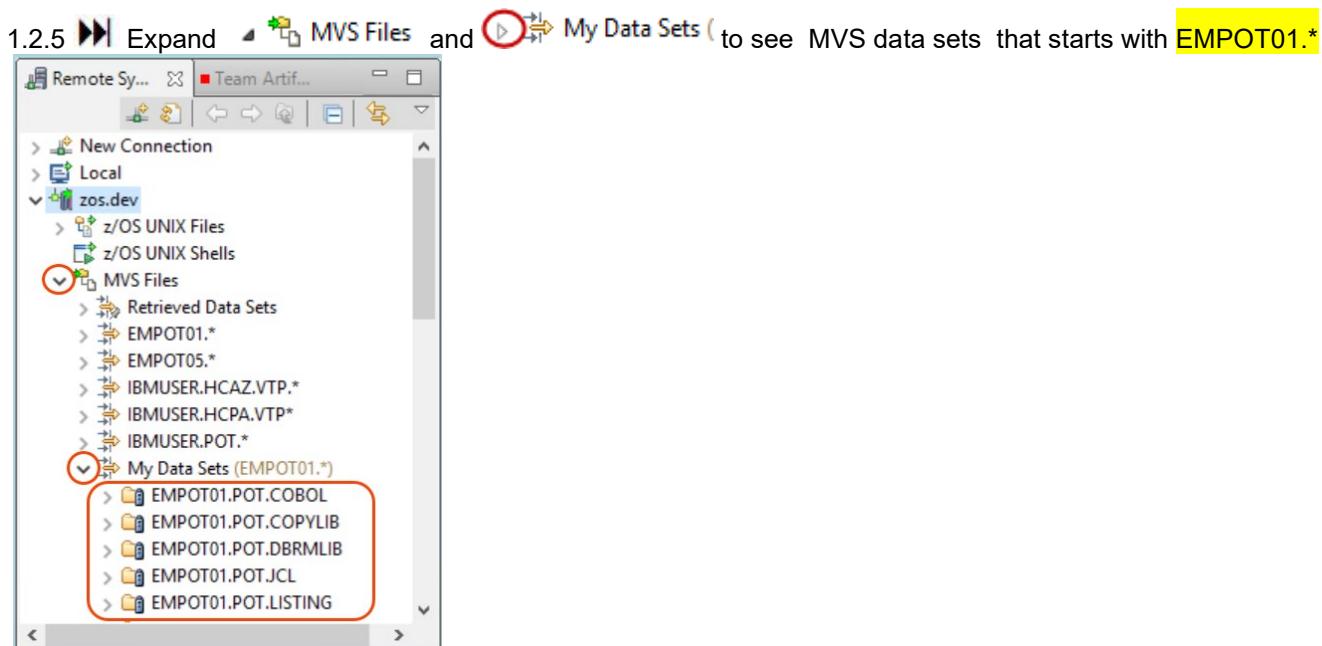
In case of connection errors...

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.



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.

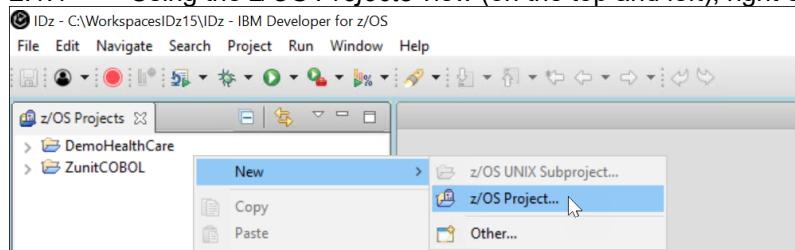
Before you create an MVS subproject, you need to have completed the following tasks:

- Connecting to a remote system
- Creating a z/OS project

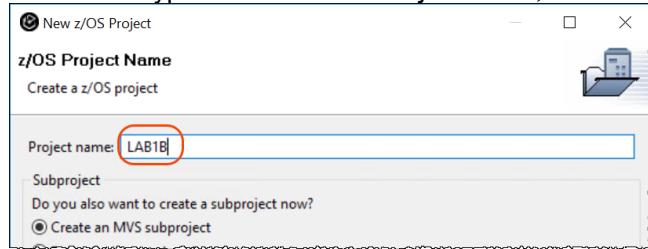
2.1 Submit a COBOL/DB2 batch to execute

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...

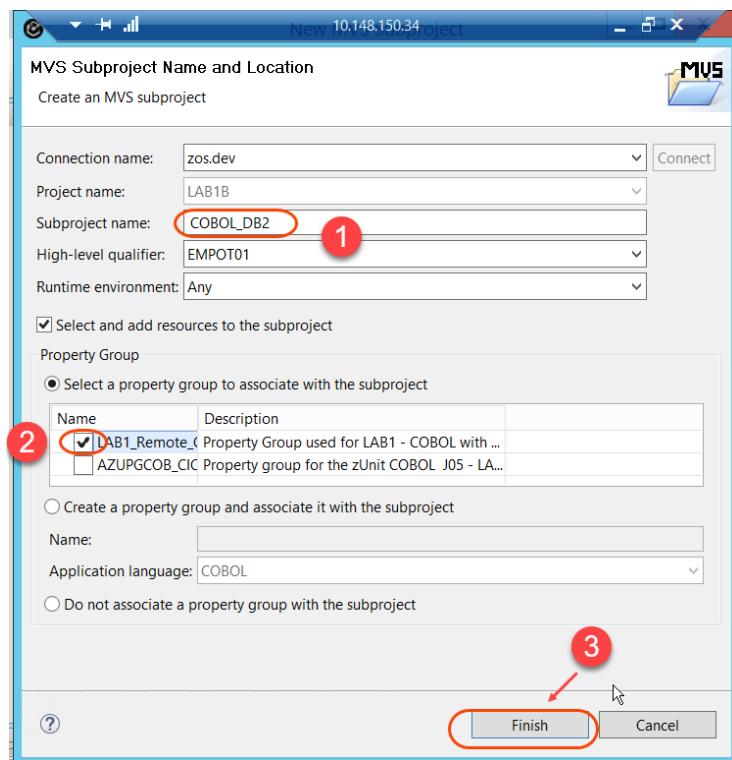


2.1.2 Type LAB1B as the Project name, select Create an MVS subproject and click Finish.

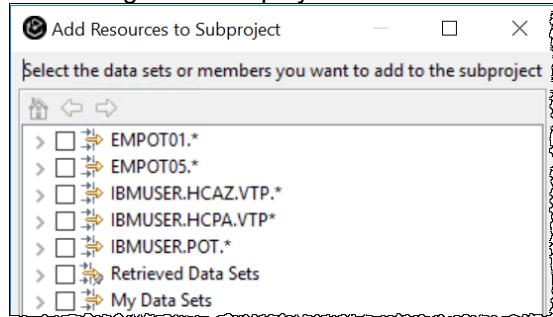


2.1.3 Type COBOL_DB2 as Subproject Name, select LAB1_Remote_Cobol.. as the property group and click Finish .

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 “_”.



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



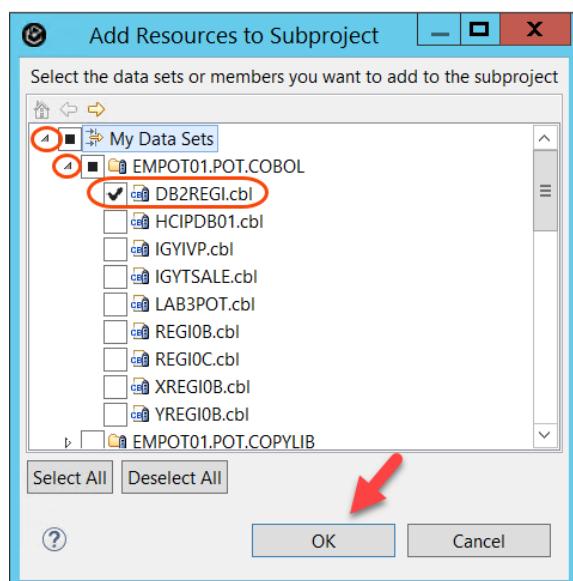
Property groups provide a way to manage resource properties, share them easily across systems, projects, resources, and users, and maintain consistency in your development and build environment.

You can, for example, define a property group that points to copybooks. This is something like //STEP LIB; 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**

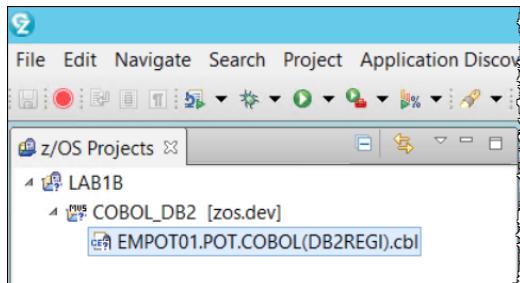


2.2.2 You should see a z/OS Project named **LAB1B** in your *z/OS Projects view*.

► Click **z/OS Projects view** (on left) and under **COBOL_DB2**, you will now see **DB2REGI** 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.



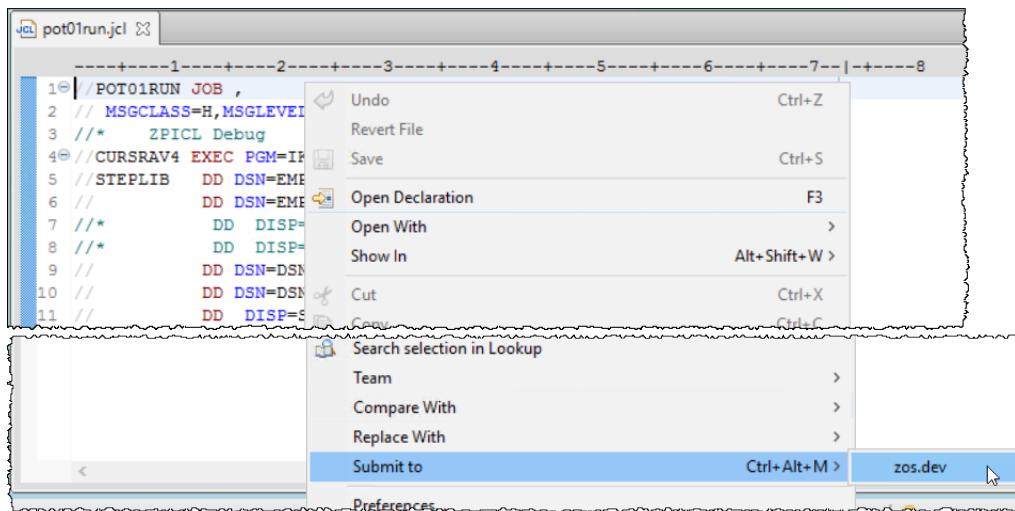
2.3 Submit a JCL to execute the COBOL program

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.

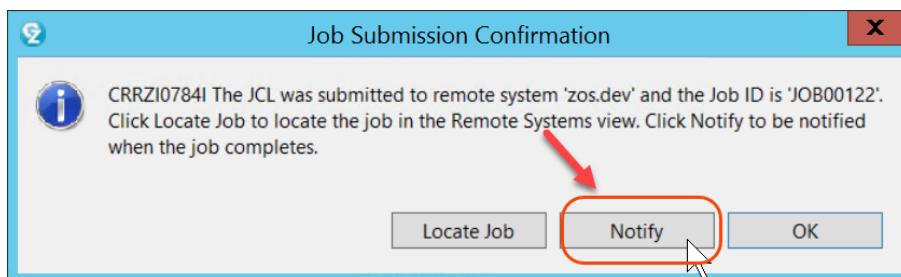
2.3.1 ► Using *Remote System View* (on top and right), scroll up to locate the file **pot01run.jcl** under **Local/Local Files/ADF_POT/empot01** and **double click** to edit.



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

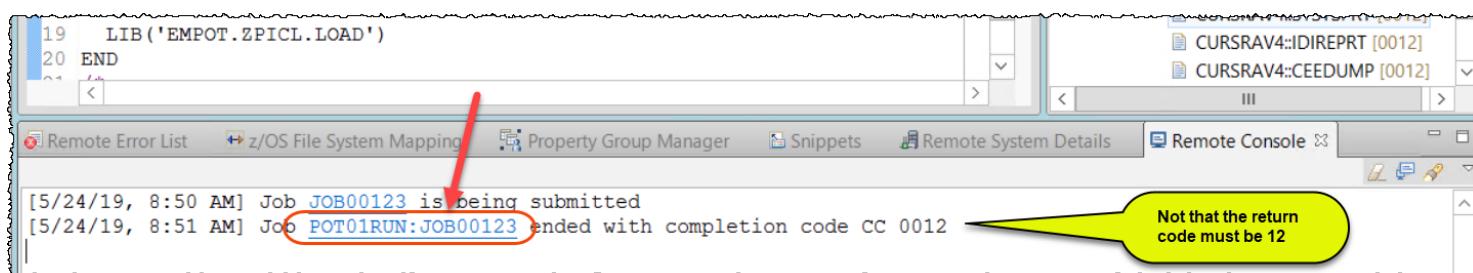


2.3.3 ► When the dialog pop up appears, click Notify. This allows you to be notified when the execution is ended.



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 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



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**.

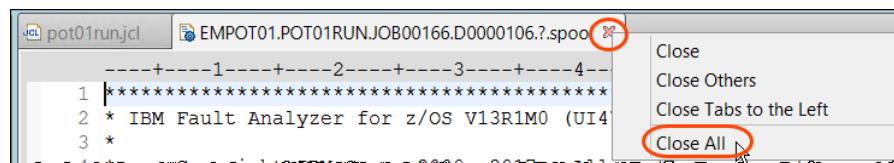
The screenshot shows the IBM z/OS Management Console interface. On the left, a code editor window displays a fault analysis report for job POT01RUN. A red box highlights the line "A system abend OCB occurred in module DB2REGI program DB2REGI at offset X'F9C'." On the right, the "Remote Systems" tree view is shown, with a red arrow labeled "1" pointing to the "Retrieved Jobs" node under the "JES" category. Another red arrow labeled "2" points to the "CURSRAV4::IDIREPRT" step within the "Retrieved Jobs" node.

```

JCL pot01run.jcl   EMPOT01.POT01RUN.JOB00147.D0000106.?spool
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8
1 ****
2 * IBM Fault Analyzer for z/OS V13R1M0 (UI47734 2017/06/01)
3 *
4 * Copyright IBM Corp. 2000, 2017. All rights reserved.
5 ****
6
7 JOBNAME: POT01RUN SYSTEM ABEND: OCB           SOW1      2021/04/02 09:09:34
8
9
10 <H1> I B M   F A U L T   A N A L Y Z E R   S Y N O P S I S
11
12
13 A system abend OCB occurred in module DB2REGI program DB2REGI at offset X'F9C'.
14
15 A program-interruption code 000B (Decimal-Divide Exception) is associated with
16 this abend and indicates that:
17
18 The divisor was zero in a signed decimal division.
19
20 The cause of the failure was program DB2REGI in module DB2REGI. The COBOL
21 source code that immediately preceded the failure was:
22
23 Source
24 Line #
25 -----

```

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.

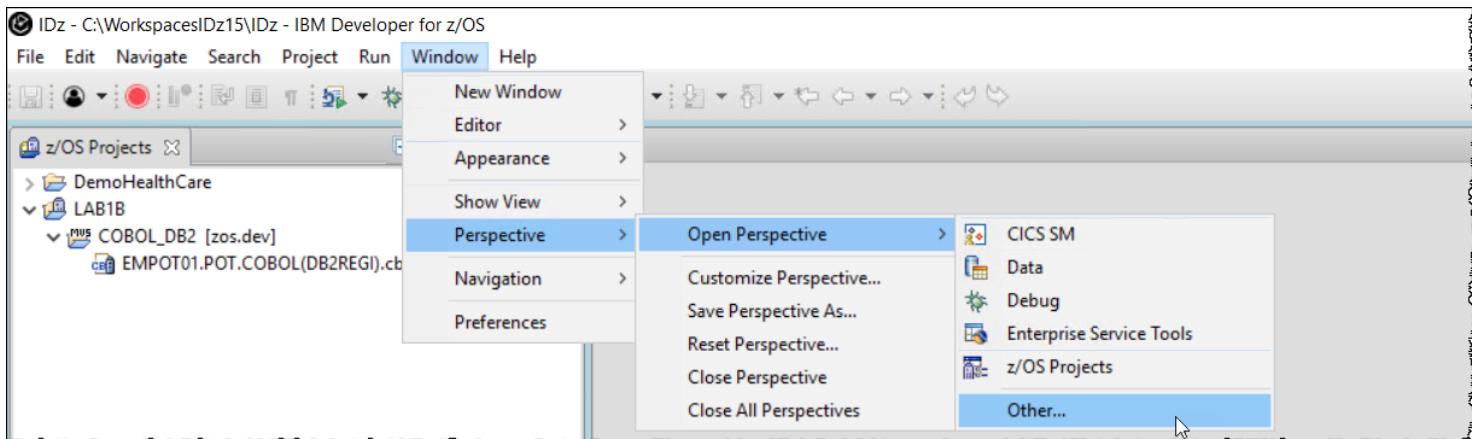
What is IBM Fault Analyzer for z/OS?

 **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.

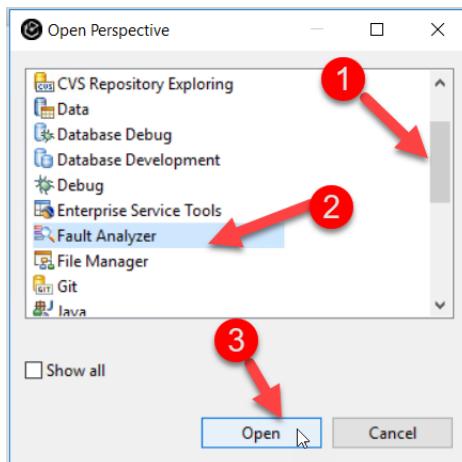
This helps the development team to identify the cause, analyze what went wrong and resolve the problem in a more timely and efficient manner to avoid costly interruptions that could jeopardize application schedules and outcomes.

3.1 Using Fault Analyzer perspective

3.1.1 ►| Open the **Fault Analyzer** perspective using **Window > Perspective > Open Perspective > Other...**

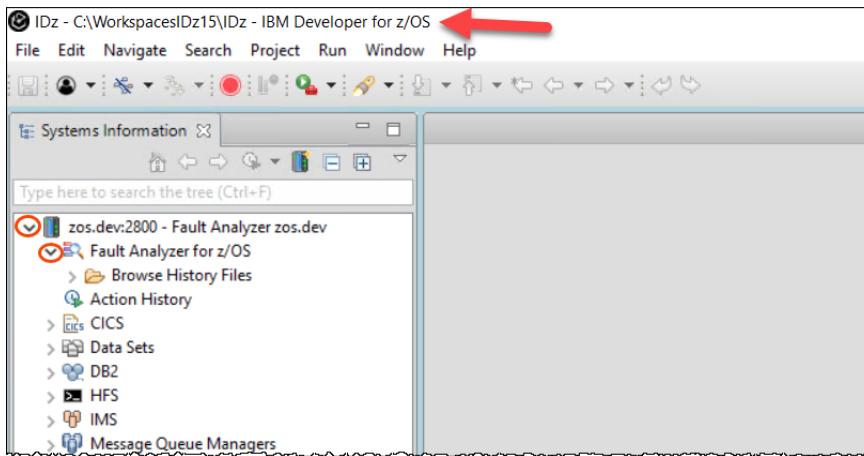


3.1.2 ►| Scroll down, select **Fault Analyzer** and click **Open**

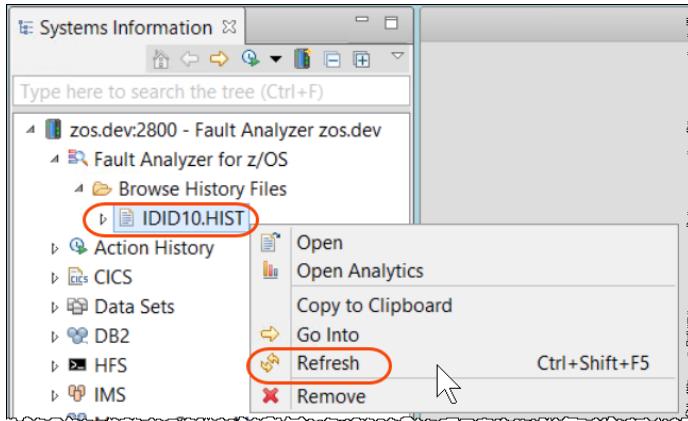


3.1.3 ►| Under **Systems Information** view (on left) be sure that **Fault Analyzer for z/OS** is expanded

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



- 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...

The screenshot shows the 'Systems Information' interface. The left pane displays the system tree. A red arrow points from the 'IDID10.HIST' folder in the tree view to the main pane. The main pane shows a table titled 'ZOS.DEV : 2800/IDID10.HIST' with the following data:

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

The screenshot shows the IBM Systems Information interface. On the left, there's a tree view under 'zos.dev:2800 - Fault Analyzer zos.dev' with nodes like 'Fault Analyzer for z/OS', 'Browse History Files', 'IDID10.HIST', 'Action History', 'CICS', 'Data Sets', 'DB2', 'HFS', 'IMS', and 'Message Queue Managers'. The main pane displays a report titled 'ZOS.DEV:2800/IDID10.HIST(F00353)-Report'. The report content includes:

```

1  ****
2  ****
3  * IBM Fault Analyzer for z/OS V13R1M0 (UI47734 2017/06/01)
4  *
5  * Copyright IBM Corp. 2000, 2017. All rights reserved.
6  ****
7  JOBNAME: POT01RUN SYSTEM ABEND: 0CB SOW1 2019/12/18 08:45:42
8
9
10 NOTE: This report was saved after reanalysis of the current fault entry--it was
11 not generated during real-time processing.
12
13
14
15
16 Module DB2REGI, program DB2REGI, source line # 365: Abend SOCBC (Decimal-Divide Exception)
17 IBM FAULT ANALYZER SYNOPSIS
18
19
20 A system abend 0CB occurred in module DB2REGI program DB2REGI at offset X'F9C'.
21
22 A program-interruption code 000B (Decimal-Divide Exception) is associated with
23 this abend and indicates that:

```

Below the report, there's a table titled 'Main Report' with columns: FAULT_ID, JOB/TRAN, USER_ID, SYS/JOB, ABEND, LABEND, JOB_ID, JOBNAME, and USERNAME. The data is as follows:

FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB	ABEND	LABEND	JOB_ID	JOBNAME	USERNAME
> F00353	POT01RUN	EMPOT01	SOW1	SOCBC	SOCBC	JOB00212	POT01RUN	
F00352	HCMA	IBMUSER	CICSTS53	4038	4038	STC00044	CICSTS53	

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.

The screenshot shows the 'Event Details' tab of the report. It displays COBOL source code and data field values at the time of the abend. Red annotations are present:

- A red circle labeled '1' points to the word 'RECEIVED-FROM-CALLED' in the source code line 19.
- A red circle labeled '2' points to the value '0' in the data field 'RECEIVED-FROM-CALLED' in the 'Data field values at time of abend' section.
- A red circle highlights the value '0' in the 'RECEIVED-FROM-CALLED' column of the table in the previous screenshot.
- A red arrow points from the 'RECEIVED-FROM-CALLED' value in the source code to the highlighted value in the data field values table.

The COBOL source code shown in the report is:

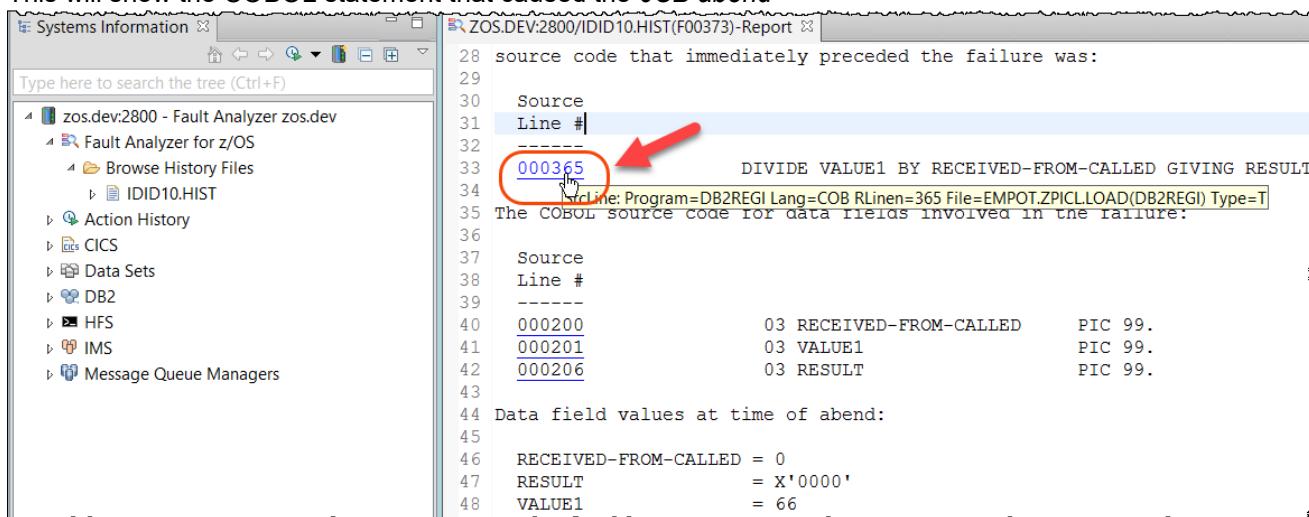
```

12
13 The cause of the failure was program DB2REGI in module DB2REGI. The COBOL
14 source code that immediately preceded the failure was:
15
16 Source
17 Line #
18 -----
19 000365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
20
21 The COBOL source code for data fields involved in the failure:
22
23 Source
24 Line #
25 -----
26 000200      03 RECEIVED-FROM-CALLED      PIC 99.
27 000201      03 VALUE1                  PIC 99.
28 000206      03 RESULT                  PIC 99.
29
30 Data field values at time of abend:
31
32 RECEIVED-FROM-CALLED = 0
33 RESULT = X'0000'
34 VALUE1 = 66
35
36

```

3.1.8 ► Using the *Main Report* view Click on the link **000365**

This will show the COBOL statement that caused the *OCB abend*



```

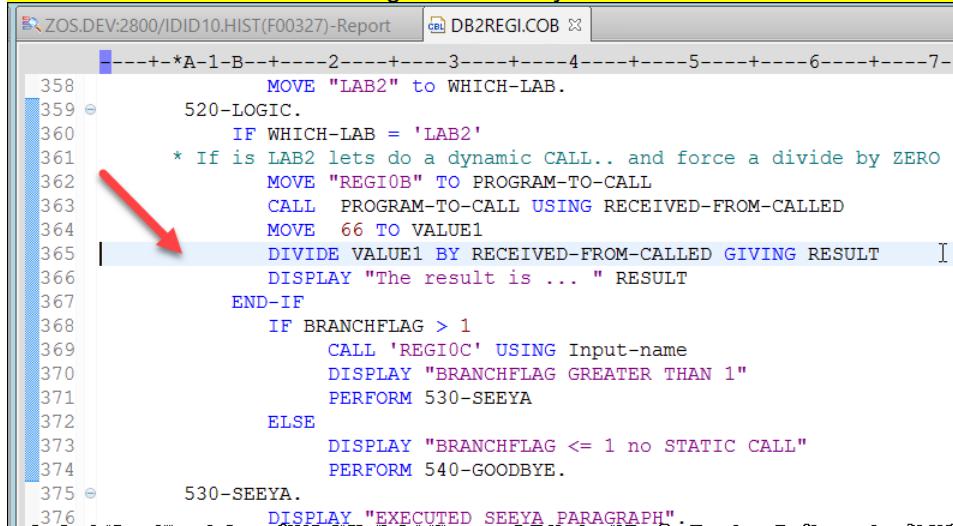
28 source code that immediately preceded the failure was:
29
30 Source
31 Line #
32
33 000365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
34 |Line: Program=DB2REGI Lang=COB RLine=365 File=EMPOT.ZPICLLOAD(DB2REGI) Type=T
35 The COBOL source code for data files involved in the failure:
36
37 Source
38 Line #
39 -----
40 000200      03 RECEIVED-FROM-CALLED      PIC 99.
41 000201      03 VALUE1                  PIC 99.
42 000206      03 RESULT                  PIC 99.
43
44 Data field values at time of abend:
45
46 RECEIVED-FROM-CALLED = 0
47 RESULT              = X'0000'
48 VALUE1              = 66

```

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.



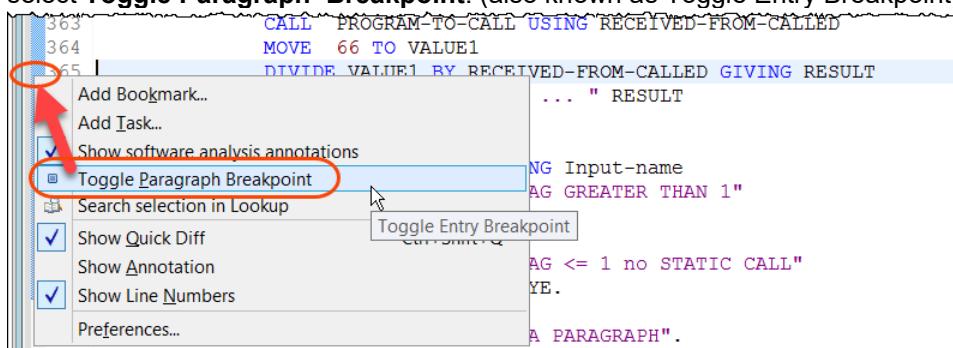
```

-----*A-1-B-----2-----3-----4-----5-----6-----7
      MOVE "LAB2" to WHICH-LAB.
359 520-LOGIC.
360   IF WHICH-LAB = 'LAB2'
361     * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
362     MOVE "REGIOB" TO PROGRAM-TO-CALL
363     CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364     MOVE 66 TO VALUE1
365     DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366     DISPLAY "The result is ... " RESULT
367   END-IF
368   IF BRANCHFLAG > 1
369     CALL 'REGIOC' USING Input-name
370     DISPLAY "BRANCHFLAG GREATER THAN 1"
371     PERFORM 530-SEEEYA
372   ELSE
373     DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
374     PERFORM 540-GOODBYE.
375 530-SEEEYA.
376  DISPLAY "EXECUTED SEEYA PARAGRAPH".

```

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)



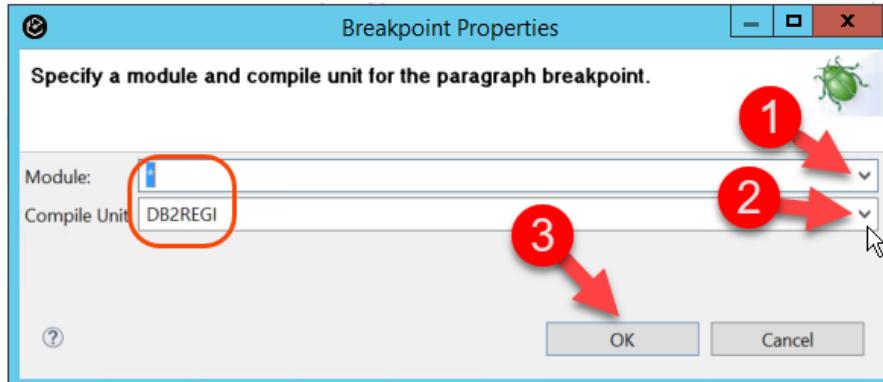
```

363   CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364   MOVE 66 TO VALUE1
365   DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
... " RESULT
      NG Input-name
      AG GREATER THAN 1"
      AG <= 1 no STATIC CALL"
      YE.
      A PARAGRAPH".

```

- Add Bookmark...
- Add Task...
- Show software analysis annotations
- Toggle Paragraph Breakpoint**
- Search selection in Lookup
- Show Quick Diff
- Show Annotation
- Show Line Numbers
- Preferences...

- 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.

```

ZOS.DEV:2800/IDID10.HIST(F00327)-Report DB2REGI.COB
-----+-----+-----+-----+-----+-----+-----+-----+
      358 MOVE "LAB2" to WHICH-LAB.
      359 520-LOGIC.
      360 IF WHICH-LAB = 'LAB2'
      361 * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
      362 MOVE "REGI0C" TO PROGRAM-TO-CALL
      363 CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
      364 MOVE 66 TO VALUE1
      365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
      366 DISPLAY "The result is ... " RESULT
      367 END-IF
      368 IF BRANCHFLAG > 1
      369 CALL 'REGI0C' USING Input-name
      370 DISPLAY "BRANCHFLAG GREATER THAN 1"
      371 PERFORM 530-SEEYA
      372 ELSE
      373 DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
      374 PERFORM 540-GOODBYE.
      375 530-SEEYA.

```

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

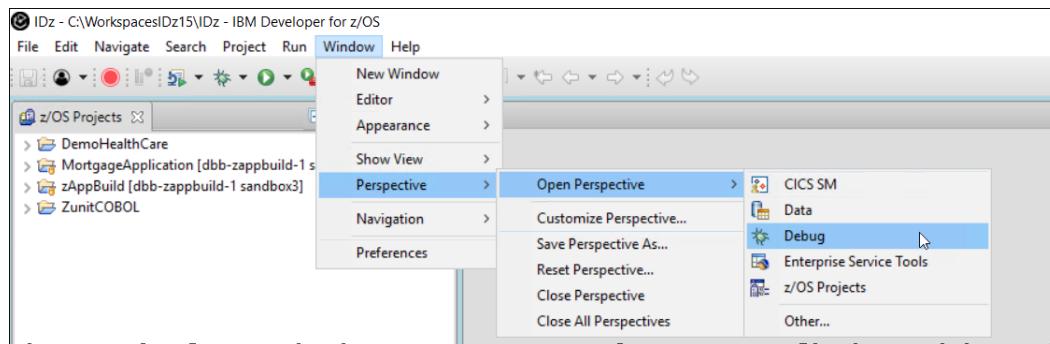
Section 4. Using the IBM z/OS Debugger for a temporary fix

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

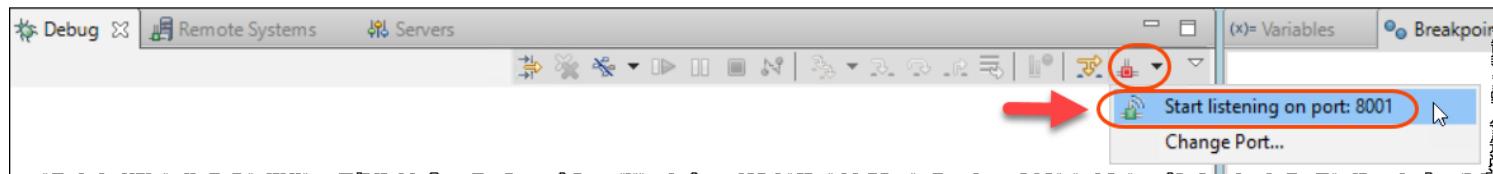
4.0 Be sure that IDz client is listening on port 8001

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**



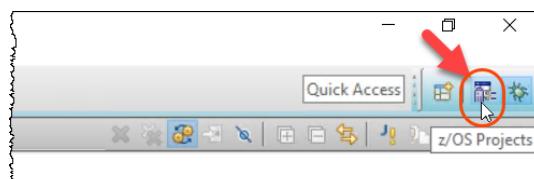
4.0.3 The listening icon will turn green and the IDz client is listening on port 8001.



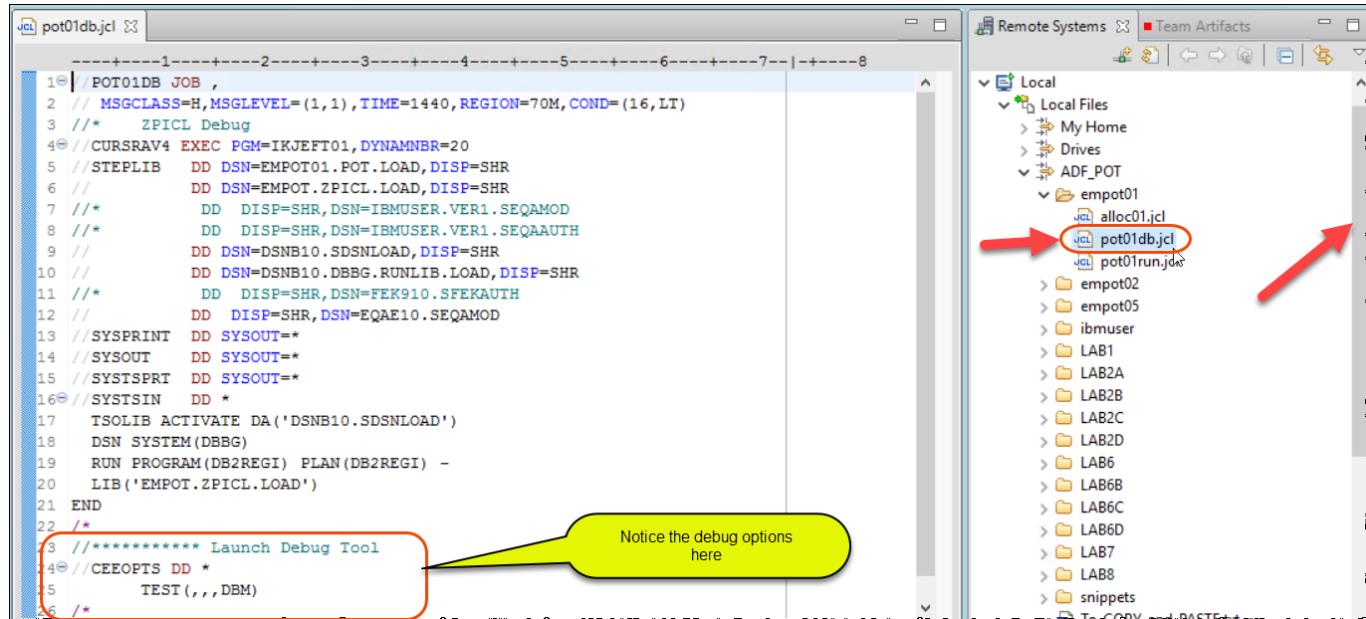
4.1 Submit the JCL to invoke the Debug

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

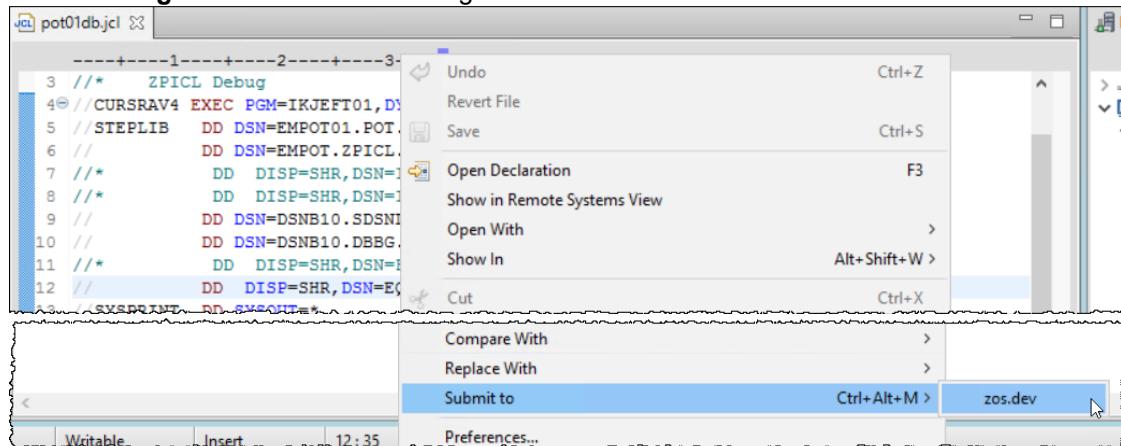
4.1.1 ► Using the top right corner Go to the **z/OS Projects** Perspective by clicking on the icon below.



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..



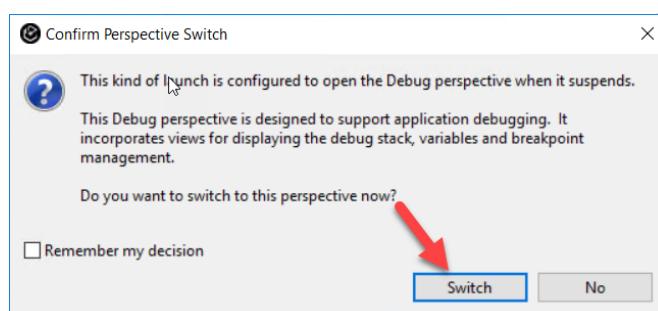
4.1.3 ► Right click on the JCL being edited and select **Submit to > zos.dev**



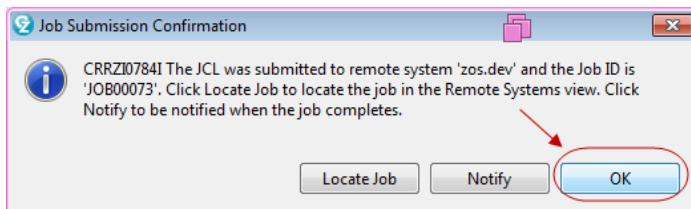
4.1.4 Once the job starts the z/OS debug will “talk” with you. Notice that the communication will be via the IDz connection (RSE), you don’t need to specify IP addresses. This may work even when firewalls are in place.

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

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

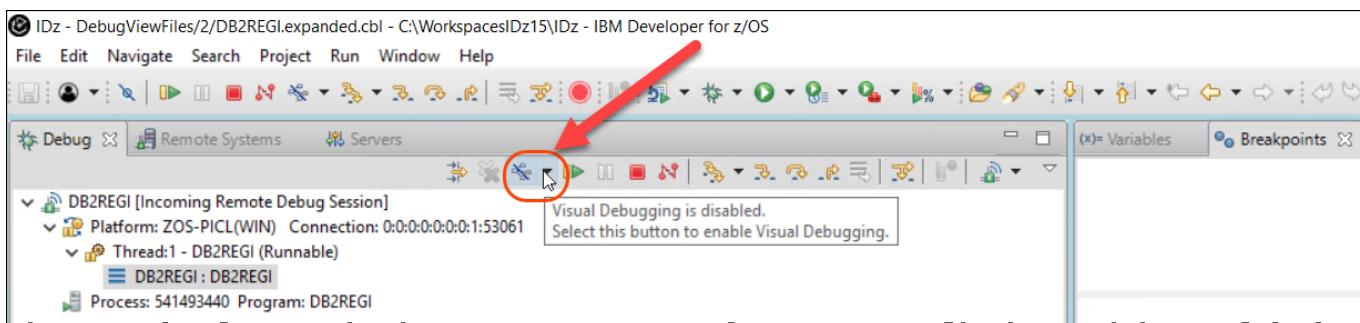


4.1.5 ► Also click **OK** for this dialog below



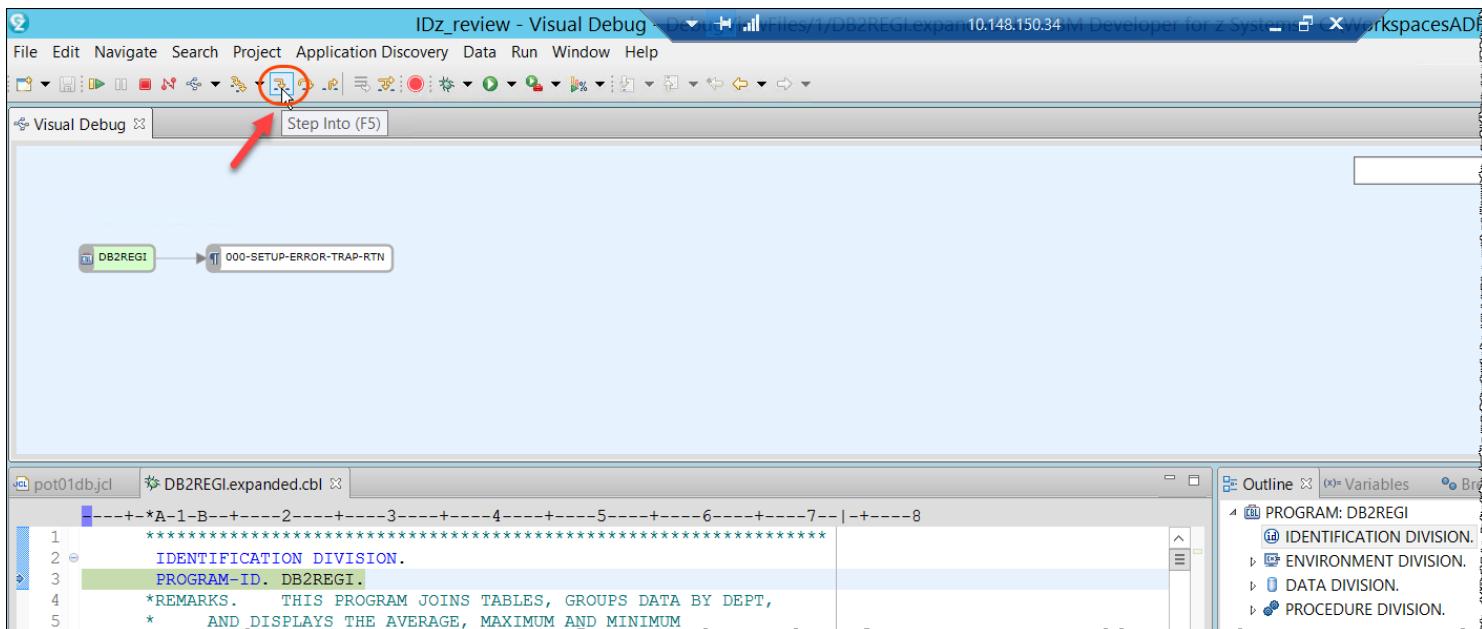
4.1.6 Using the *Debug* perspective:

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



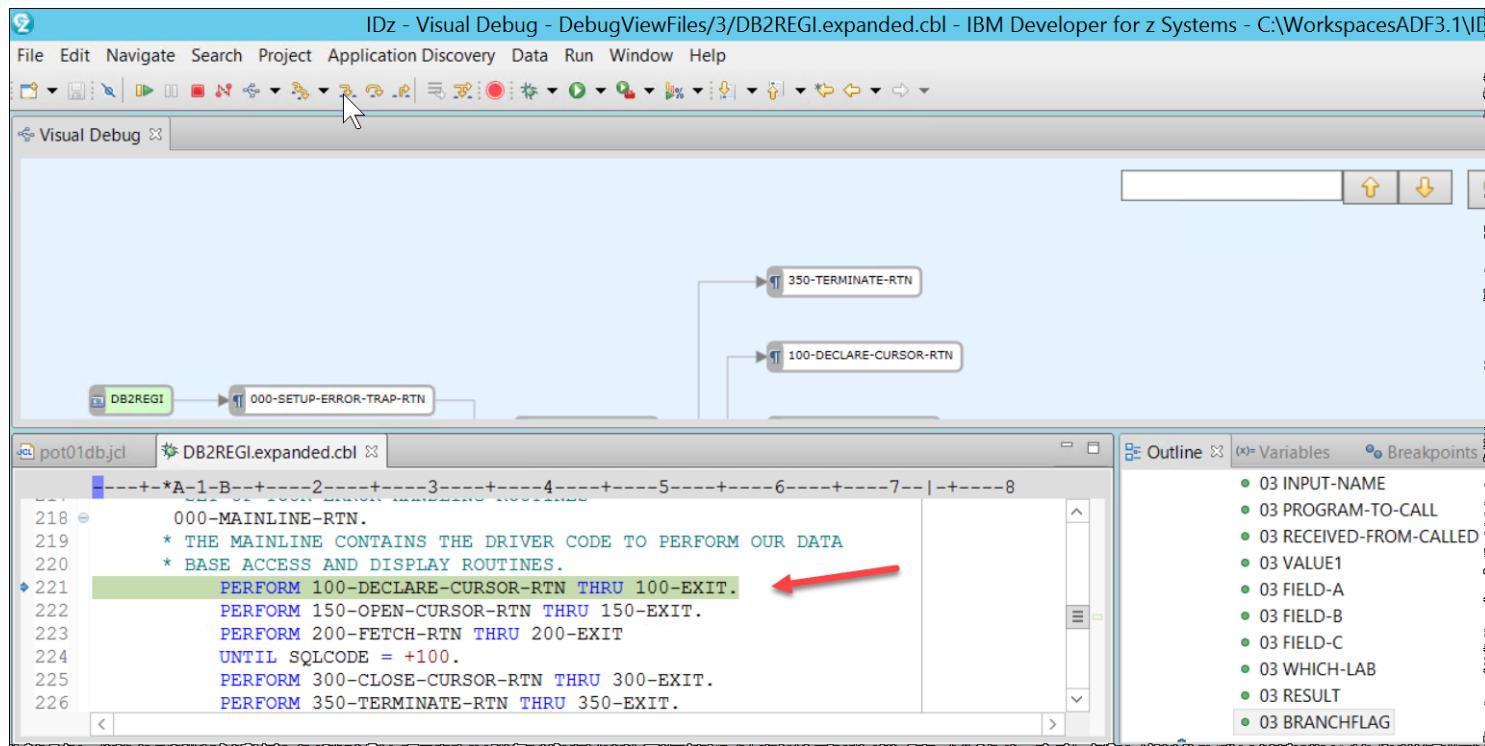
4.1.7 The *Visual Debug* view show the paragraphs being executed (in the top)

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



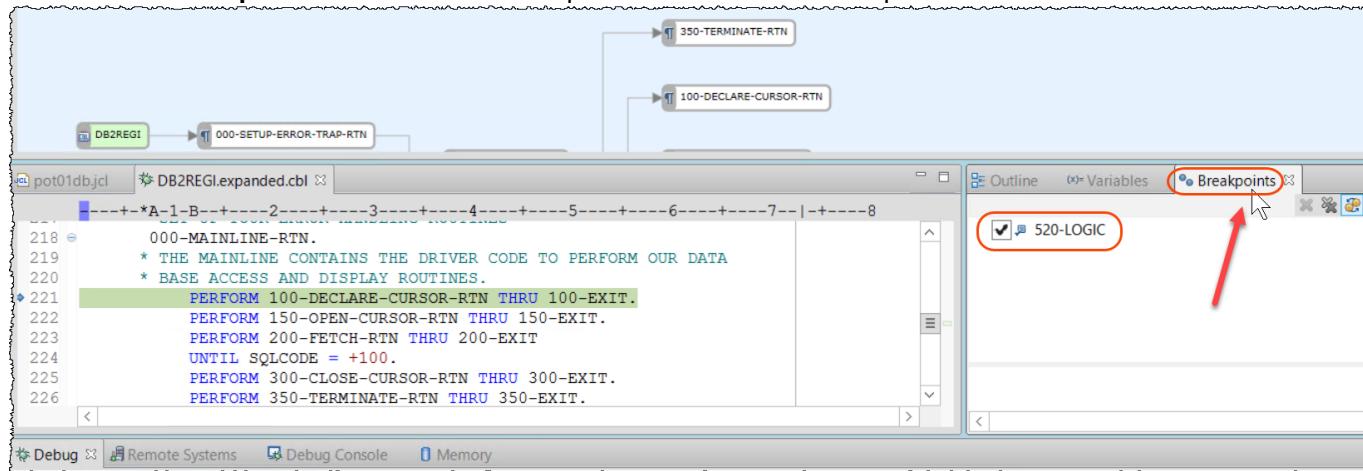
4.1.8 The execution will stop at the statement that will execute

```
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?

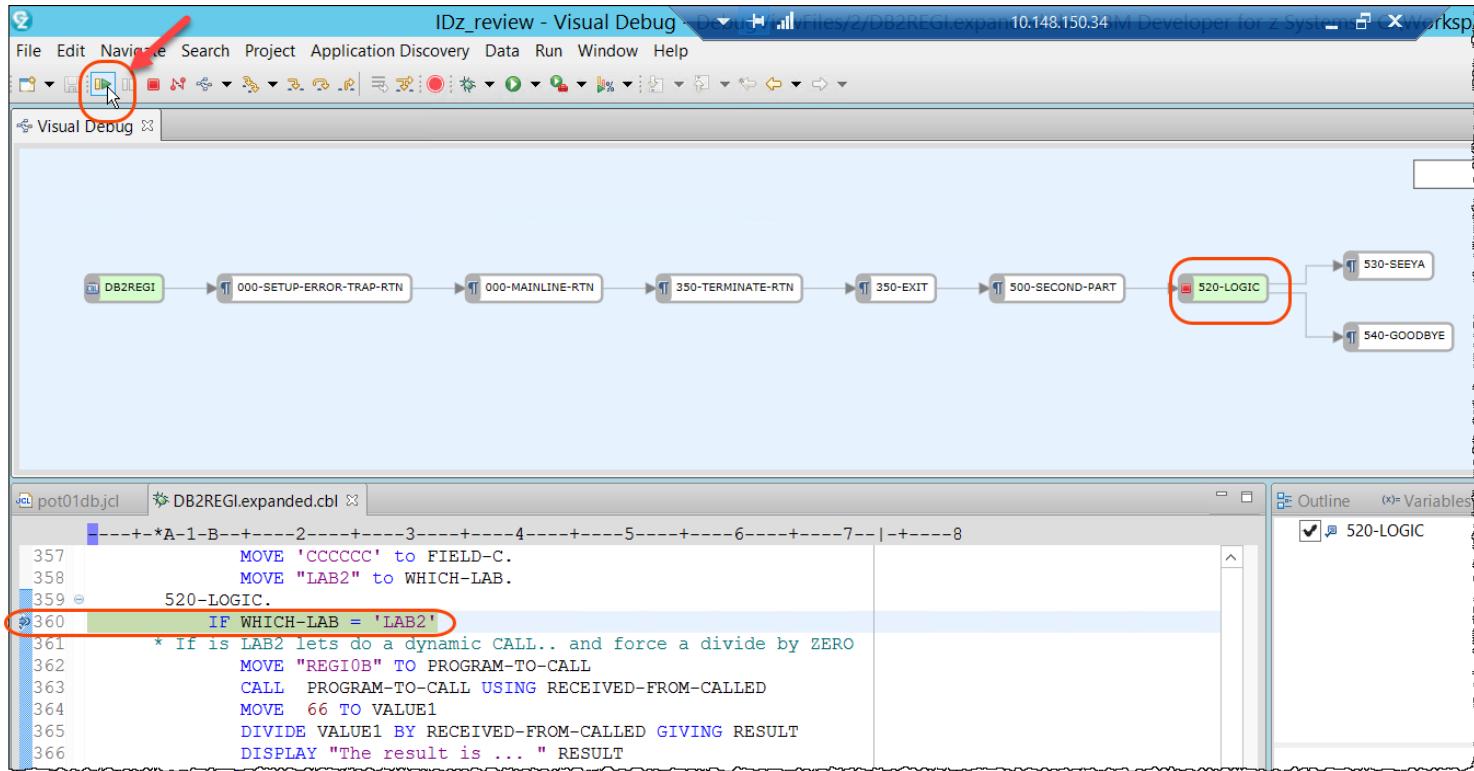


Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.

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.

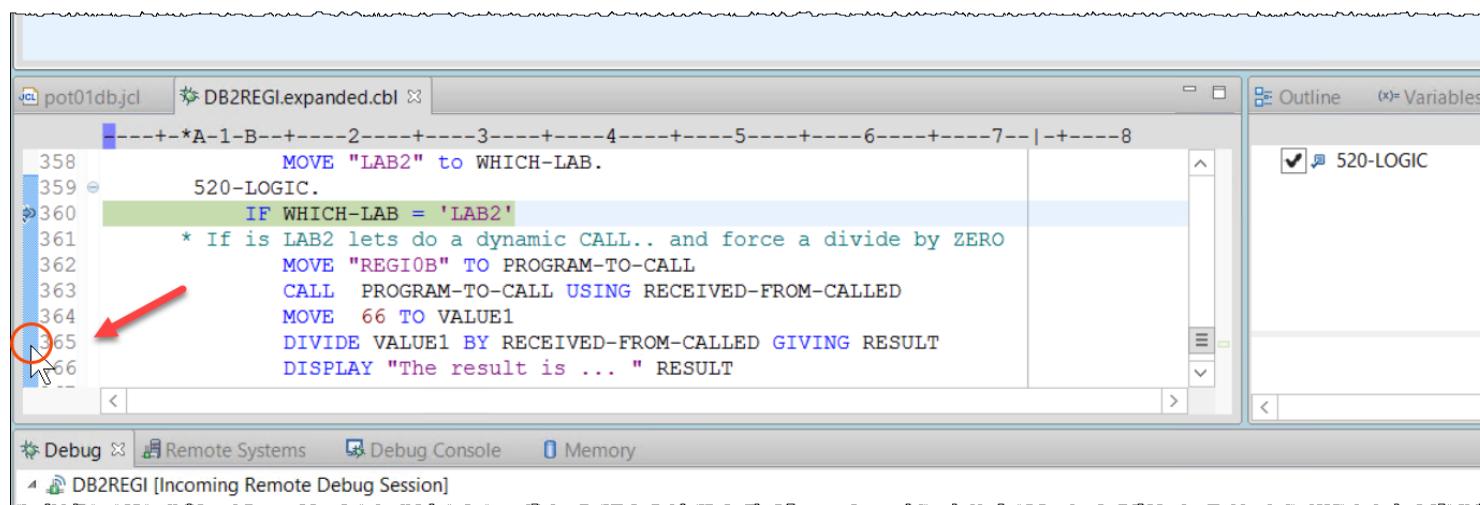
In PL/I, the stack trace represents the procedure call chain.

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.

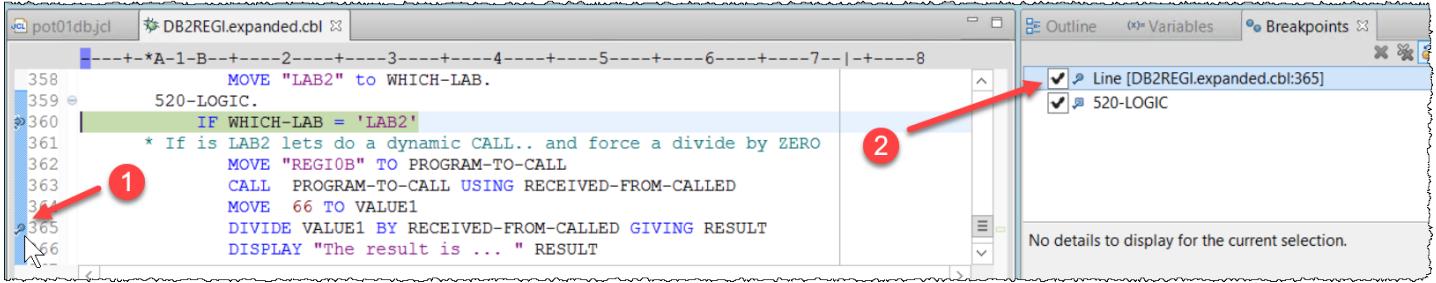


4.1.11. As you verified using Fault Analyzer, the abend occurred on line 365 where you had a divide by zero (see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.

► 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

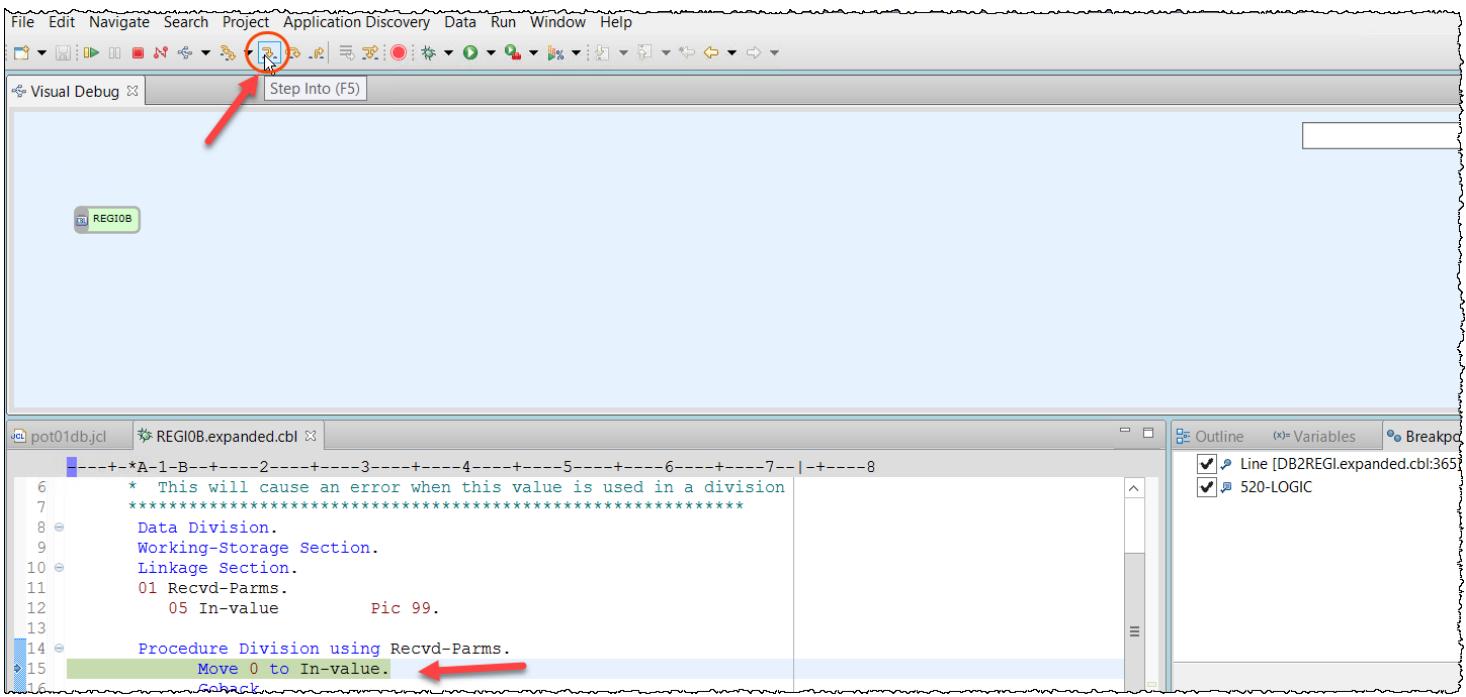


```

358      MOVE "LAB2" to WHICH-LAB.
359 520-LOGIC.
360 IF WHICH-LAB = 'LAB2'.
361 * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
362 MOVE "REG10B" TO PROGRAM-TO-CALL
363 CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364 MOVE 66 TO VALUE1
365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366 DISPLAY "The result is ... " RESULT

```

4.1.13  Continue by clicking on  or using F5 until you will see that a program named REG10B is called and this program is returning a value of 0.



```

* This will cause an error when this value is used in a division
***** Data Division.
9 Working-Storage Section.
10 Linkage Section.
11 01 Recvd-Parms.
12     05 In-value          Pic 99.
13
14 Procedure Division using Recvd-Parms.
15     Move 0 to In-value.
16

```

- 4.1.14 ① Click on or press F8.

The execution will stop at the breakpoint that you created (line 365). .

- ② 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).

```

File Edit Navigate Search Project Application Discovery Data Run Window Help
Visual Debug
File pot01db.jcl DB2REGI.expanded.cbl
----+*A-1-B---2---+---3---+---4---+---5---+---6---+---7---|---8
362 MOVE "REGIOB" TO PROGRAM-TO-CALL
363 CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364 MOVE 66 TO VALUE1
365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366 DISPLAY "The result is ... " RESULT
367 END-IF
368 IF BRANCHFLAG > 1
369 CALL 'REGI0C' USING Input-name
370 DISPLAY "BRANCHFLAG GREATER THAN 1"
371 PERFORM 530-SEEYA

```

- 4.1.15 ① Click the Variables tab (right) and ② verify that RECEIVED-FROM-CALLED is 00.

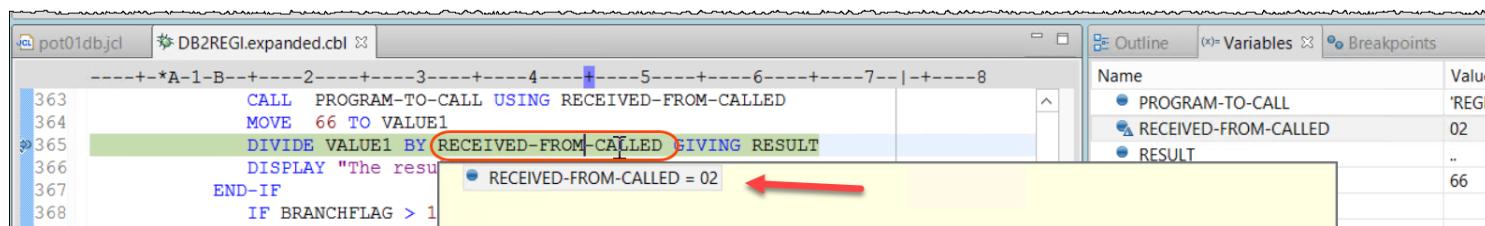
Name	Value
PROGRAM-TO-CALL	'REGIOB'
RECEIVED-FROM-CALLED	00
RESULT	..
VALUE1	66

- 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.

Name	Value
PROGRAM-TO-CALL	'REGIOB'
RECEIVED-FROM-CALLED	02
RESULT	..
VALUE1	66

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).



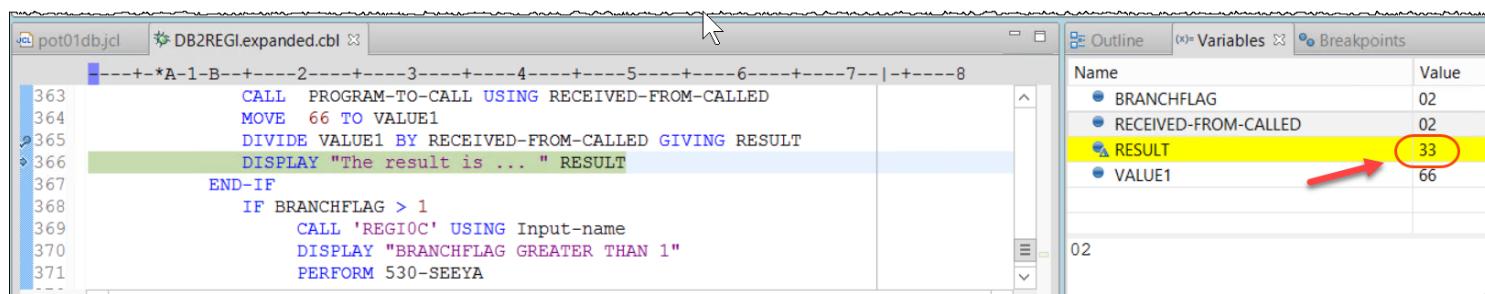
```

JCL pot01dbjcl DB2REGI.expanded.cbl
-----+-----+-----+-----+-----+-----+-----+-----+
363      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364      MOVE 66 TO VALUE1
365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366      DISPLAY "The resu
367      END-IF
368      IF BRANCHFLAG > 1
369      CALL 'REGI0C' USING Input-name
370      DISPLAY "BRANCHFLAG GREATER THAN 1"
371      PERFORM 530-SEEEYA
-----+-----+-----+-----+-----+-----+-----+-----+

```

Name	Value
PROGRAM-TO-CALL	REGI0C
RECEIVED-FROM-CALLED	02
RESULT	..
BRANCHFLAG	02
RECEIVED-FROM-CALLED	02
RESULT	33
VALUE1	66

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**.



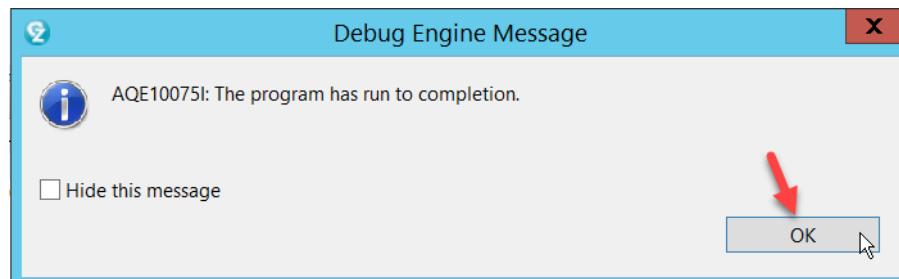
```

JCL pot01dbjcl DB2REGI.expanded.cbl
-----+-----+-----+-----+-----+-----+-----+-----+
363      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364      MOVE 66 TO VALUE1
365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366      DISPLAY "The result is ... " RESULT
367      END-IF
368      IF BRANCHFLAG > 1
369      CALL 'REGI0C' USING Input-name
370      DISPLAY "BRANCHFLAG GREATER THAN 1"
371      PERFORM 530-SEEEYA
-----+-----+-----+-----+-----+-----+-----+-----+

```

Name	Value
BRANCHFLAG	02
RECEIVED-FROM-CALLED	02
RESULT	33
VALUE1	66

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

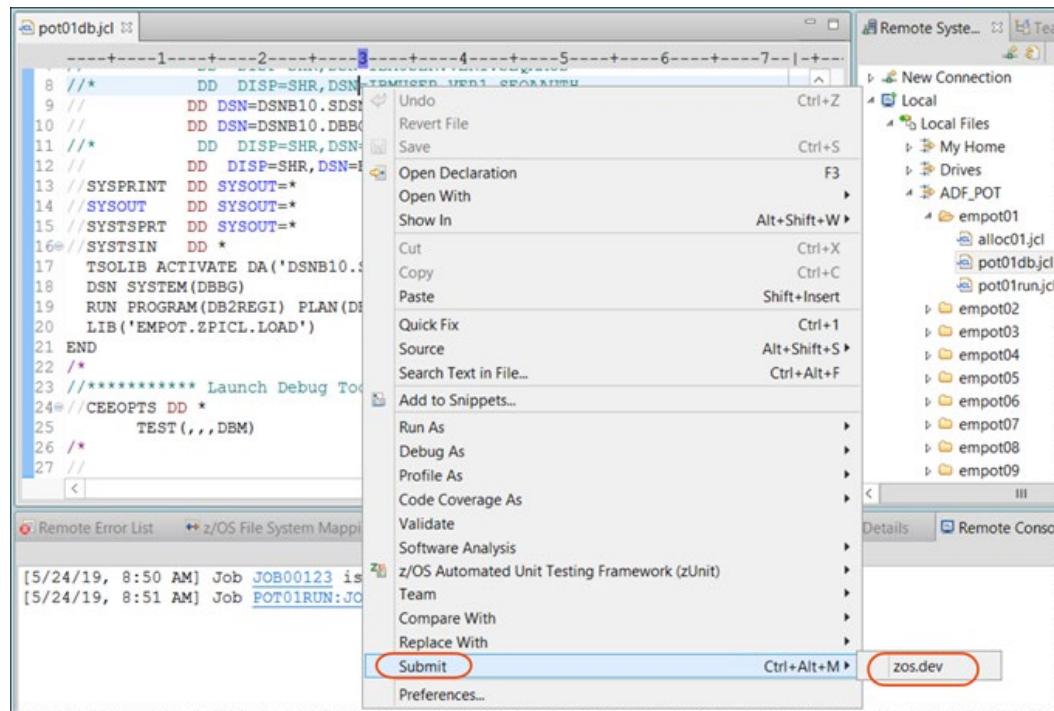
Stack pattern breakpoint



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.

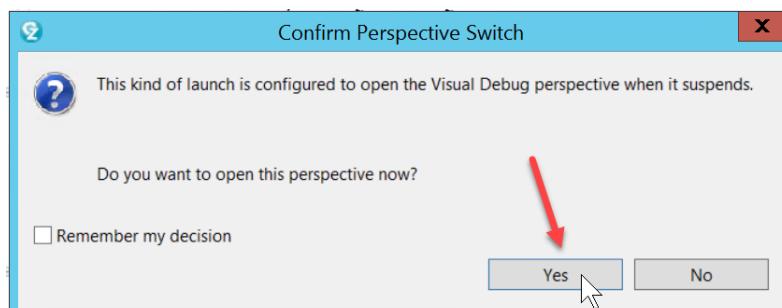
Visual debugging supports a stack pattern integration feature, which allows you to select a connected path from the program control flow diagram and set a stack pattern breakpoint using the selected path as a stack pattern.

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

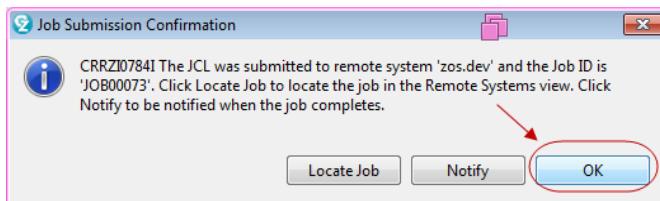


4.2.2 (Optional) Once the job starts the z/OS debug will “talk” with you.

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

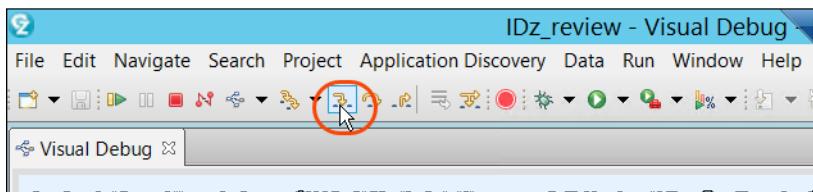


4.2.3 ► (Optional) Also click **OK** for this dialog below

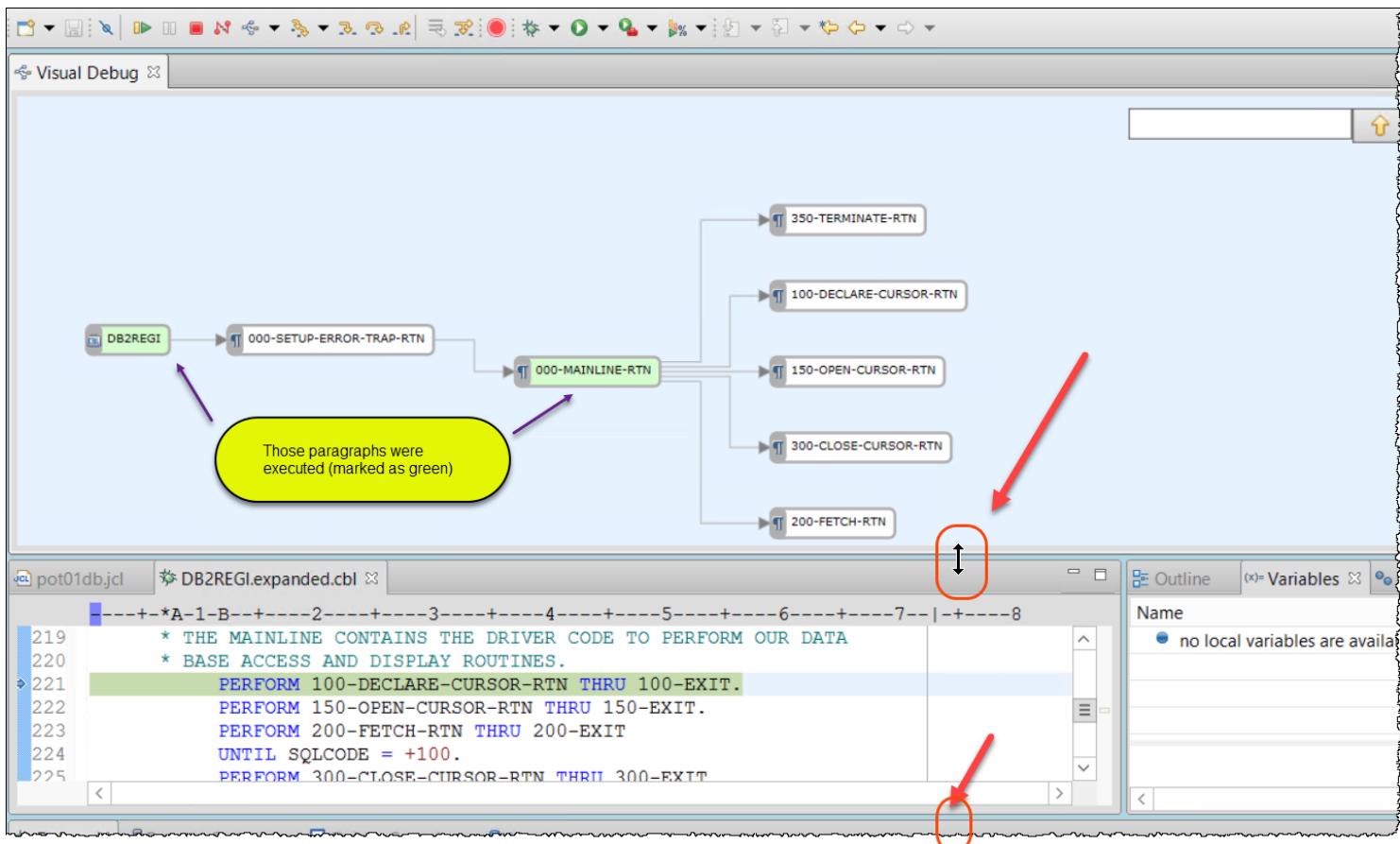


4.2.4 (Optional) Using the **Visual Debug** perspective:

► Click on icon (or Press **F5**)



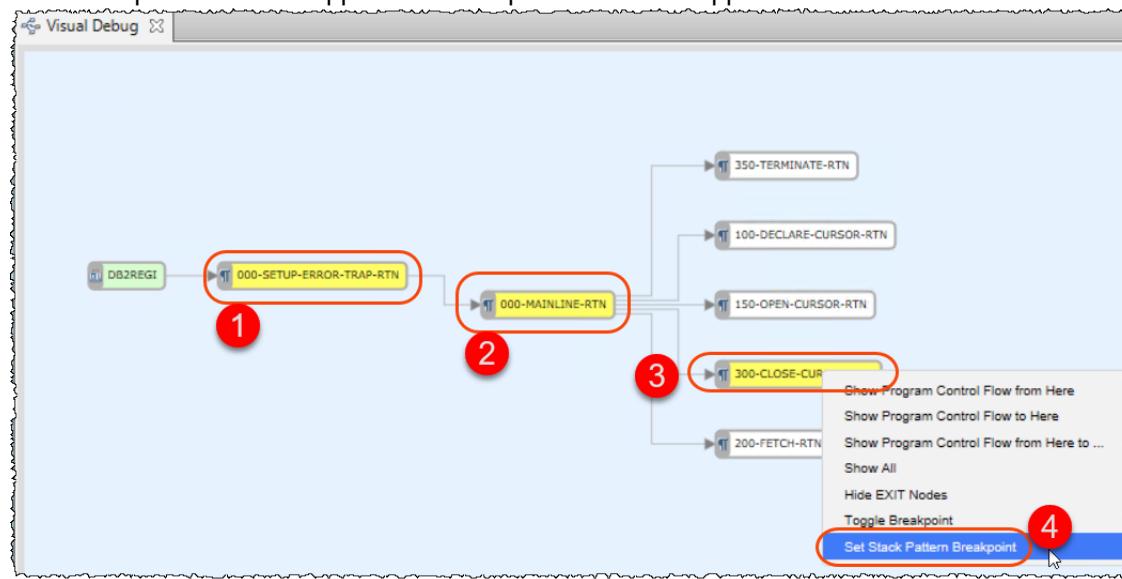
4.2.5 ► (Optional) Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green



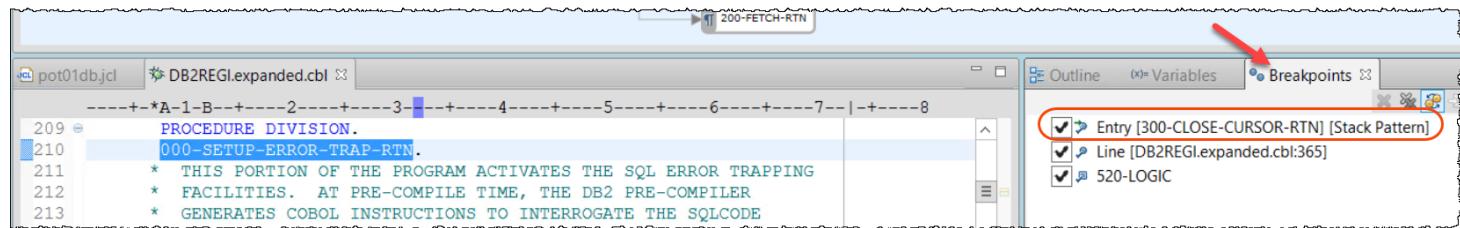
4.2.6 ► (Optional) Using the *Visual Debug* view, scroll down

► 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.

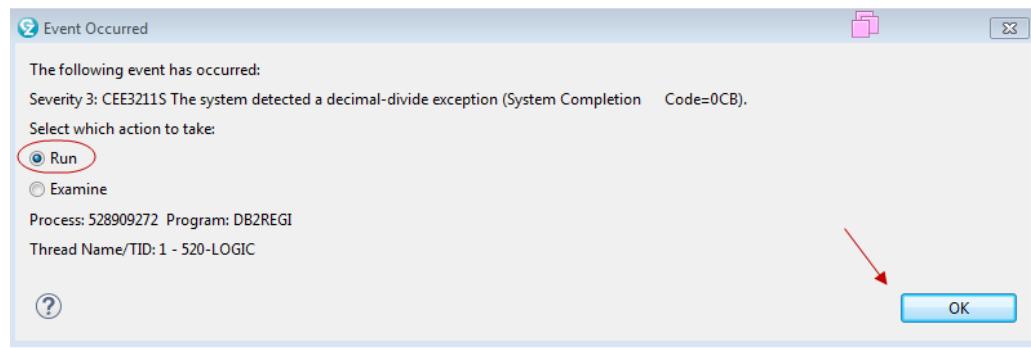


4.2.7 ► (Optional) Click on **Breakpoints** view and you will see this breakpoint created.



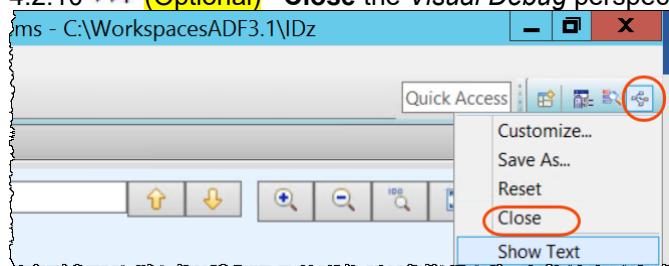
4.2.8 ► (Optional) Click on ► (or press **F8**) few times to resume the execution

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

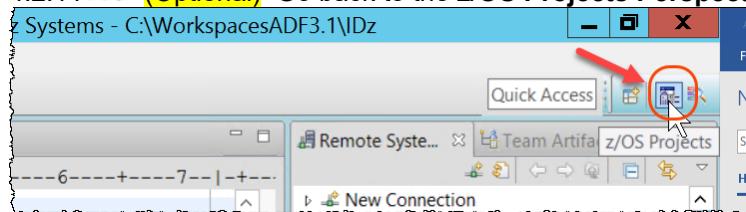


Notice – This breakpoint did not happen since the execution did not execute the 3 selected paragraphs.

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 in the top right corner.



4.3 Accessing the z/OS JES

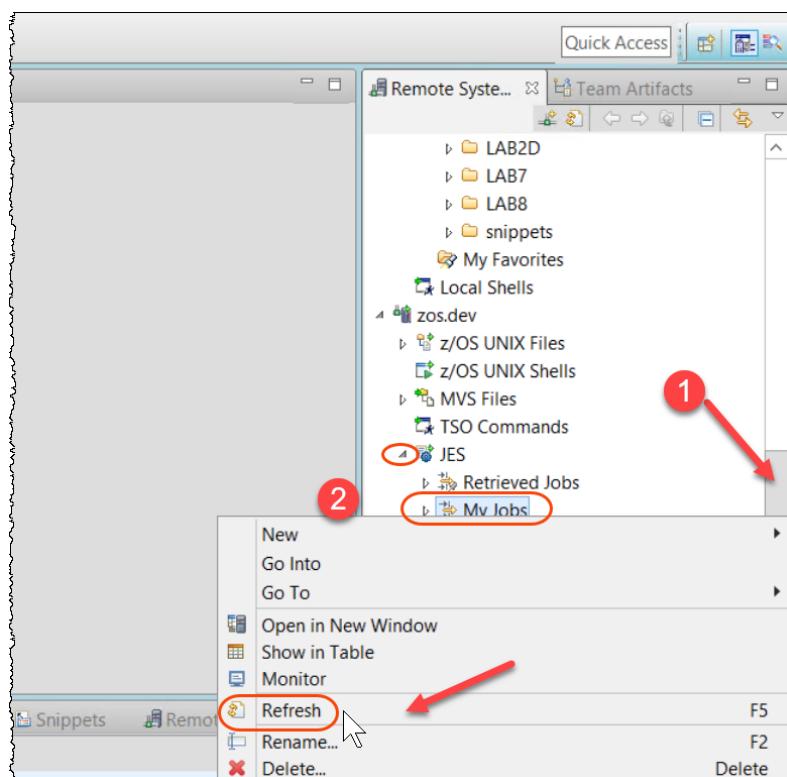
On ISPF many people use *SDSF* for this activity.

You will use the Job Monitor subsystem part of the host components.

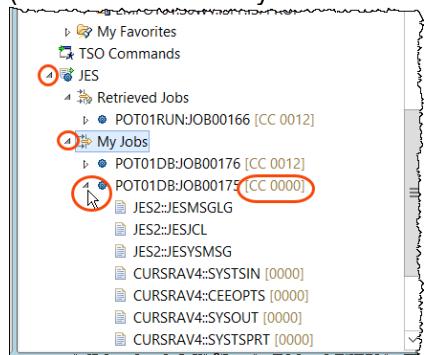
4.3.1 ► Use **Ctrl + Shift + F4** to close all opened editors. Do not save any changes.

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**

The 'Remote Systems' view shows a list of jobs. The 'POT01DB:JOB00175 [CC 0000]' entry is highlighted with a red circle. A red arrow points from this entry to the 'JOB LOG' window on the left, which displays the job log for 'POT01DB:JOB00175'. The log output includes details like job start time (MONDAY, 12 AUG 2019), system resources used, and various job steps.

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

The 'Remote Systems' view shows a list of jobs. The 'POT01DB:JOB00175 [CC 0000]' entry is highlighted with a red circle. A red arrow points from this entry to the 'JOB LOG' window on the left. The 'JOB LOG' window displays the job log for 'POT01DB:JOB00175', showing various job steps and system resource usage. A red circle labeled '2' is placed near the bottom of the log window, indicating where to scroll down to see more steps.

4.3.6 ► Close all opened editors. (**Ctrl + Shift + F4**). Click **No** if asked to save 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**.

```

DB2REGI.cbl

-----+*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+
234      500-SECOND-PART.
235      MOVE 2 TO BRANCHFLAG.
236      MOVE 'AAAAAA' to FIELD-A.
237      MOVE 'BBBBBB' to FIELD-B.
238      MOVE 'CCCCCC' to FIELD-C.
239      MOVE "LAB2" to WHICH-LAB.
240
241      520-LOGIC.
242      IF WHICH-LAB = 'LAB2'
243      * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
244      MOVE "REGI0B" TO PROGRAM-TO-CALL
245      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
246      MOVE 66 TO VALUE1
247      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
248      DISPLAY "The result is ... " RESULT
END-IF

```

A red arrow points to the dynamic call statement at line 244, which is highlighted with a red box.

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.

The screenshot shows the RAD interface with the 'z/OS Projects' view on the left and the 'DB2REGI.cbl' editor on the right. The editor displays the COBOL source code for the DB2REGI program. A red arrow points from the 'z/OS Projects' view to the 'DB2REGI.cbl' editor window.

```

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 DEPT.
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.

```

z/OS Projects : LAB1
  COBOL_DB2 [zos.dev]
    EMPOT01.POT.COBOL(DB2REGI).cbl

DB2REGI.cbl :
-----+---1-B---+---2---+---3---+---4---+---5---+---6---+---7---+
85      03 FIELD-C          PIC X(6).
86      03 WHICH-LAB        PIC X(4).
87      03 RESULT            PIC 99.
88      03 BRANCHFLAG       PIC 99.

89
90
91      PROCEDURE DIVISION.
92      000-SETUP-ERROR-TRAP-RTN.
93      * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPING
94      * FACILITIES. AT PRE-COMPILATION TIME, THE DB2 PRE-COMPILER
95      * GENERATES COBOL INSTRUCTIONS TO INTERROGATE THE SQLCODE
96      * (RETURN CODE) FROM EACH CALL. IF A SQLERROR CONDITION IS
97      * DETECTED (NEGATIVE RETURN CODE), EXECUTION WILL BRANCH TO THE
98      * 999-ERROR-TRAP-RTN TO DISPLAY AN APPROPRIATE ERROR MSG.
99      * SET UP YOUR ERROR HANDLING ROUTINES
100     000-MAINLINE-RTN.
101     * THE MAINLINE CONTAINS THE DRIVER CODE TO PERFORM OUR DATA
102     * BASE ACCESS AND DISPLAY ROUTINES.
103     PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT.

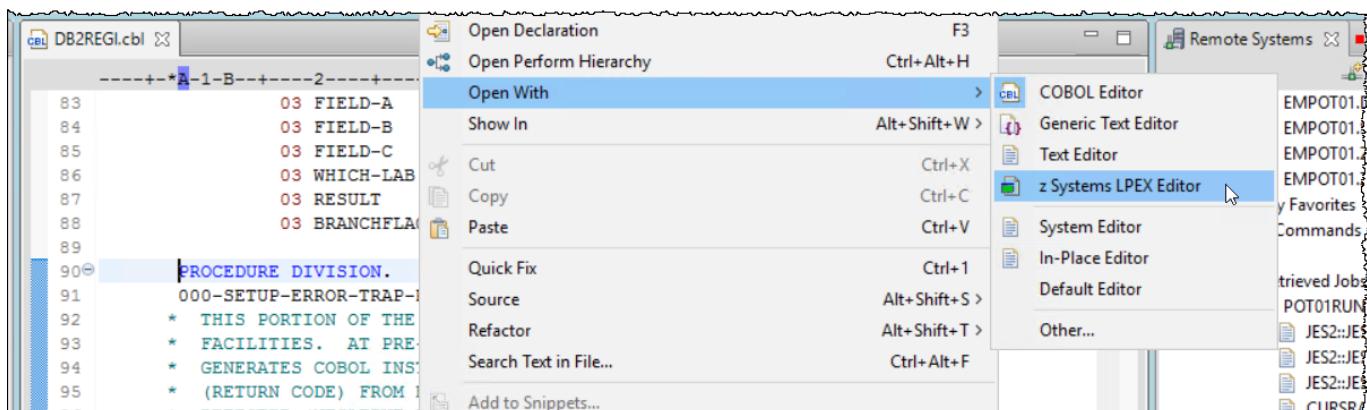
PROGRAM: DB2REGI
  IDENTIFICATION DIVISION.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  PROCEDURE DIVISION.

Properties Outline Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote Systems
Name Description Last Edit
zos.dev
LAB1_Remote_COBOL_Property_Screen used for LAB1_COBOL.wit May 28, 2019, 8:39:26 AM

```

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 :
-----+---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
000091      PROCEDURE DIVISION.
000092      000-SETUP-ERROR-TRAP-RTN.
000093      * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPING

```

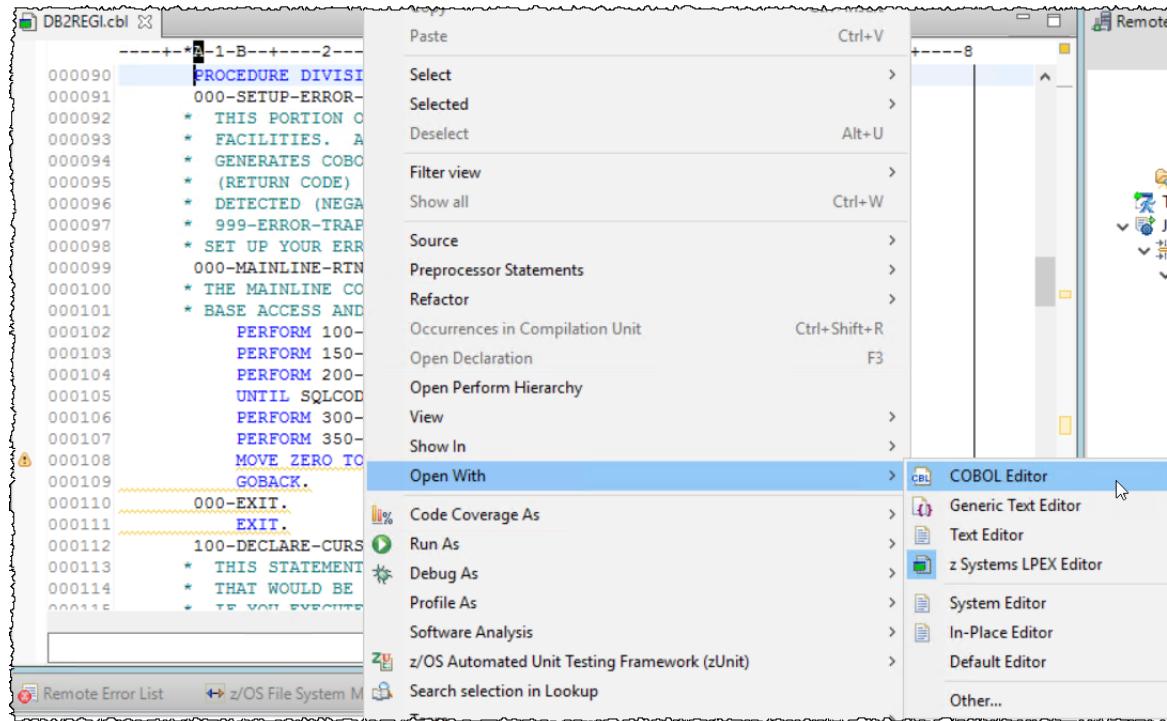
5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)

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

Click **No** for saving changes.

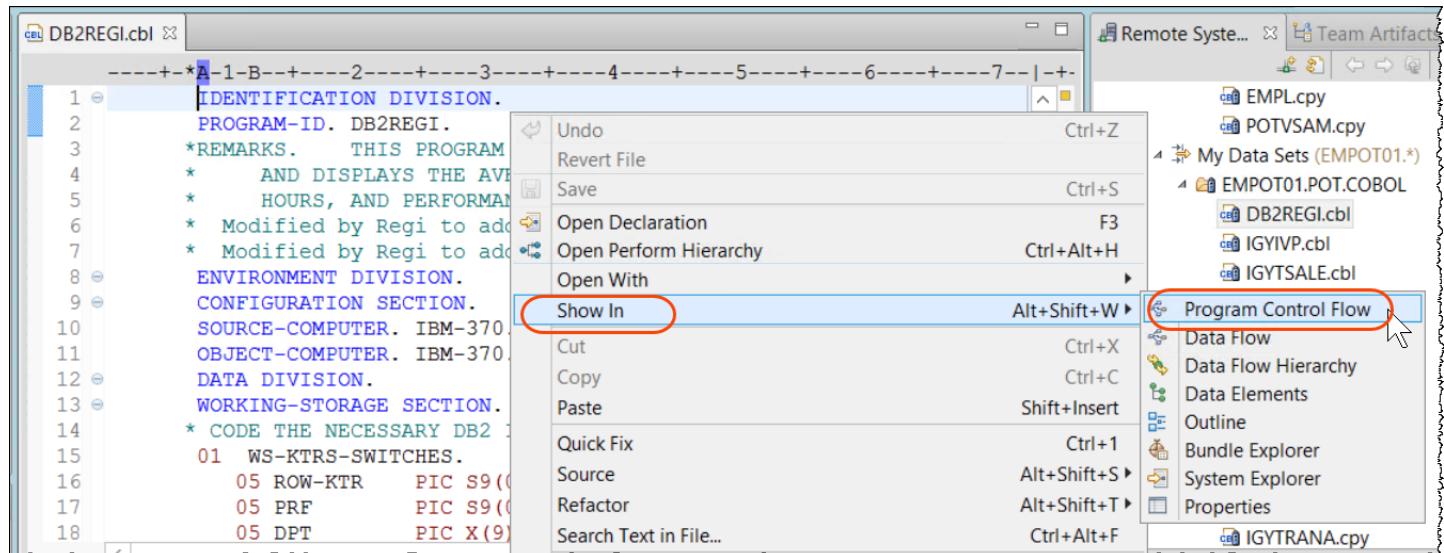
This editor is more like the Java Editor and gives you more capabilities. When you see all advantages, probably you will prefer this editor than the ISPF like editor (LPEX)..

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



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

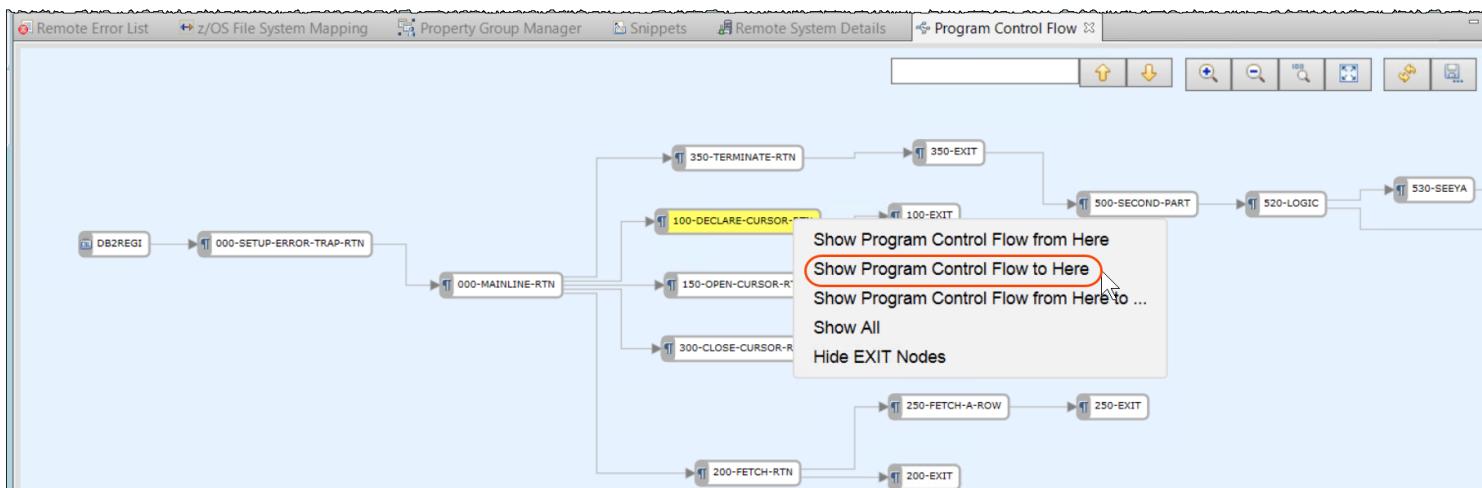
► Right click on the editor and select **Show In → Program Control Flow**



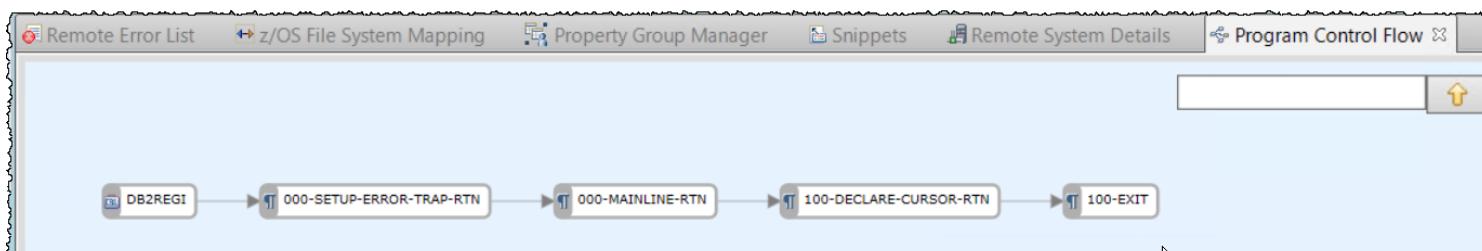
5.1.7 See the results under *Program Control Flow* view.

► 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**.



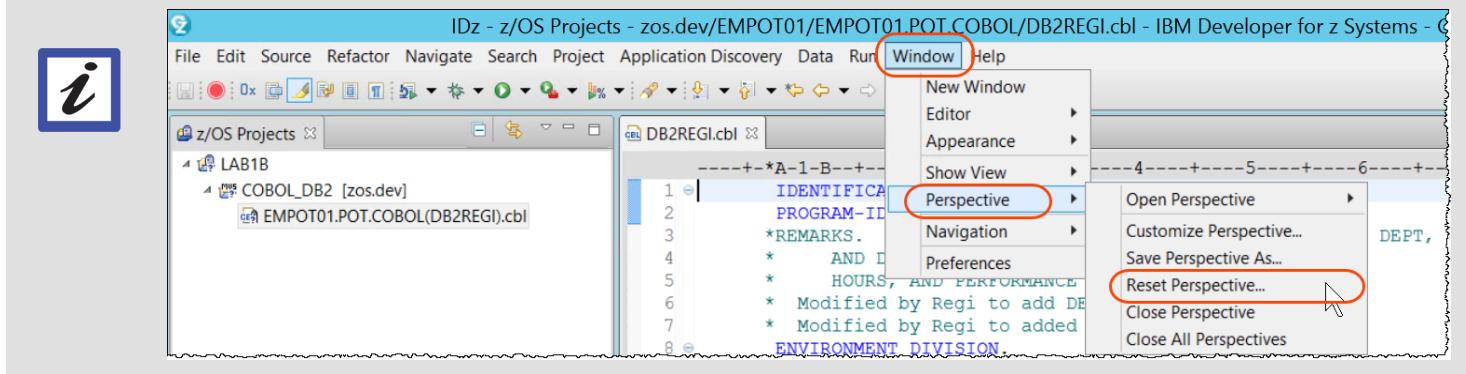
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



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.

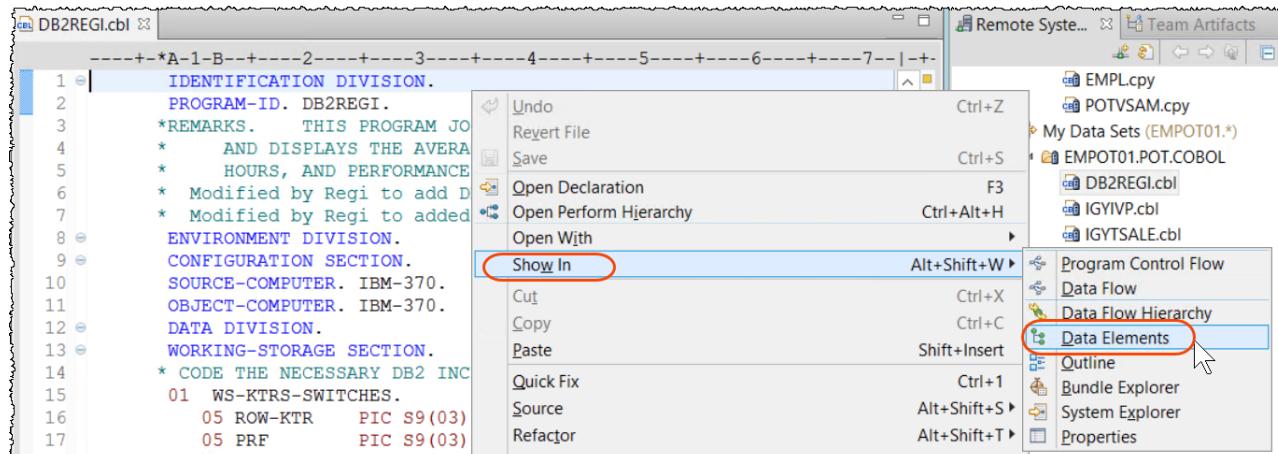
Just select **Windows > Perspective > Reset Perspective...** as below:



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

Name	Level	Top-level Item	Declaration	Declared In	Line Number	References	Item Type
PRF	5	WS-KTRS-SWITCHES	PIC S9(03)	DB2REGI.cbl	17	0	Data
PROGRAM-TO-CALL	3	WORK-FIELDS	PIC X(07)	DB2REGI.cbl	80	2	Data
PROJ	10	EMPL	PIC X(2)	EMPLcpy	46	0	Data
RECEIVED-FROM-CALLED	3	WORK-FIELDS	PIC 99	DB2REGI.cbl	81	2	Data
RESULT			PIC 99	DB2REGI.cbl	87	2	Data
ROW-KTR			PIC S9(03)	DB2REGI.cbl	16	3	Data
ROW-MSG				DB2REGI.cbl	61	1	Data
ROW-STAT			PIC Z99	DB2REGI.cbl	64	1	Data

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.

```

241      IF WHICH-LAB = 'LAB2'
242      * If is LAB2 lets do a dynamic CALL... and force a divide by ZERO
243      MOVE "REGIOB" TO PROGRAM-TO-CALL
244      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
245      MOVE 66 TO VALUE1
246      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
247      DISPLAY "The result is ... " RESULT
248      END-IF
249      IF BRANCHFLAG > 1
250          CALL 'REGIOC' USING Input-name
251          DISPLAY "BRANCHFLAG GREATER THAN 1"
252          PERFORM 530-SEYYA
253      ELSE
254          DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
255          PERFORM 540-GOODBYE.
256      530-SEYYA.
257          DISPLAY "EXECUTED SEYYA PARAGRAPH".
258      540-GOODBYE.

```

'RECEIVED-FROM-CALLED' - 3 matches in compilation unit of 'DB2REGI.cbl'

- DB2REGI.cbl (3 matches)
 - 241: 03 RECEIVED-FROM-CALLED PIC 99.
 - 242: CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
 - 246: DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT

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.

```

DB2REGI.cbl
-----+-----+-----+-----+
78      01 WORK-FIELDS.
79          03 INPUT-NAME          PIC X(30).
80          03 PROGRAM-TO-CALL    PIC X(07).
81          03 RECEIVED-FROM-CALLED PIC 99.
82          03 VALUE1              PIC 99.
83          03 FIELD-A             PIC X(6).
84          03 FIELD-B             PIC X(6).
85          03 FIELD-C             PIC X(6).
86          03 WHICH-LAB           PIC X(4).
87          03 RESULT              PIC 99.
88          03 BRANCHFLAG          PIC 99.
89
90  PROCEDURE DIVISION.
91 000-SETUP-ERROR-TRAP-RTN.
92 * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPING
93 * FACILITIES. AT PRE-COMPILE TIME, THE DB2 PRE-COMPILER
94 * GENERATES COBOL INSTRUCTIONS TO INTERROGATE THE SQLCODE
95 * (RETURN CODE) FROM EACH CALL. IF A SQLERROR CONDITION IS
96 * DETECTED (NEGATIVE RETURN CODE), EXECUTION WILL BRANCH TO THE
97 * 999-ERROR-TRAP-RTN TO DISPLAY AN APPROPRIATE ERROR MSG.
98 * SET UP YOUR ERROR HANDLING ROUTINES
99 000-MAINLINE-RTN.
100 * THE MAINLINE CONTAINS THE DRIVER CODE TO PERFORM OUR DATA
101 * BASE ACCESS AND DISPLAY ROUTINES.
102     PRRDRBY... 100  PRRDRBY... 100  PRRDRBY...
103

```

Remote Error... z/OS File Syst... Property Grou... Snippets Remote Syste... Console Remote Cons... Program Cont...

'RECEIVED-FROM-CALLED' - 3 matches in compilation unit of 'DB2REGI.cbl'

DB2REGI.cbl (3 matches)

81:03 RECEIVED-FROM-CALLED PIC 99.

244: CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED

246: DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT

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).

```

DB2REGI.cbl
-----+-----+-----+-----+
78      01 WORK-FIELDS.
79          03 INPUT-NAME          PIC X(30).
80          03 PROGRAM-TO-CALL    PIC X(07).
81          03 RECEIVED-FROM-CALLED PIC 99.
82          03 VALUE1              PIC 99.
83          03 FIELD-A             PIC X(6).
84          03 FIELD-B             PIC X(6).
85          03 FIELD-C             PIC X(6).
86          03 WHICH-LAB           PIC X(4).
87          03 RESULT              PIC 99.
88          03 BRANCHFLAG          PIC 99.
89
90  PROCEDURE DIVISION.
91 000-SETUP-ERROR-TRAP-RTN.
92 * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPING
93 * FACILITIES. AT PRE-COMPILE TIME, THE DB2 PRE-COMPILER
94 * GENERATES COBOL INSTRUCTIONS TO INTERROGATE THE SQLCODE
95 * (RETURN CODE) FROM EACH CALL. IF A SQLERROR CONDITION IS
96 * DETECTED (NEGATIVE RETURN CODE), EXECUTION WILL BRANCH TO THE
97 * 999-ERROR-TRAP-RTN TO DISPLAY AN APPROPRIATE ERROR MSG.
98 * SET UP YOUR ERROR HANDLING ROUTINES
99 000-MAINLINE-RTN.
100 * THE MAINLINE CONTAINS THE DRIVER CODE TO PERFORM OUR DATA
101 * BASE ACCESS AND DISPLAY ROUTINES.
102     PRRDRBY... 100  PRRDRBY... 100  PRRDRBY...
103

```

DB2REGI.cbl (3 matches)

81:03 RECEIVED-FROM-CALLED PIC 99.

244: CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED

246: DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT

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.

The screenshot shows the Rational Developer for System z interface. At the top is the DB2REGI.cbl editor window displaying COBOL code. In the middle is the Data view, which contains a diagram of data structures. A red arrow points from the cursor in the editor to a specific node in the Data view, indicating a connection between the code and the data model.

```

240      520-LOGIC.
241      IF WHICH-LAB = 'LAB2'
242      * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
243      MOVE "REGI0B" TO PROGRAM-TO-CALL
244      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
245      MOVE 66 TO VALUE1
246      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
247      DISPLAY "The result is ... " RESULT
248      END-IF
249      IF BRANCHFLAG > 1
250          CALL 'REGI0C' USING Input-name
251          DISPLAY "BRANCHFLAG GREATER THAN 1"

```

5.1.15 ► Close all opened editors if still opened. (**CTRL + Shift + F4**)

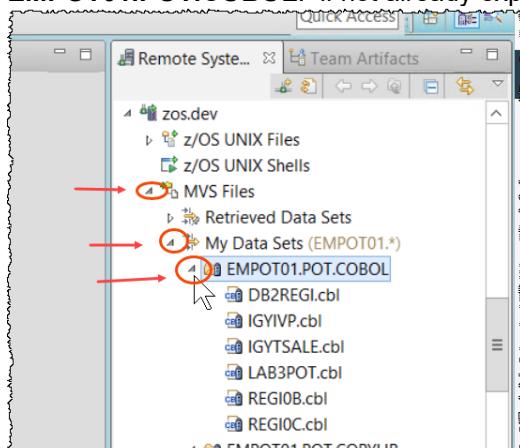
Or just click on the of each opened editor.

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.

```

1 | Identification Division.
2 | Program-ID. "REGI0B".
3 |
4 * This program is called.
5 * It returns a value (0)
6 * This will cause an error when this value is used in a division
7
8 Data Division.
9 Working-Storage Section.
10 Linkage Section.
11 01 Recvd-Parms.
12     05 In-value      Pic 99.
13
14 Procedure Division using Recvd-Parms.
15     Move 0 to In-value.
16     Goback.

```

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

```

1 | Identification Division.
2 | Program-ID. "REGI0B".
3 |
4 * This program is called.
5 * It returns a value (0)
6 * This will cause an error when this value is used in a division
7
8 Data Division.
9 Working-Storage Section.
10 Linkage Section.
11 01 Recvd-Parms.
12     05 In-value      Pic 99.
13
14 Procedure Division using Recvd-Parms.
15     Move 0 to In-value.
16     Goback.

```

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)

```

Linkage Section.
01 Recvd-Parms.
    05 In-value      Pic 99.

Procedure Division using Recvd-Parms.
    Move 0 to In-value.
    m
    G
    REC MERGE
    REC MOVE
    REC MULTIPLY
    REC MULTIPLY - NOT ON SIZE ERROR - END-MULTIPLY
    REC MULTIPLY - ON SIZE ERROR - END-MULTIPLY
    REC MULTIPLY - ON SIZE ERROR - NOT ON SIZE ERROR - END-M

```

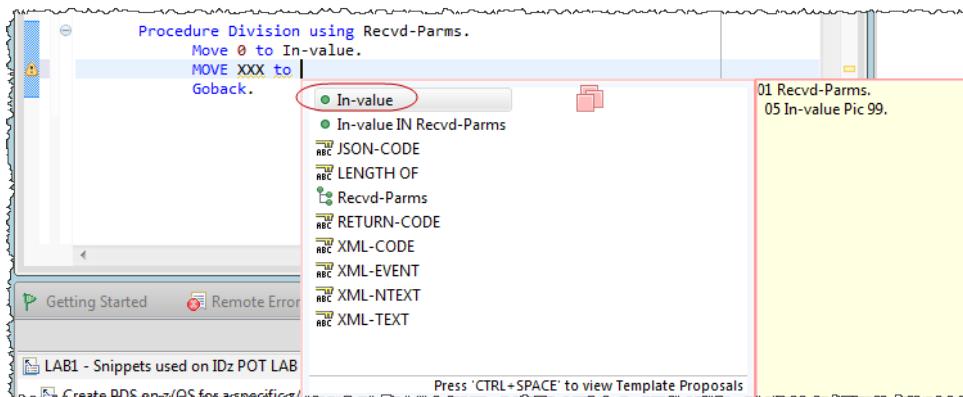
5.2.8 ► Type "XXX to"

```

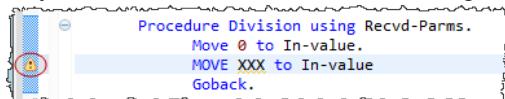
Procedure Division using Recvd-Parms.
    Move 0 to In-value.
    MOVE XXX to
    Goback.

```

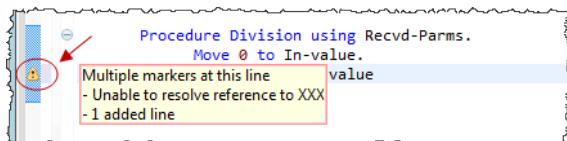
5.2.9 ►| Press **Ctrl + Space** to list the variables and select **In-value**.
 Notice that variables could be defined in copybooks.



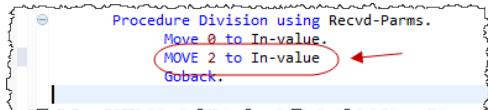
A yellow mark shows that something is wrong.



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.



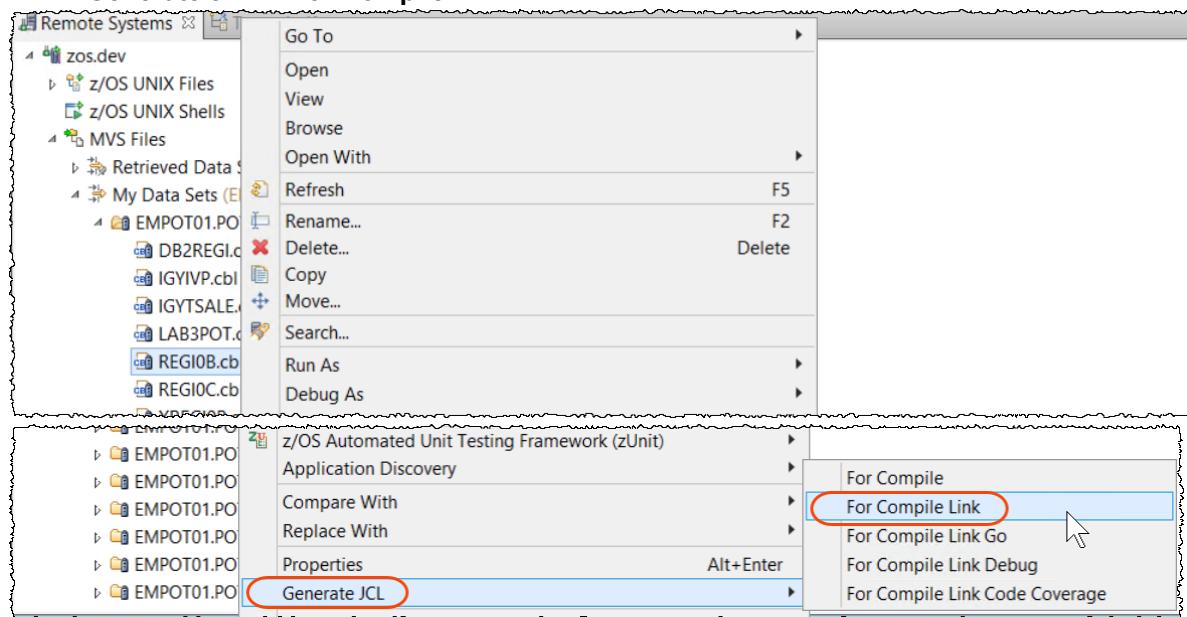
5.3 Compile and link REGI0B.

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.

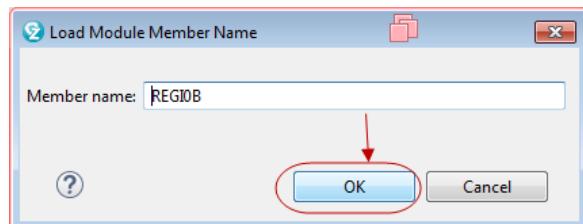
5.3.1 ►| Click **Ctrl + Shift + F4** to close the editors. Or just left click on the of each opened editor.
 ►| Click **Save** to save the changes.



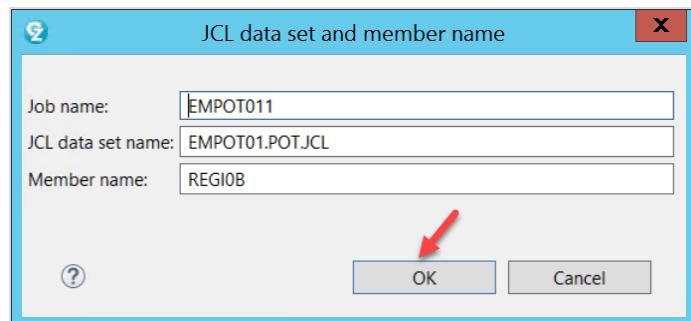
5.3.2 ► Using the Remote Systems view right click on **REGI0B.cbl** and select **Generate JCL > For Compile Link**.



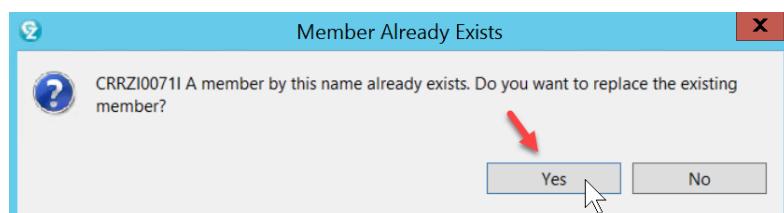
5.3.3 ► Accept the default member name and click **OK**



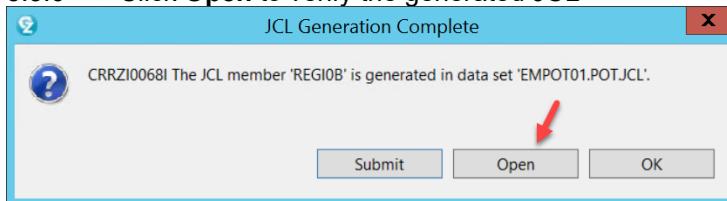
5.3.4 ► Accept the default and click **OK**



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



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



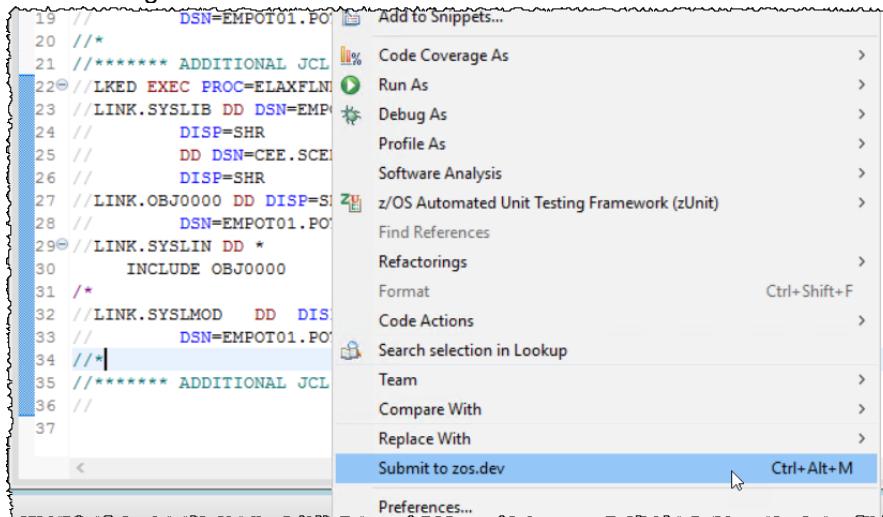
5.3.7 ► Scroll down and notice that the *Load module* will be created on the PDS: **EMPOT01 . POT . LOAD**.

```

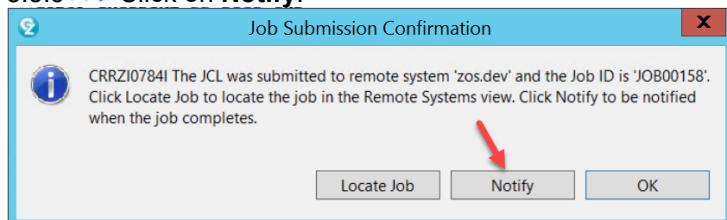
14 //      DD DISP=SHR,
15 //      DSN=EMPOT.ZPICL.COPYLIB
16 //COBOL.SYSXMLSD DD DUMMY
17 //COBOL.SYSIN DD DISP=SHR,
18 //      DSN=EMPOT01.POT.COBOL (REGI0B)
19 //*
20 //***** ADDITIONAL JCL FOR COMPILE HERE *****
21 //LKED EXEC PROC=ELAXFLNK
22 //LINK.SYSLIB DD DSN=EMPOT01.POT.OBJ,
23 //      DISP=SHR
24 //      DD DSN=CEE.SCEELKED,
25 //      DISP=SHR
26 //LINK.OBJ0000 DD DISP=SHR,
27 //      DSN=EMPOT01.POT.OBJ (REGI0B)
28 //LINK.SYSLIN DD *
29     INCLUDE OBJ0000
30 /*
31 //LINK.SYSLMOD DD DISP=SHR,
32 //      DSN=EMPOT01.POT.LOAD (REGI0B)
33 //*
34 //***** ADDITIONAL JCL FOR LINK HERE *****
35 /

```

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**

```

20 //***** ADDITIONAL JCL FOR COMPILE HERE *****
21 //LKED EXEC PROC=ELAXFLNK
22 //LINK.SYSLIB DD DSN=EMPOT01.POT.OBJ,
   DTCP-SUP
  
```

[5/28/19, 10:34 AM] Job **JOB00158** is being submitted
[5/28/19, 10:34 AM] Job **EMPOT01:JOB00158** ended with completion code CC 0000

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**.

```

298 SAVE OPERATION SUMMARY:
299
300 MEMBER NAME      REGI0B
301 LOAD LIBRARY    EMPOT01.POT LOAD
302 PROGRAM TYPE    PROGRAM OBJECT (FORMAT 3)
303 VOLUME SERIAL   C2DBAR
304 DISPOSITION     ADDED NEW
305 TIME OF SAVE    09.33.57 MAY 28, 2019
306
307
308 SAVE MODULE ATTRIBUTES:
309
310 AC             000
311 AMODE          31
312 COMPRESSION    NONE
313 DC             NO
314 EDITABLE        YES
315 EXCEEDS 16MB   NO
316 EXECUTABLE     YES
317 LONGPARM       NO
318 MIGRATABLE     NO
  
```

REGI0B is now on
EMPOT01.POT LOAD

Team Artifacts browser:

- EMPOT01.POT.PLI
- EMPOT01.POT.PLI.LISTING
- EMPOT01.POT.SP2.COBOL
- EMPOT01.SOW1.ISPF.ISPPROF
- EMPOT01.D108.T1125224.POT01D
- EMPOT01.D144.T1029076.POT01D
- EMPOT01.D144.T1045268.POT01D
- EMPOT01.D144.T1448006.POT01D
- EMPOT01.D144.T1452131.POT01D
- My Favorites
- TSO Commands
- JES
- Retrieved Jobs
 - EMPOT01:JOB00158 [CC 0000]
 - JES2:JESMSGLG
 - JES2:JESJCL
 - JES2:JESYSMSG
 - LKD:LINK:SYSLIN [0000]
 - LKD:LINK:SYSPRINT [0000]
 - POT01RUN:JOB00123 [CC 0012]

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..

```

//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.SEQAUTH
// DD DSN=DSNB10.SDSNLOAD,DISP=SHR
// DD DSN=DSNB10.DBDBG.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(DBDBG)
RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
LIB('EMPOT.ZPICL.LOAD')
END
/*

```

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

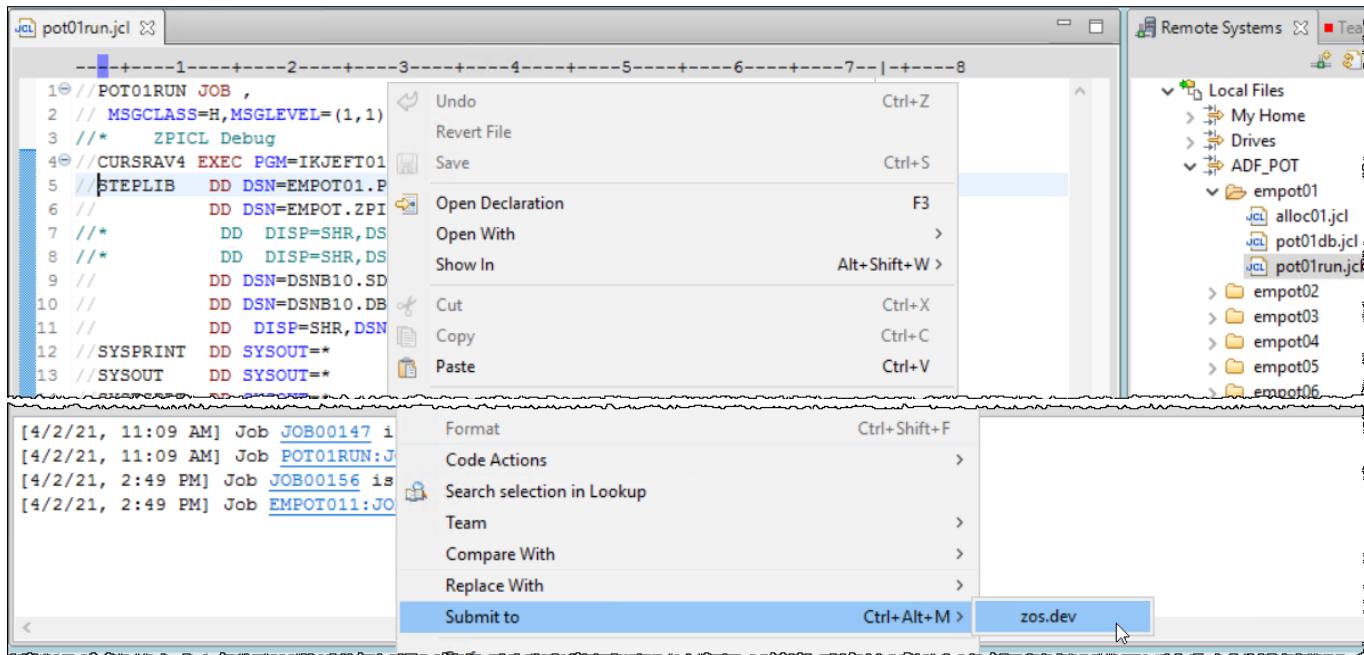
5.4.3 ► Notice that your PDS where REG10B is created is already on the STEPLIB.

```

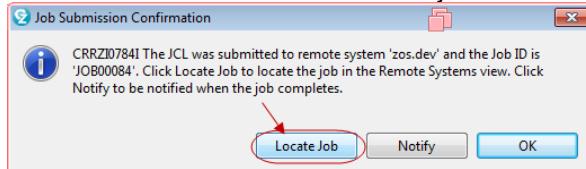
//POT01RUN JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
///* ZPICL Debug
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
// DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
///* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
///* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAUTH
// DD DSN=DSNB10.SDSNLOAD,DISP=SHR
// DD DSN=DSNB10.DBDBG.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(DBDBG)
RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
LIB('EMPOT.ZPICL.LOAD')
END
/*

```

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

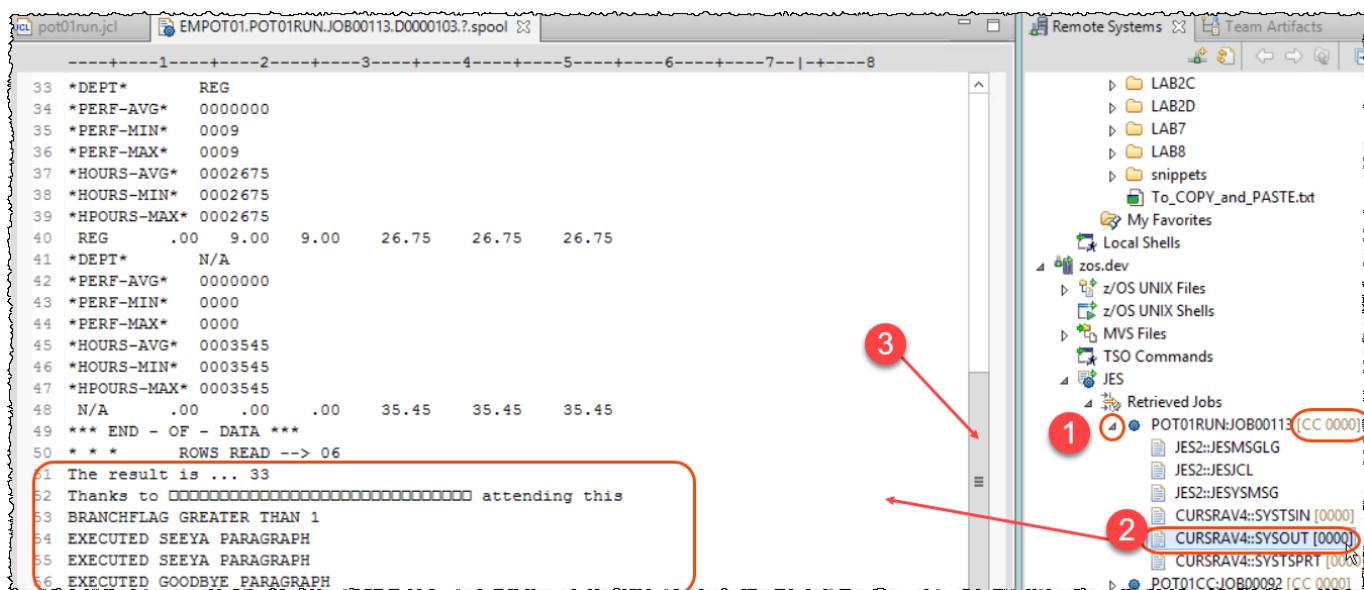


5.4.5 ► Click on Locate Job. The job submitted will be shown under JES folder under Remote Systems view



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:JOB00xxxx** (where 00xxxx 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



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.

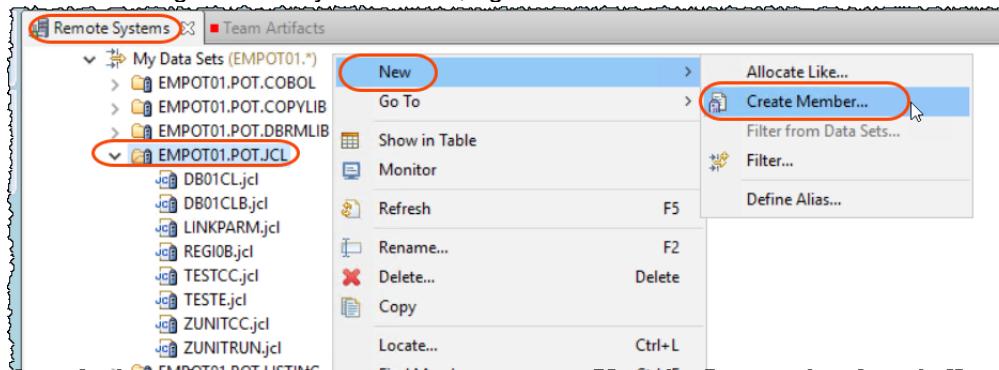
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode, A sample JCL procedure, to process a single program run, and a set of scripts to start and stop a permanently active Code coverage collector that can process multiple program runs.

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.

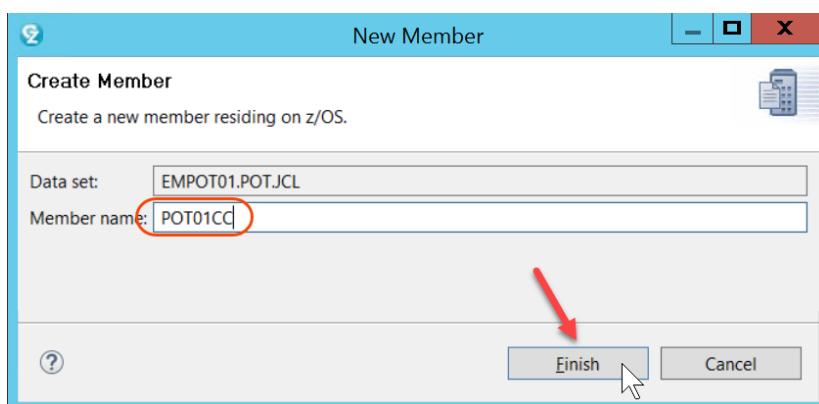
6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

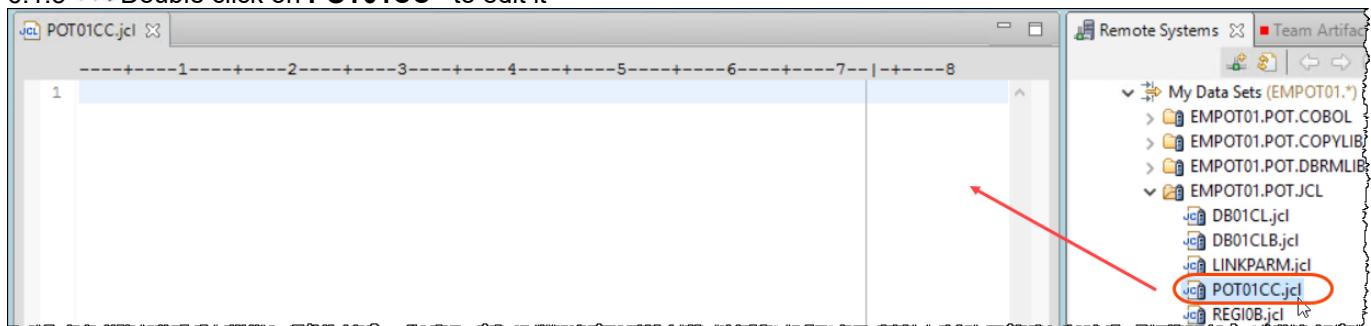
6.1.1 ► Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member...**



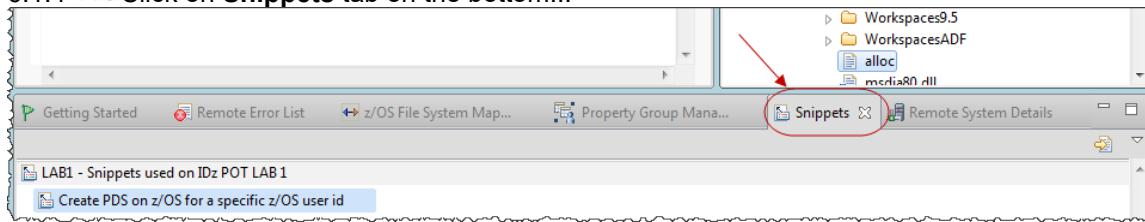
6.1.2 ► Name it as **POT01CC** and click **Finish**



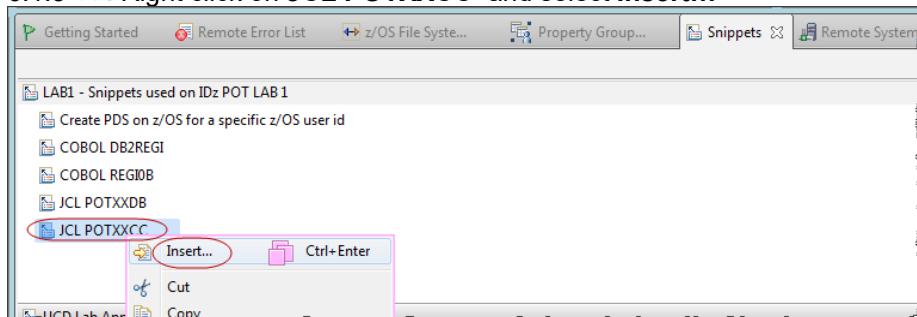
6.1.3 ► Double click on **POT01CC** to edit it



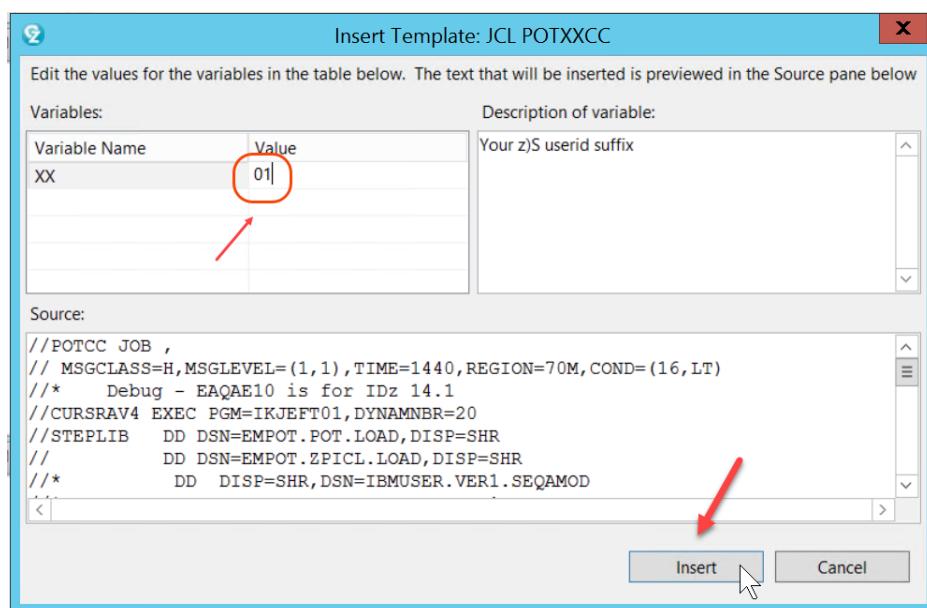
6.1.4 ► Click on **Snippets** tab on the bottom...



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**



6.1.7 . Notice that the Snippets variables will be replaced.

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

```

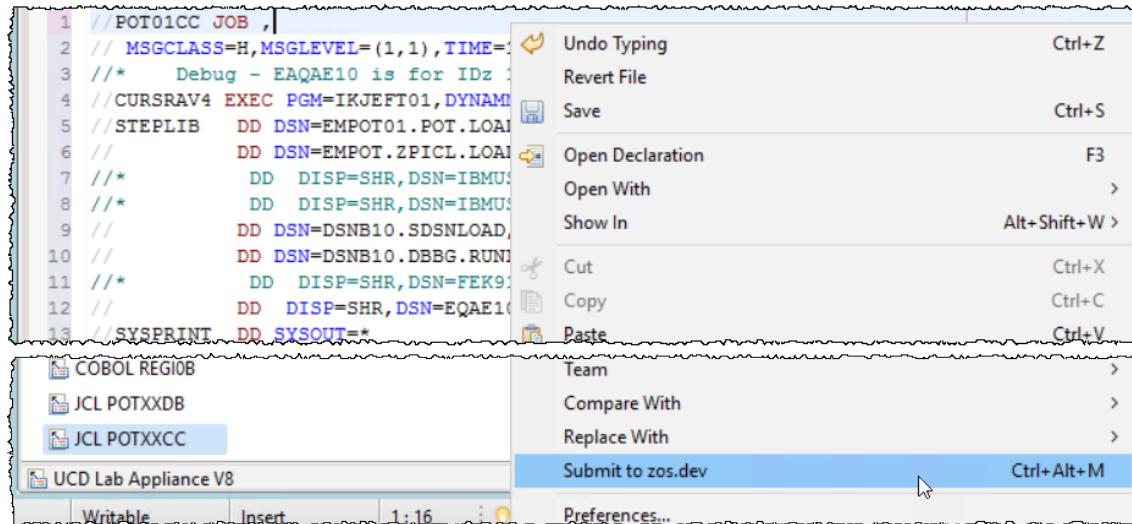
1 //POT01CC JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 /* Debug - EAQAE10 is for IDz 14.1
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=EMPOT01.POT LOAD,DISP=SHR
6 // DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
7 /**
8 /**
9 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR
10 // DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
11 /**
12 // DD DISP=SHR,DSN=FEK910.SFEKAUTH
13 //SYSPRINT DD SYSOUT=*
14 //SYSOUT DD SYSOUT=*
15 //SYSTSPRT DD SYSOUT=*
16 //SYSTSIN DD *
17 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
18 DSN SYSTEM(DBDBG)
19 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -

```

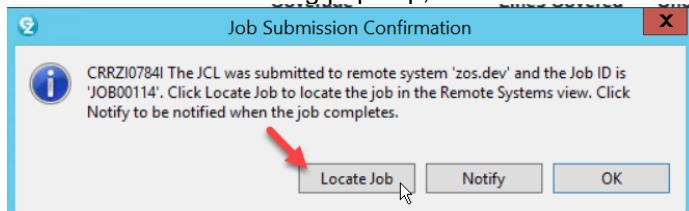
6.2 Submit the JCL for z/OS execution

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**

Code Coverage Report
Code coverage report for 'DB2REGI_2021_04_16_131035_0684', analyzed Apr 16, 2021 1:10:35 PM

Name	Coverage	Lines Covered	Uncovered Lines	Total
DB2REGI.DB2REGI.cob	69%	61	27	
DB2REGI.REGI0C.cob	100%	9	0	
REGI0B.REGI0B.cob	100%	3	0	
Summary (Elapsed time: 0.718 sec)	73%	73	27	

Off On Show below : 80 % Refresh Export Files Modules

Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote System Details **Code Coverage Results**

Name Status Coverage Level Analyzed Date Additional Info

- Compiled Code Coverage Workspace Results
 - DB2REGI_2021_04_16_131035_0684 (Status: 73% Line, Analyzed Date: Apr 16, 2021 1:10:35 PM)
- Java Code Coverage Workspace Results

6.2.4 ▶ Click on **DB2REGI.DB2REGI.cob**

Name	Coverage	Lines Covered	Uncovered Lines	Total
DB2REGI.DB2REGI.cob	69%	61	27	

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**.

```

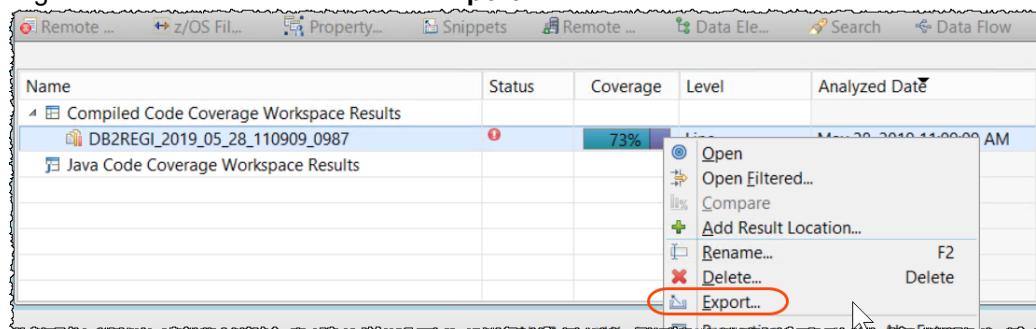
332 EXIT.
333 350-TERM-RTN.
334 MOVE ROW-KTR TO ROW-STAT.
335 DISPLAY ROW-MSG.
336 350-EXIT.
337 * EXIT.
338 GO TO 500-SECOND-PART.
339 999-ERROR-TRAP-RTN.
340 ****
341 * ERROR TRAPPING ROUTINE FOR NEGATIVE SQLCODES *
342 ****
343 Lines 343-351 not covered. **** WE HAVE A SERIOUS PROBLEM HERE ****.
344 999-ERROR-TRAP-RTN'.
345 MULTIPLY SQLCODE BY -1 GIVING SQLCODE.
346 DISPLAY 'SQLCODE ==> ' SQLCODE.
347 DISPLAY SQLCA.
348 DISPLAY SQLERRM.
349 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
350 EXEC SQL ROLLBACK WORK END-EXEC.
351 *
352 500-SECOND-PART.
353 MOVE 2 TO BRANCHFLAG.
354 MOVE 'AAAAAA' to FIELD-A.
355 MOVE 'BBBBBB' to FIELD-B.
356 MOVE 'CCCCCC' to FIELD-C.

```

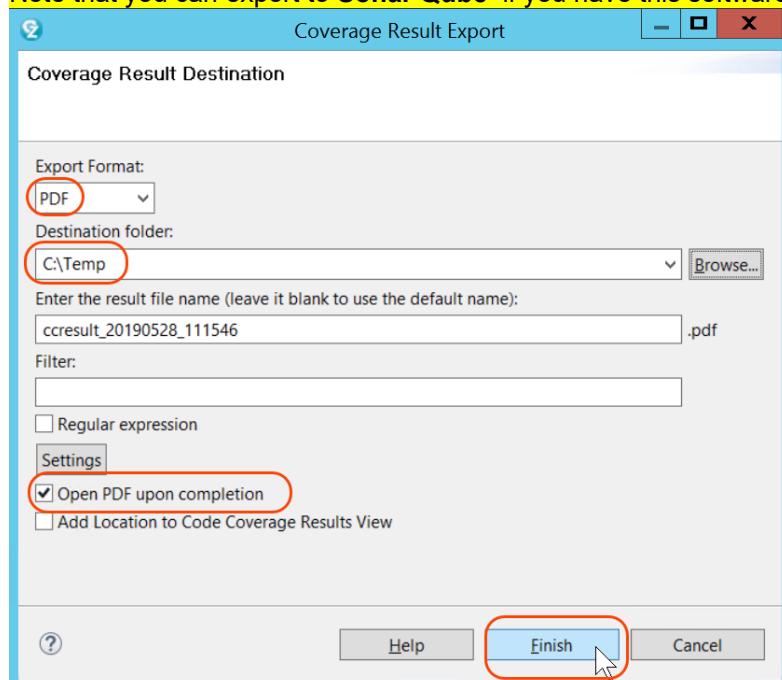
As you can see the error routine on the picture above is not tested..

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.

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.

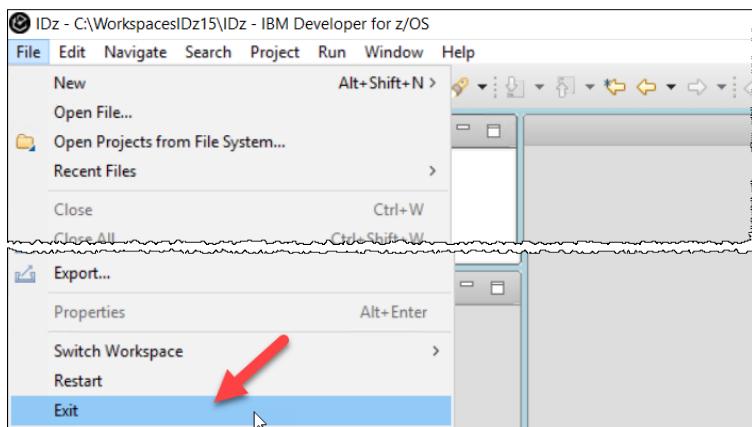
Data Studio is a foundational offering that includes support for key tasks across the data management lifecycle, including administration, application development, and query tuning.

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.

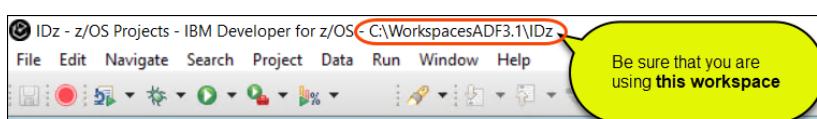
► Click on **File > Exit**



7.0.2 Start **IBM Developer for z Systems version 14**

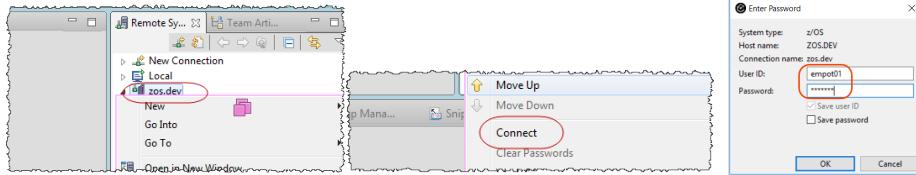
► Using the windows desktop double click on **IDz 14.2.4** icon.

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.



7.0.3 To Connect to the z/OS system:

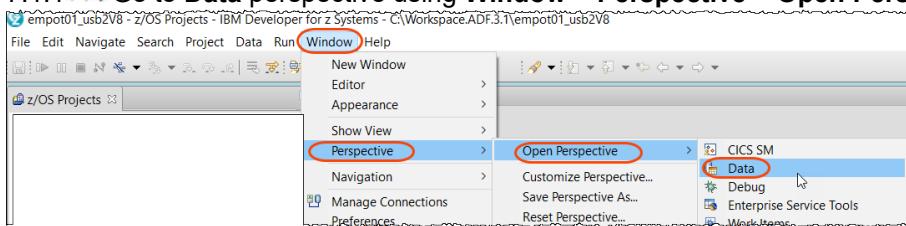
- ▶▶ Using the **Remote Systems** view, right-click on **zos.dev** and select **Connect** .
- ▶▶ Use **empot01** and password **empot01**



7.1 Creating a connection to the DB2 on z/OS

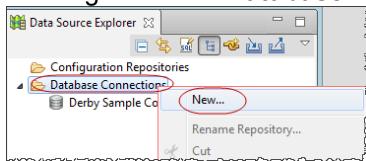
You will need to have authorization to connect to the DB2. Now you will use **IBMUSER** as a new z/OS ID

7.1.1 ▶▶ Go to Data perspective using Window > Perspective > Open Perspective > Data

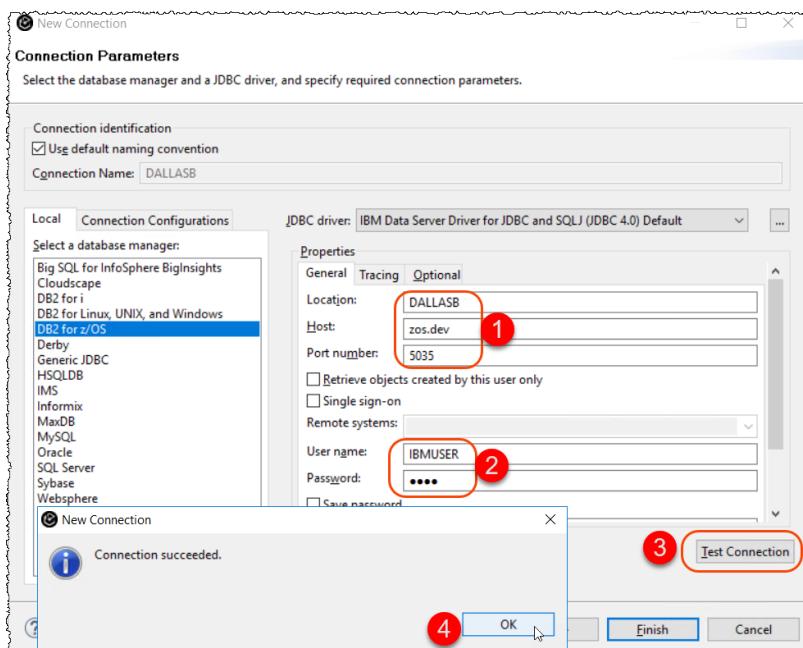


7.1.2 Using the Data Source Explorer view, create a new z/OS DB2 connection:

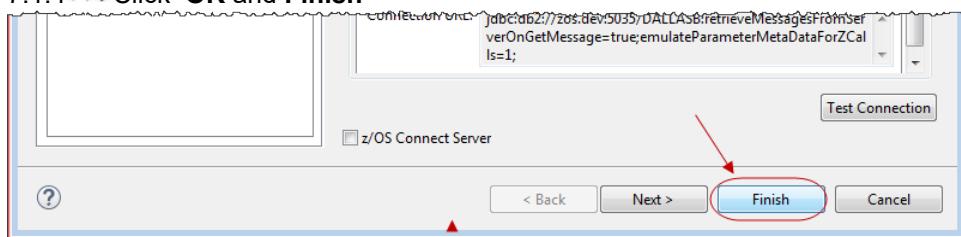
- ▶▶ Right click on **Database Connections** and select **New..**



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.

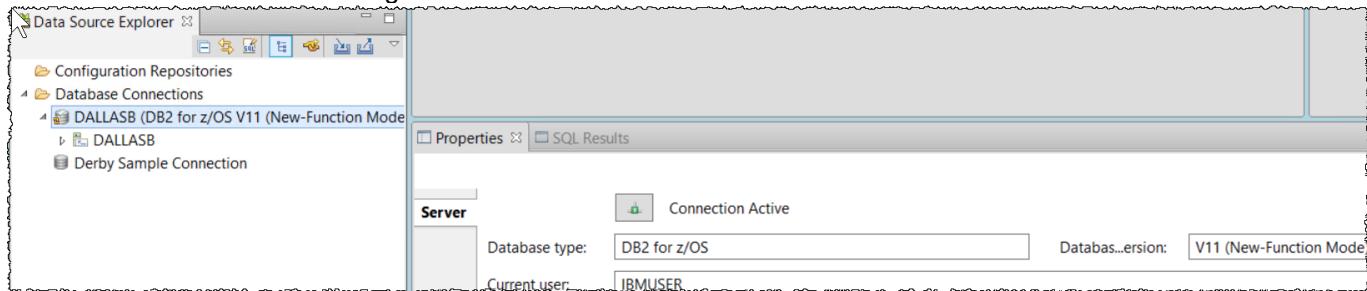


7.1.4 ► Click OK and Finish



The connection is created. You could see tables, modify contents etc..

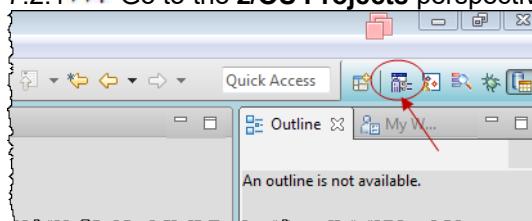
But we want to focus on the integration of DB2 and the COBOL/DB2 editor.



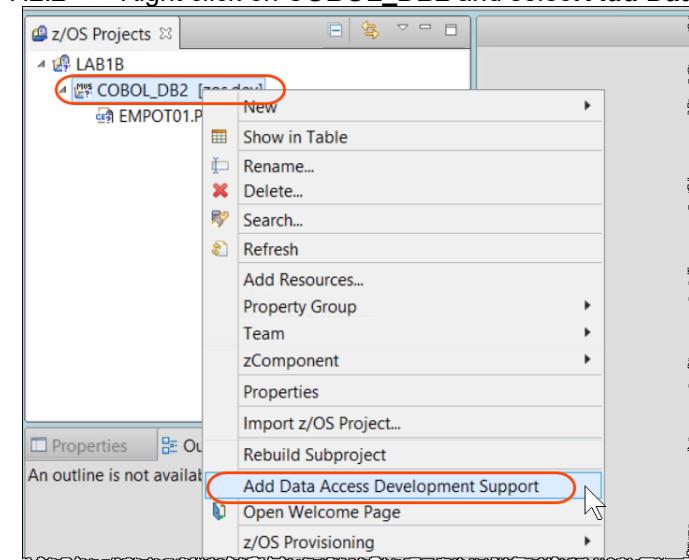
7.2 Running a SQL query from COBOL program

You will need to have authorization to run the queries.

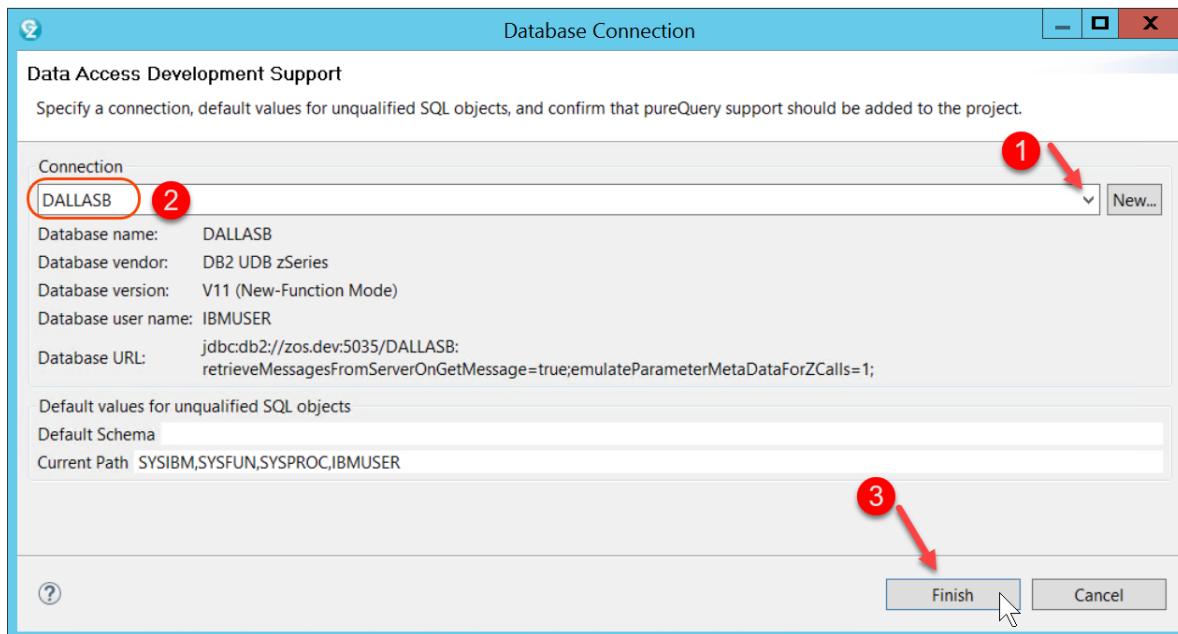
7.2.1 ► Go to the z/OS Projects perspective



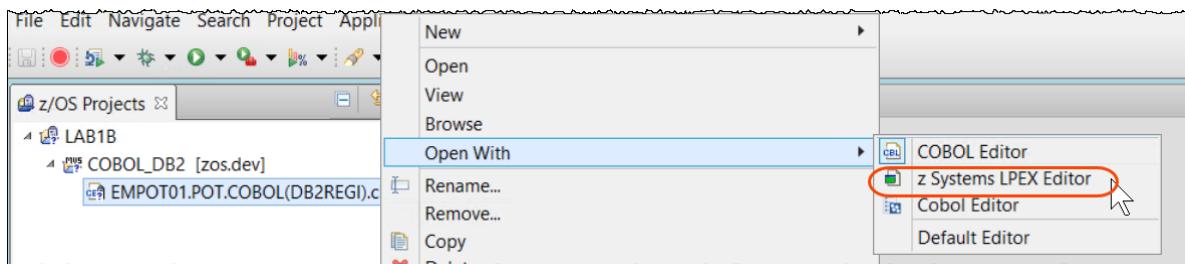
7.2.2 ► Right click on COBOL_DB2 and select Add Data Access Development Support



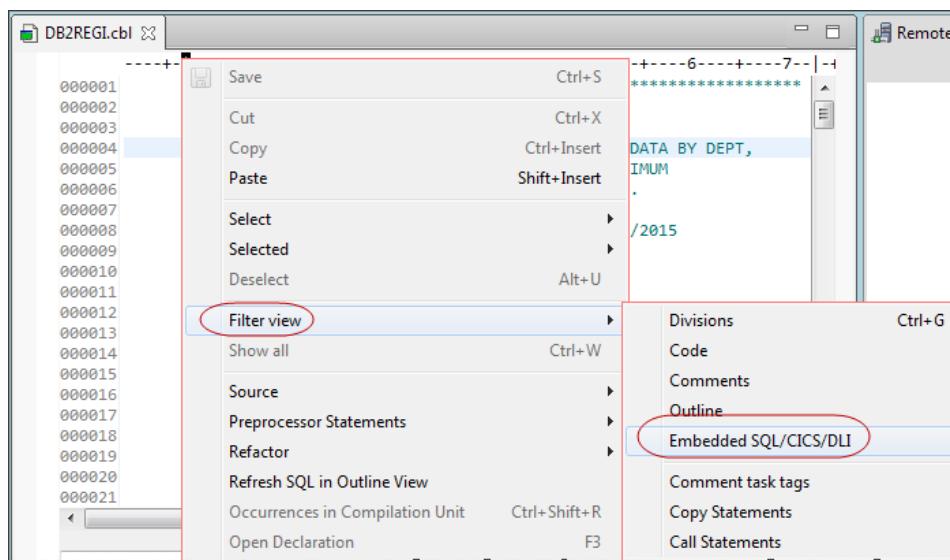
7.2.3 ► Select the connection created **DALLASB** and click **Finish**



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**



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

The screenshot shows the DB2REGL.cbl editor window. A context menu is open over a block of SQL code. The menu items include Paste, Shift+Insert, Select, Selected, Deselect, Filter view, Show all, Source, Preprocessor Statements, Refactor, Tune SQL, Run SQL (which is circled in red), Refresh SQL in Outline View, Occurrences in Compilation Unit, Open Declaration, Open Perform Hierarchy, and View.

```

DB2REGL.cbl
-----+---A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+---8
+ 000030      EXEC SQL INCLUDE SQLCA END-EXEC.
+ 000068      EXEC SQL INCLUDE DIAGCODE END-EXEC.
+ 000069      EXEC SQL INCLUDE EMPL END-EXEC.
+ 000070      EXEC SQL INCLUDE CUST1 END-EXEC.
+ 000121      EXEC SQL
+ 000122      DECLARE C1 CURSOR FOR
+ 000123      SELECT DEPT, MIN(PERF), MAX(PERF),
+ 000124          MIN(HOURS), MAX(HOURS), AVG(HOURS)
+ 000125      FROM RBAROSA.EMPL E, RBAROSA.PAY P
+ 000126      WHERE E.NBR = P.NBR
+ 000127      * AND PERF > :PERF
+ 000128      GROUP BY DEPT
+ 000129      END-EXEC.
+ 000130      EXEC SQL OPEN C1
+ 000131      END-EXEC.
+ 000187      EXEC SQL FETCH C1 INTO
+ 000188          :DEPT-TBL:DEPT-NULL,

```

7.2.7 ► Click as below and you will have the query results

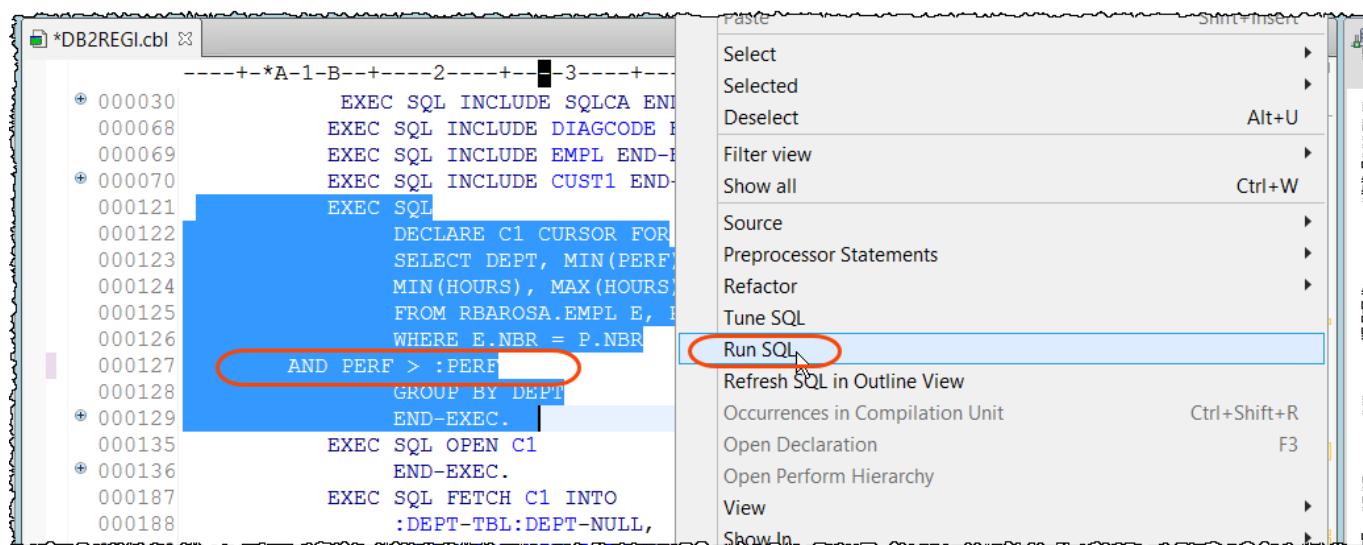
The screenshot shows the DB2REGL.cbl editor window with the results of the executed query displayed in the bottom pane. The results are presented in a table with columns labeled DEPT, 2, 3, 4, 5, 6, and 7. The table contains six rows of data. A red circle labeled '1' is on the status bar, and a red circle labeled '2' is on the results table header.

	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...

1 Type query expression here
2 Status Result1

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

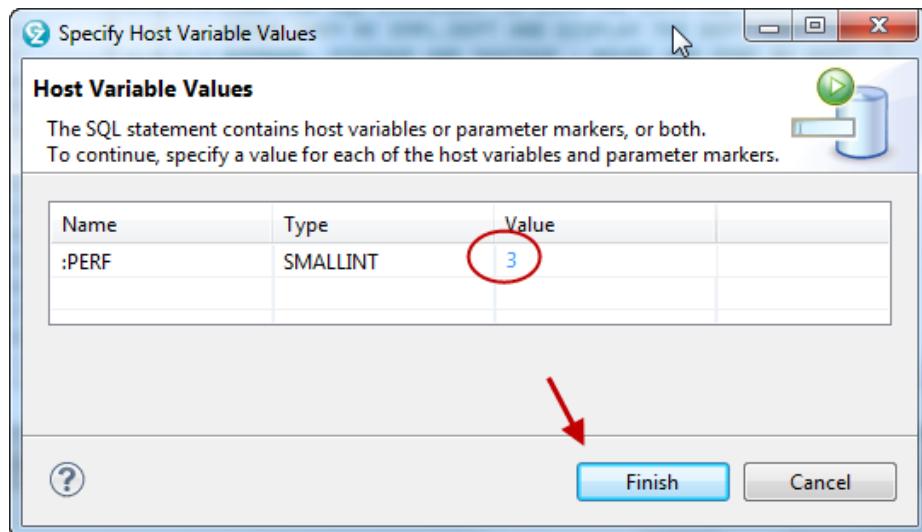


```

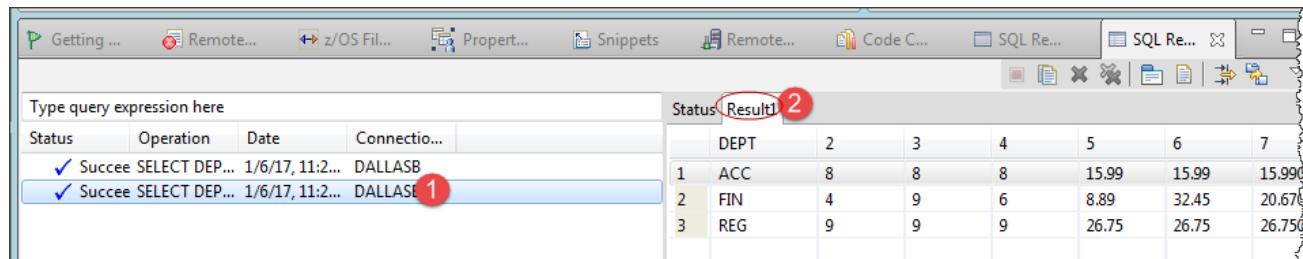
*DB2REGL.cbl
-----+-----+-----+-----+
+ 000030      EXEC SQL INCLUDE SQLCA END-EXEC.
+ 000068      EXEC SQL INCLUDE DIAGCODE END-EXEC.
+ 000069      EXEC SQL INCLUDE EMPL END-EXEC.
+ 000070      EXEC SQL INCLUDE CUST1 END-EXEC.
000121      EXEC SQL
000122      DECLARE C1 CURSOR FOR
000123      SELECT DEPT, MIN(PERF)
000124      MIN(HOURS), MAX(HOURS)
000125      FROM RBAROSA.EMPL E,
000126      WHERE E.NBR = P.NBR
000127      AND PERF > :PERF
000128      GROUP BY DEPT
000129      END-EXEC.
+ 000135      EXEC SQL OPEN C1
+ 000136      END-EXEC.
000187      EXEC SQL FETCH C1 INTO
:DEPT-TBL:DEPT-NULL,
000188

```

7.2.9 ► Specify a value like 3 and click Finish



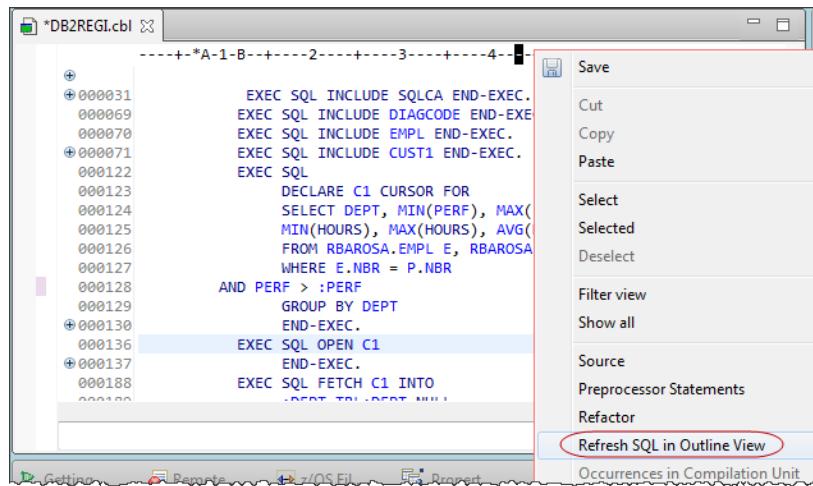
7.2.10 The new results now will be:



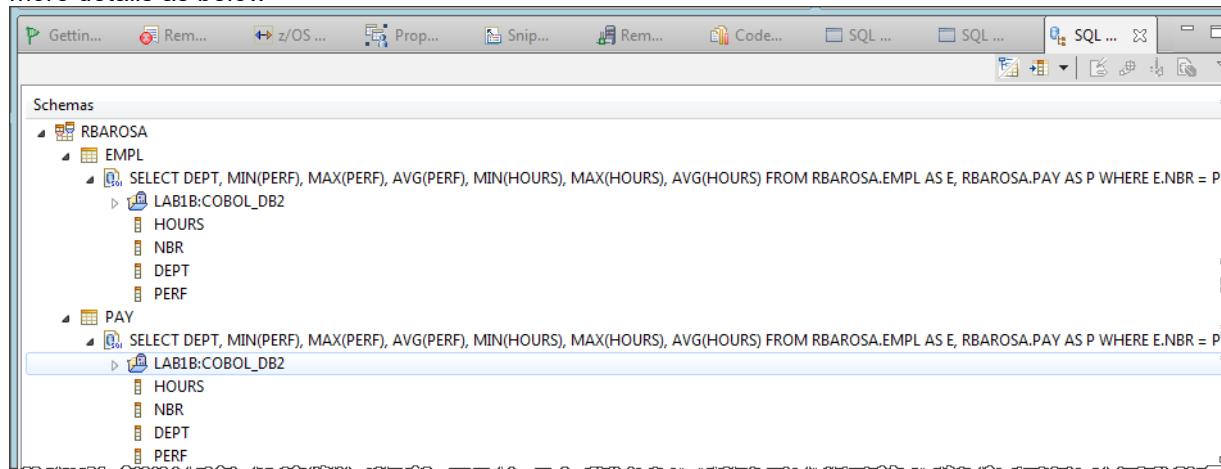
DEPT	2	3	4	5	6	7
1 ACC	8	8	8	15.99	15.99	15.99
2 FIN	4	9	6	8.89	32.45	20.67
3 REG	9	9	9	26.75	26.75	26.75

7.3 Using the SQL Outline view

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

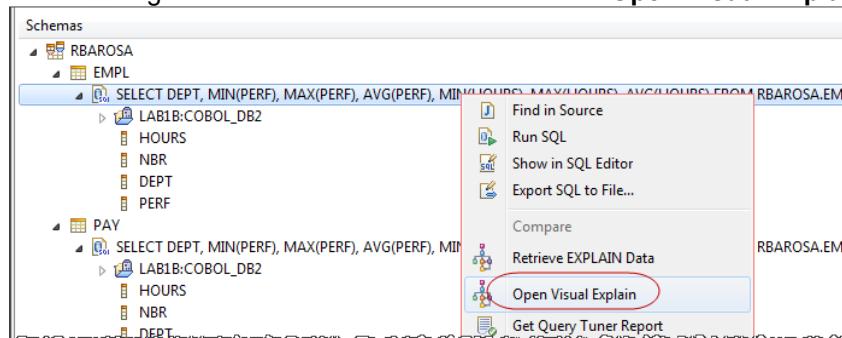


7.3.2 Using SQL Outline View, expand RBAROSA, EMPL, PAY and the SQL statements and you will have more details as below

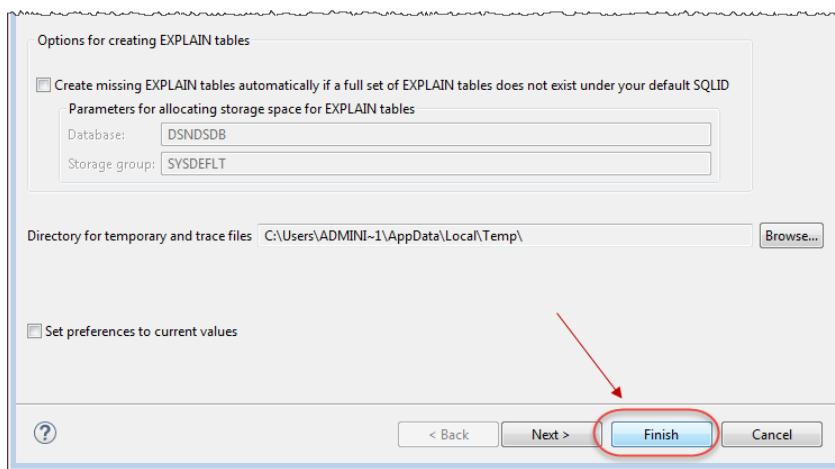


7.4 Using SQL Visual Explain

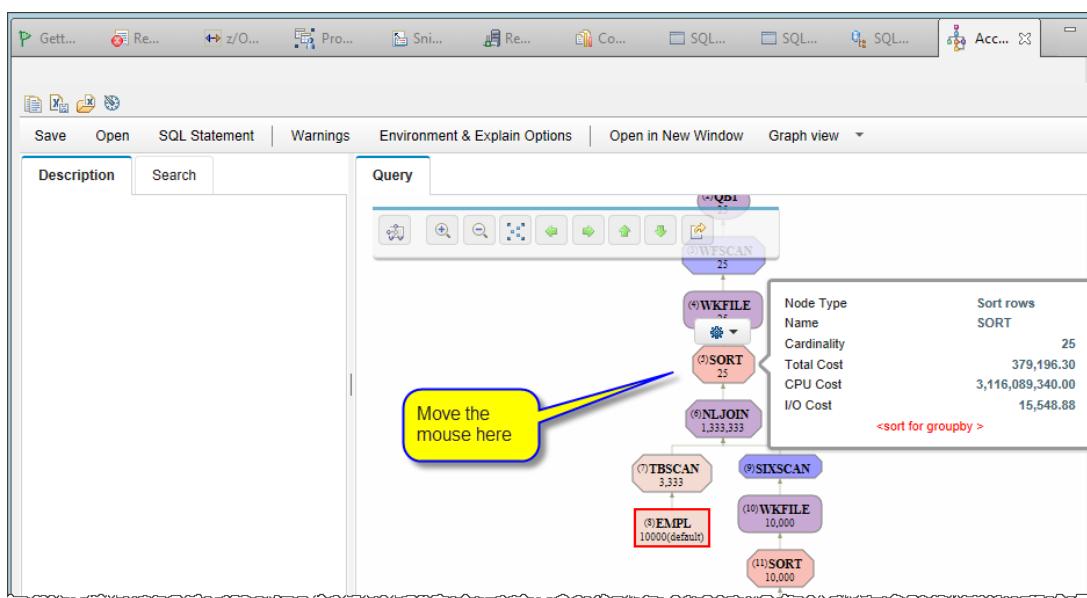
7.4.1 Right click on the first Select and choose Open Visual Explain



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

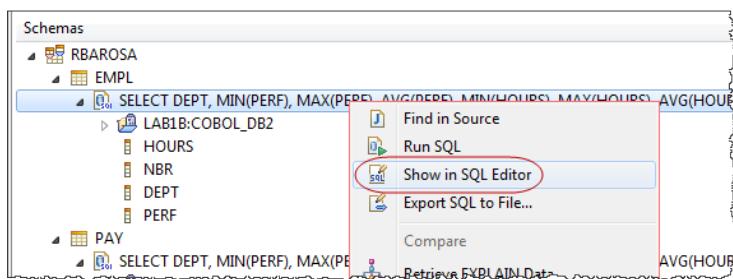


7.4.3 ► Double click on the tab to have a bigger view.
You will have the report below to work with the SQL explain

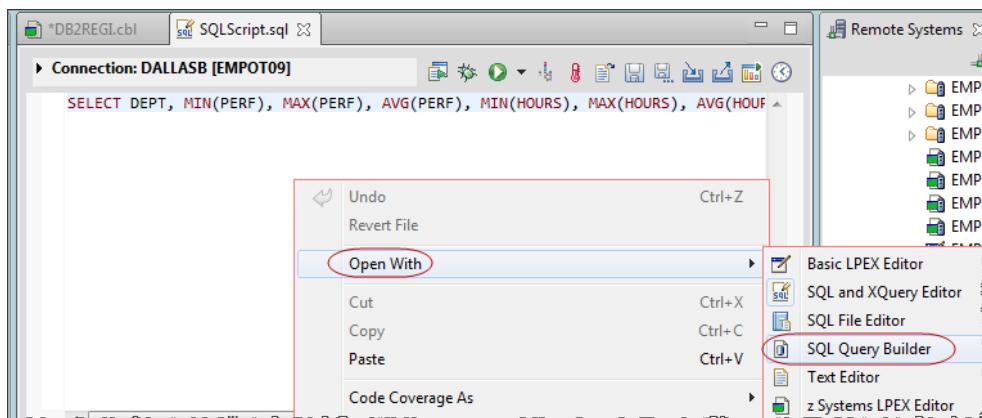


7.5 Reverse engineer the Query

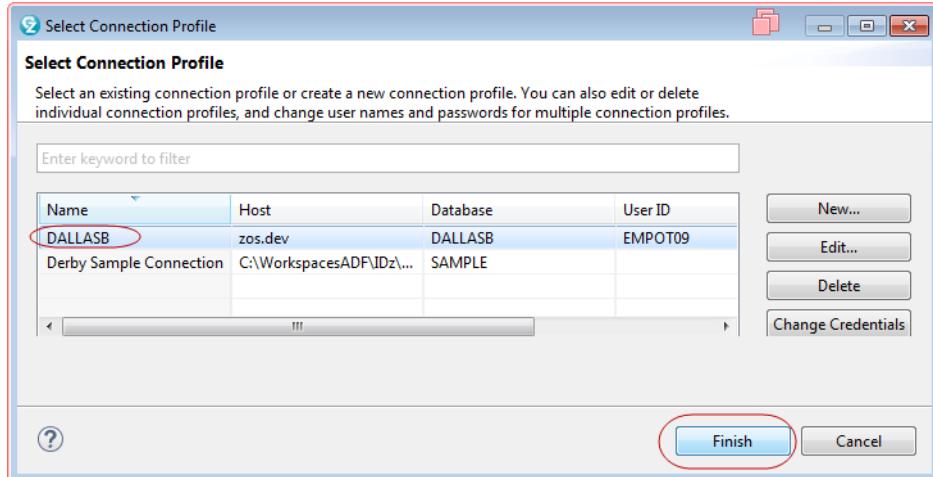
7.5.1 ► Using the SQL Outline view, right click on the Select and choose **Show in SQL Editor**



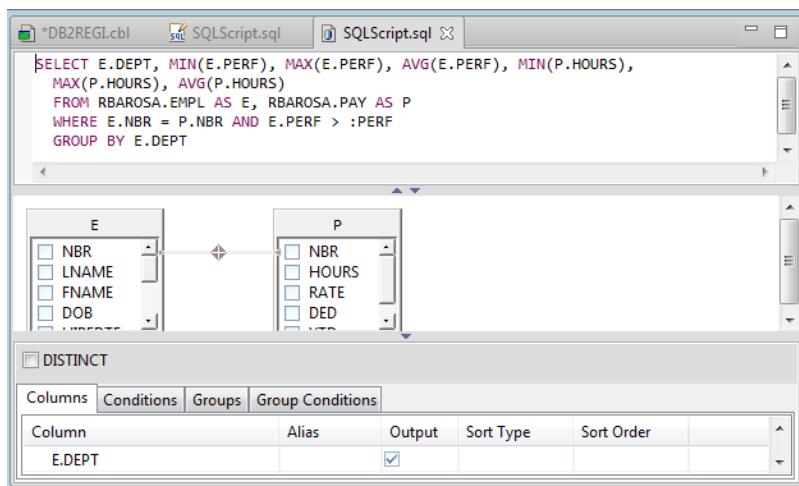
7.5.2 ► When opened,.. select Open With > SQL Query Builder



7.5.3 ► Choose DALLASB and click Finish



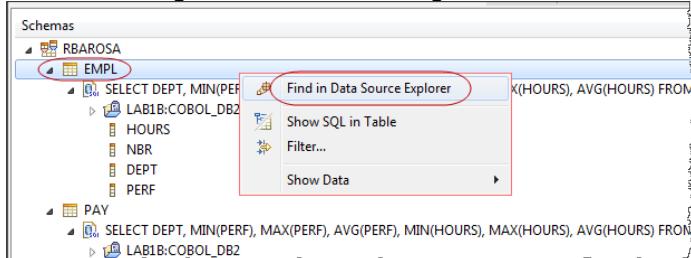
7.5.4 Now we have the graphical editor



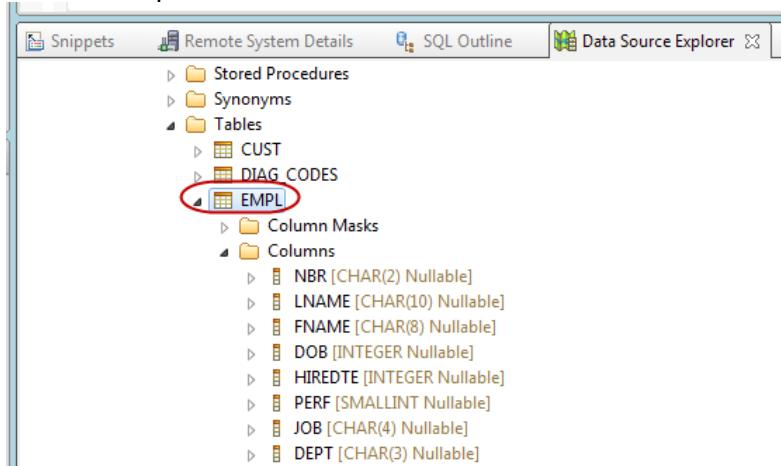
7.5.5 ► You might use this graphical editor to make changes on SQL statements. Notice that the changes reflects on the COBOL code..

7.6 Displaying DB2 table content

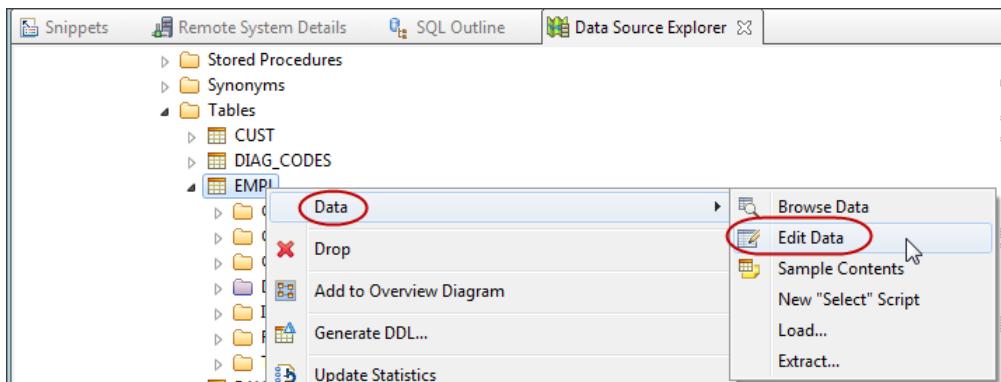
7.6.1 ► Using SQL Outline view, right click on EMPL and select Find Data Source Explorer



7.6.2 ► Expand Tables and EMPL

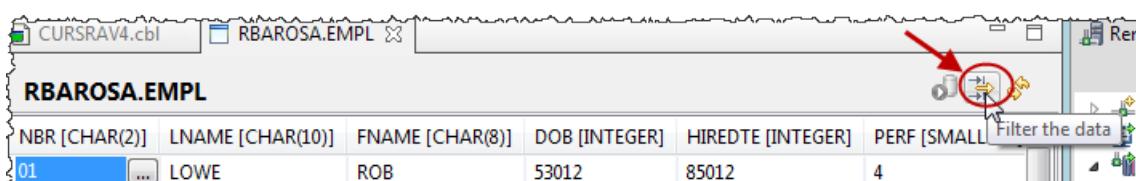


7.6.3 ► Right click on EMPL and select Data > Edit Data

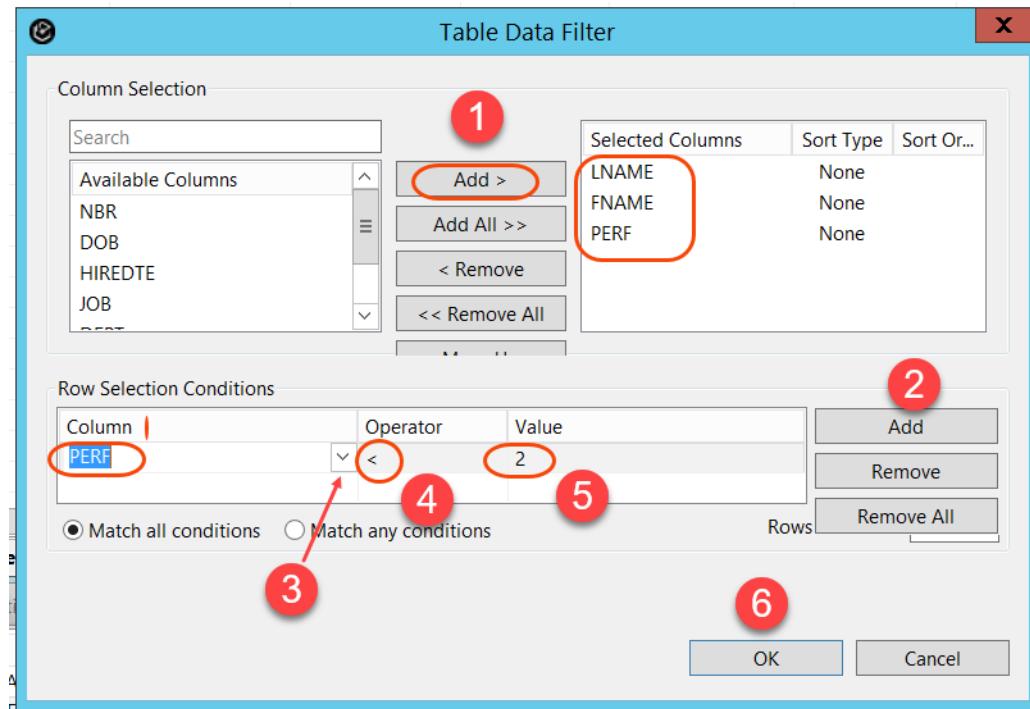


Filtering data results.

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:

RBAROSA.EMPL [Filtered]			
	LNAME [CHAR(10)]	FNAME [CHAR(8)]	PERF [SMALLINT]
1	MOORE	ROGER	1
2	LANCASTER	BURT	1
3	BLAIR	LINDA	1
4	MOORE	ROGER	1
5	MOSTEL	ZERO	0
6	LANCASTER	BURT	1
7	BLAIR	LINDA	1
8			0
<new row>			

Connection : DALLASB Showing all 8 rows [Change SQL results view options](#)

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.

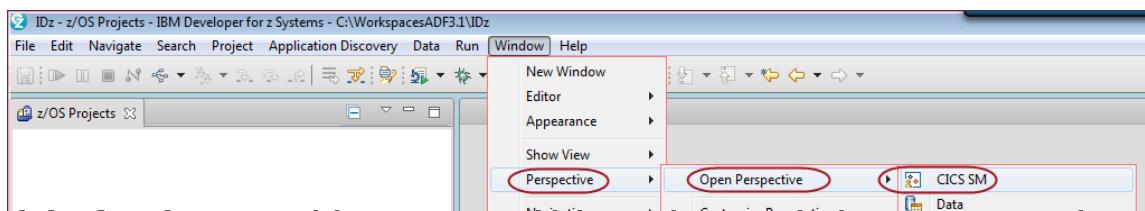
Note that this exercise could be also done with either IDz version 14 or version 15. Here we are using version 14, but instructions will be the same.

8.1 Verify that you are connected to the PDTTOOLS Common components

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

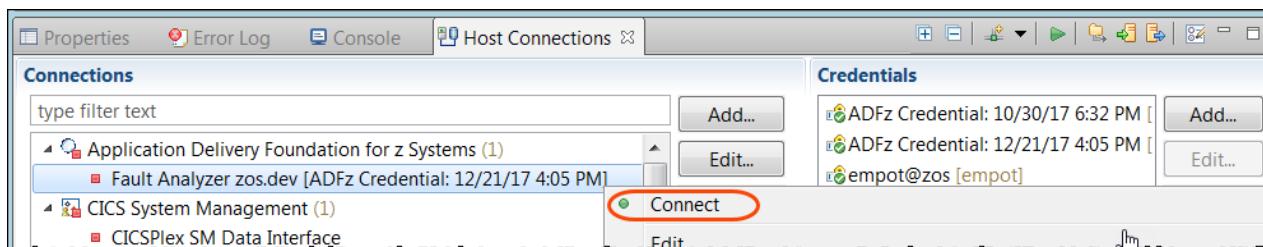
8.1.1 Go to **CICS SM** perspective using

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

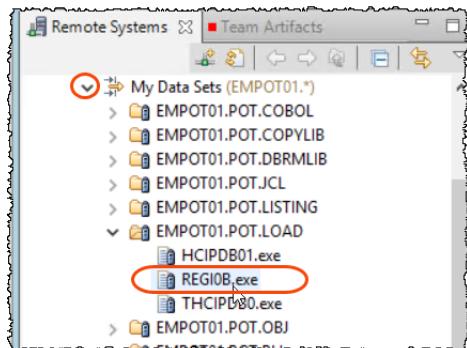
You can use the **View Load Module** utility function to print a list of the symbols (CSECTs, common sections, entry points, and ZAPs) in a load module. You also could compare load modules, using the Compare wizard.

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](#).

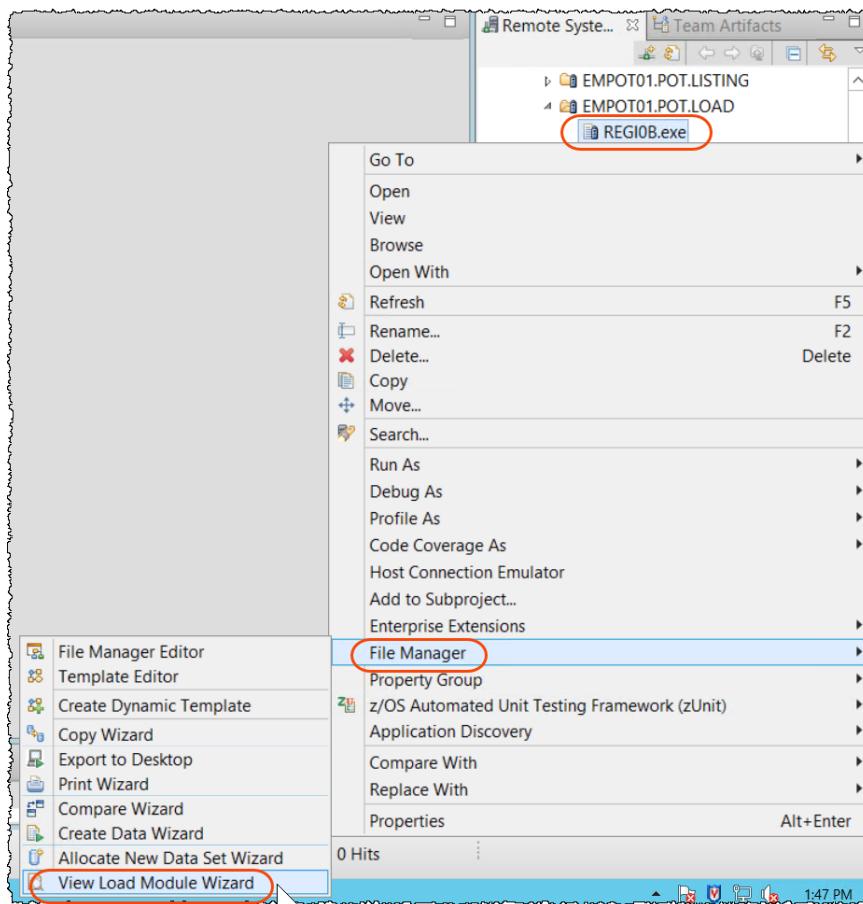
8.2.1 ► Go to **z/OS Projects** perspective clicking on the upper top right icon as below



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..

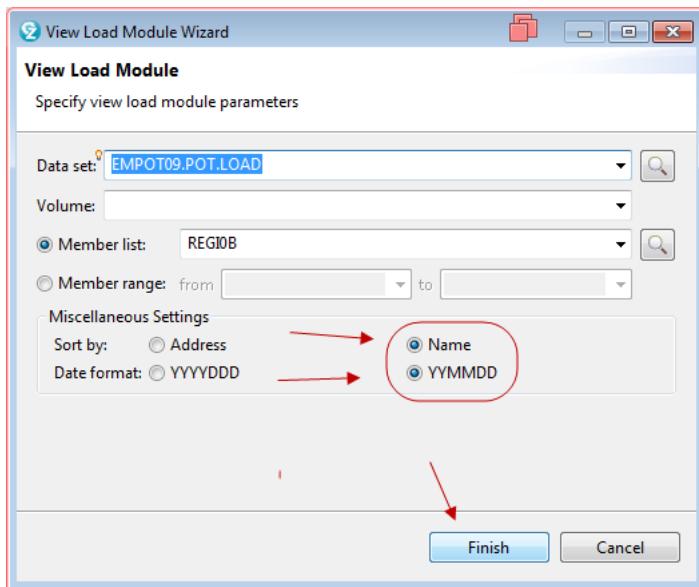


8.2.3 ► Right click on **REGI0B.exe** and select **File Manager** and **View Load Module Wizard**.



► 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.

```

tmp1559069406042.txt
-----+-----+-----+-----+-----+-----+-----+
000018 0
000019 Load Module Information
000020
000021 Load Library EMPOT01.POT.LOAD
000022 Load Module REGI0B
000023 Linked on 19/05/28 at 10:33:32 by PROGRAM BINDER 5695-PMB V2R2
000024 EPA 000000 Size 000142C TTR 000005 SSI AC 00 AM
000025
000026 Name Type Address Size Class A/RMODE Compiler 1
000027 -----
000028 -PRIVATE PC 0000000 0000008 C_@PPA2 ANY/ANY
000029 -PRIVATE PC 0000000 0000004 C_@CSINIT ANY/ANY
000030 -PRIVATE PC 0000000 0000084 C_WSA ANY/ANY
000031 CEEARLU SD 0000718 00000B8 B_TEXT MIN/ANY PL/X 390 V2R4
000032 CEEBETBL SD 00004C0 0000028 B_TEXT MIN/ANY HIGH-LEVEL AS
000033 CEEBINT SD 0000710 0000008 B_TEXT MIN/ANY PL/X 390 V2R4
000034 CEEBLIST SD 00006B0 000005C B_TEXT MIN/ANY HIGH-LEVEL AS
000035 CEELLIST LD 00006C0 000004C B_TEXT 31/ANY PL/X 390 V2R4
000036 CEEBPIRA SD 00007D0 00002A0 B_TEXT
-----+-----+-----+-----+-----+-----+-----+

```

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

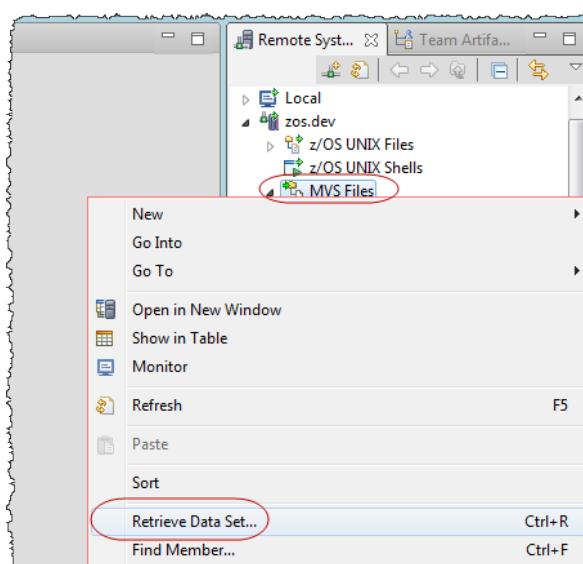
In this section, you will open a variable blocked sequential dataset in the editor or viewer, using a copybook as a layout.

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.

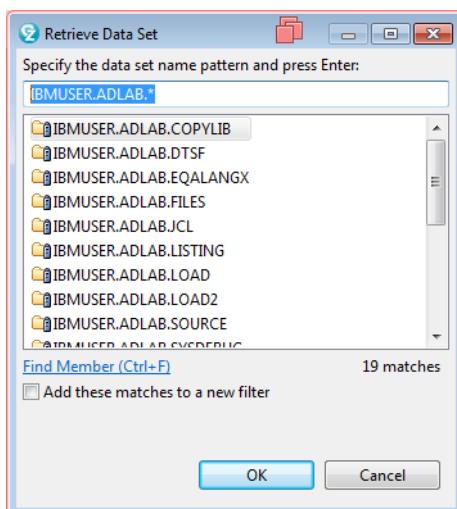
8.3.1 Before you start mapping the dataset using a COBOL COPYBOOK, let us copy the copybook that maps the file to your PDS.

► Using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set**

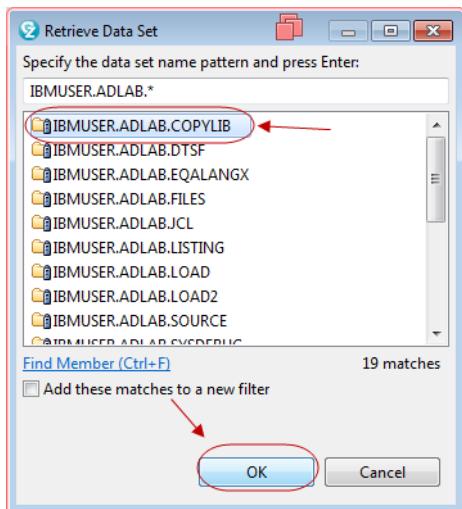
This is equivalent at the TSO 3.4 function.



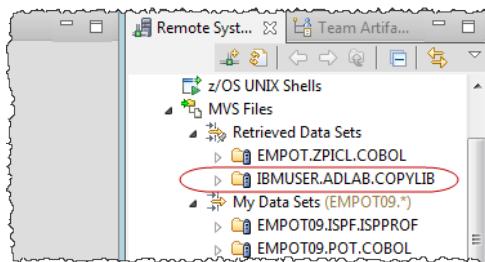
8.3.2 ► Type **IBMUSER.ADLAB.*** and press **Enter**.



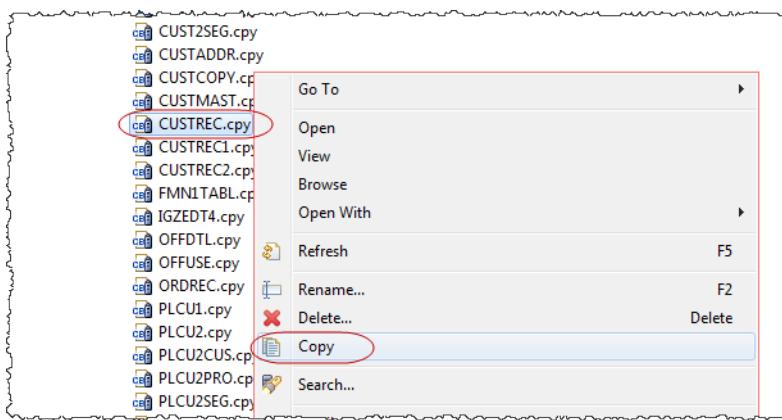
8.3.3 ► Select **IBMUSER.ADLAB.COPYLIB** and click **OK**.



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

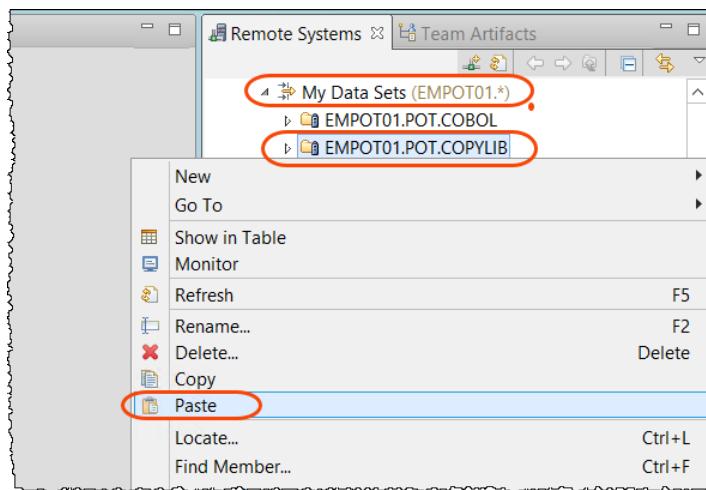


8.3.5 ► Expand **IBMUSER.ADLAB.COPYLIB**, right click on **CUSTREC.cpy** and select **Copy**. You want to copy this member to your PDS.

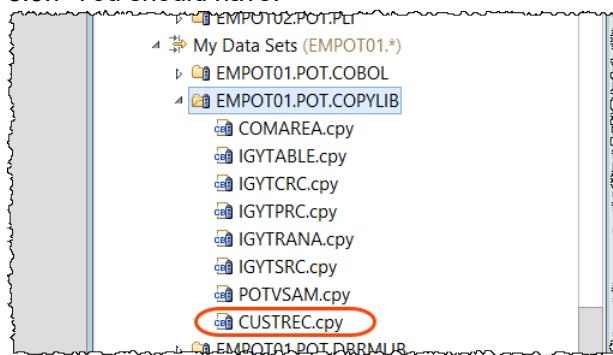


8.3.6 ➡ Navigate to **My Data Sets**, right click on **EMPOT01.POT.COPYLIB** and select **Paste**

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.

A screenshot of the COBOL editor showing the code for 'CUSTREC.cpy'. The code defines a record structure with various fields like CUST-ID, NAME, ACCT-BALANCE, etc. On the right side of the screen, the 'Remote Systems' interface shows the 'My Data Sets (EMPOT01.*)' tree, with 'CUSTREC.cpy' highlighted by a red oval. A large red arrow points from this highlighted entry in the tree to the corresponding code in the editor window.

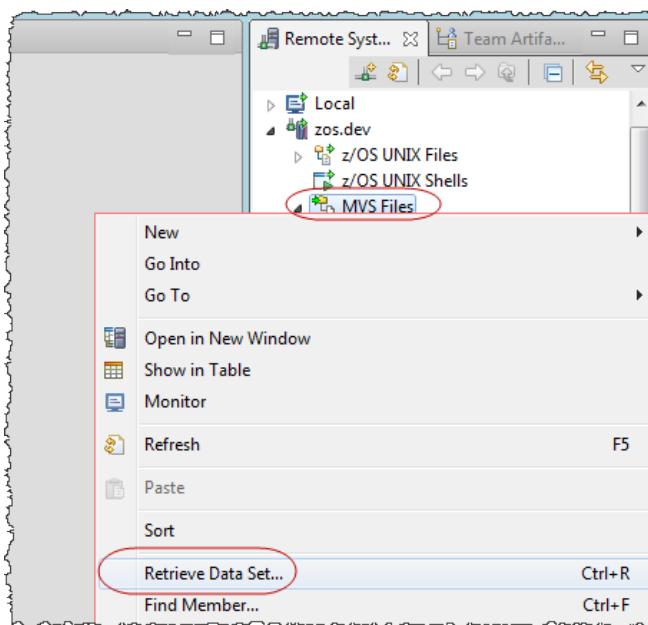
```

*** ++++++1-----2-----3-----4-----5-----6-----7-----8
1  *+
2  * Sample COBOL Copybook for IBM PD Tools Workshops
3  *
4  * The sample data described by this copybook
5  *      is <USERID>.ADLAB.CUSTFILE
6  *** ++++++1-----2-----3-----4-----5-----6-----7-----8
7 01 CUST-REC.
8    05 CUSTOMER-KEY.
9      10 CUST-ID          PIC X(5).
10     10 REC-TYPE         PIC X.
11    05 NAME             PIC X(17).
12    05 ACCT-BALANCE     PIC S9(7)V99  COMP-3.
13    05 ORDERS-YTD       PIC S9(5)   COMP.
14    05 ADDR              PIC X(20).
15    05 CITY              PIC X(14).
16    05 STATE              PIC X(02).
17    05 COUNTRY            PIC X(11).
18    05 MONTH              PIC S9(7)V99  COMP-3  OCCURS 12.
19    05 OCCUPATION         PIC X(30).
20    05 NOTES              PIC X(120).
21    05 LAB-DATA-1         PIC X(05).

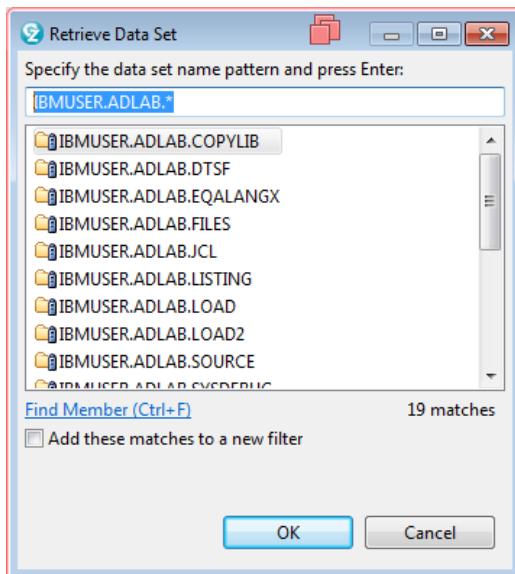
```

8.3.9 ► To see the data set, using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set**

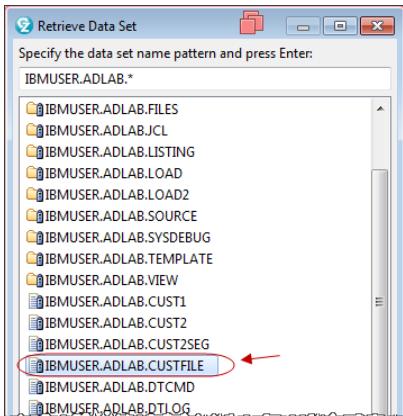
This is equivalent at the TSO 3.4 function.



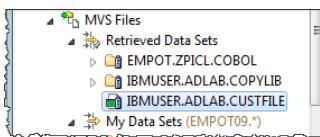
8.3.10 ► Type **IBMUSER.ADLAB.*** and press **Enter**.



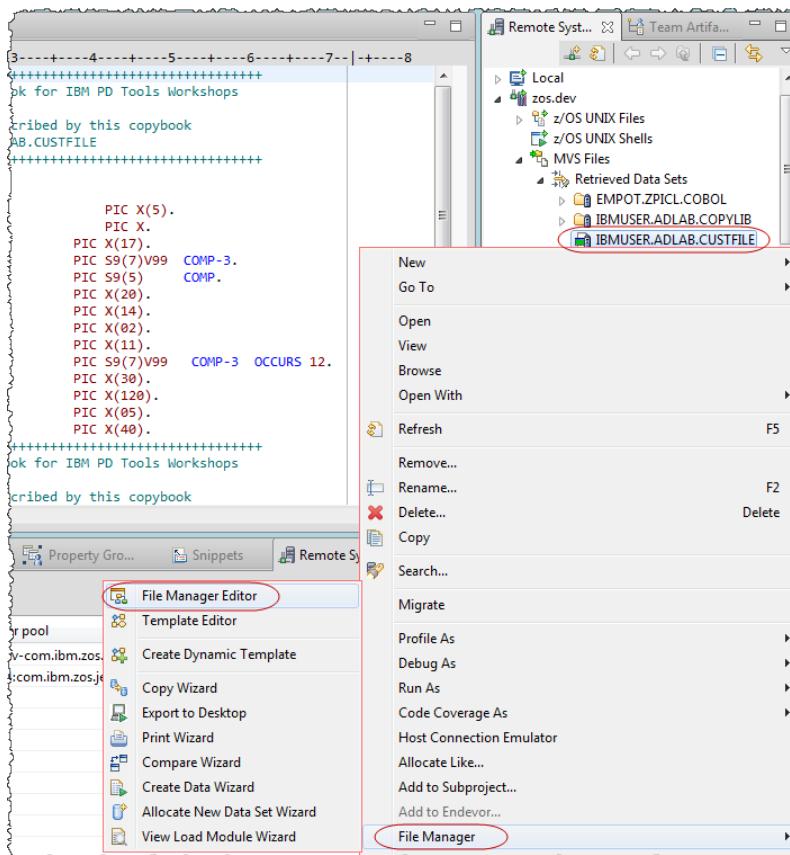
8.3.11  Scroll down, select **IBMUSER.ADLAB.CUSTFILE** and click **OK**.



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

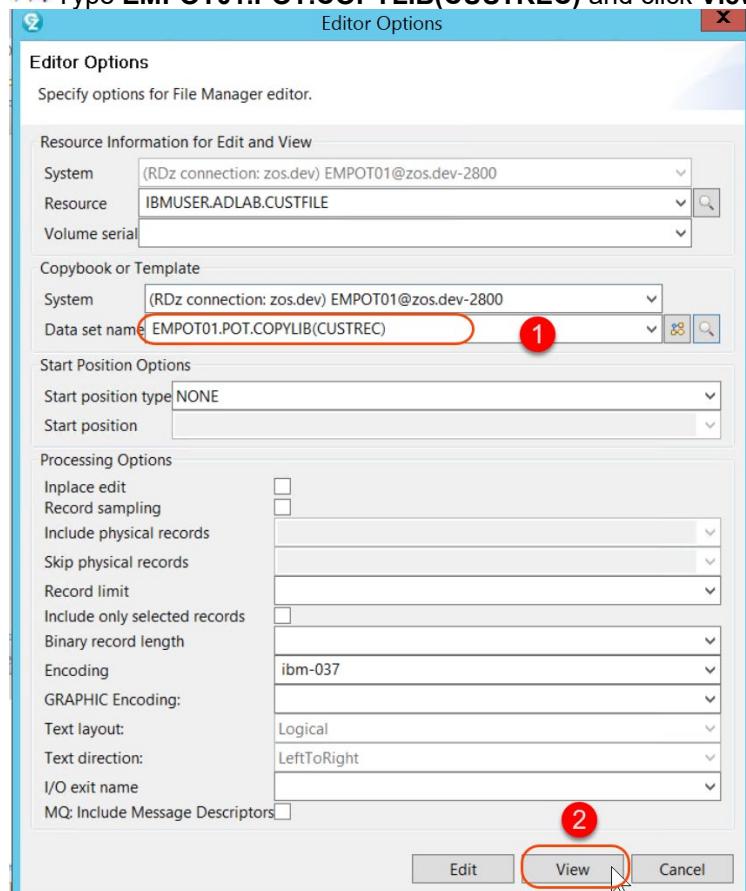


8.3.13  Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**. You want to map this file against the COBOL COPYBOOK that you copied.



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**.



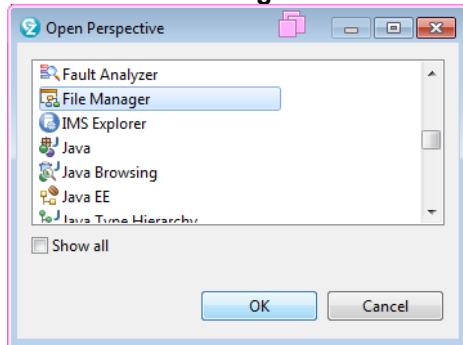
8.3.15 The file record is displayed (type A) in formatted mode.

	CUST-ID	REC-TYPE	NAME	ACCT-BALANCE	ORDERS-YTD	ADD
1	03115	A	Graham, Holly	254.53	1	310
2	CONT...					
3	CONT...					
4	05580	A	Moore, Adeline	498.95	3	476
5	CONT...					
6	CONT...					
7	CONT...					
8	06075	A	Dubree, Dustin	192.98	1	922
9	06927	A	Buchs, Jillian	99.99	0	41
10	07008	A	Houston, Roger	296.97	1	411
11	07025	A	Marx, Audrey	450.51	2	90
12	CONT...					
13	CONT...					
14	CONT...					
15	11204	A	Ness, Luke	513.06	3	516
16	CONT...					
17	CONT...					

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.

	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...									

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	FD	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

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)..

The screenshot shows the CBL CUSTREC.cpy editor interface. At the top, there is a toolbar with various icons. Below the toolbar is a navigation bar with buttons for 'TOP', 'Column# 1', 'Cursor', and 'Current 2 (Suppressed record)'. The main area contains a table with 14 rows of customer data. Row 2 is highlighted with a red box and labeled '2 CONTACT-R'. A red arrow points from this row to the single-record view below. The single-record view is titled 'Layout CONTACT-REC' and shows a detailed table of field definitions:

Field	Picture	Type	Start	Length	Data
CUST-ID	X(5)	AN	1	5	03115
REC-TYPE	X	AN	6	1	B
NAME	X(17)	AN	7	17	Graham, Holly
DESCRIPTION	X(10)	AN	24	10	Home Phone
CONTACT-INFO	X(20)	AN	34	20	112-555-6736

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.

The screenshot shows the CBL CUSTREC.cpy editor interface. The layout dropdown menu is open, and 'CONTACT-REC' is selected. A red box highlights the 'CONTACT-REC' option in the dropdown menu. A red arrow points from the 'CONTACT-REC' option to the single-record view below. The single-record view is titled 'Layout CONTACT-REC' and shows a detailed table of field definitions, identical to the one in the previous screenshot.

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

	CUST-ID	REC-TYPE	NAME	DESCRIPTION	CONTACT-INFO
1	CUST-...				
2	03115	B	Graham, Holly	Home Phone	112-555-6736
3	03115	C	Graham, Holly	Cell Phone	135-555-2338
4	CUST-...				
5	05580	B	Moore, Adeline	Work Phone	161-555-4024
6	05580	C	Moore, Adeline	Home Phone	221-555-7598
7	05580	D	Moore, Adeline	Cell Phone	138-555-2410
8	CUST-...				
9	CUST-...				
10	CUST...				
11	CUST...				
12	07025	B	Marx, Audrey	Cell Phone	232-555-7244
13	07025	C	Marx, Audrey	Home Phone	240-555-4245
14	07025	D	Marx, Audrey	Work Phone	232-555-8753

8.3.19 ►► Right click on the editor and verify that there are many options.

►► Click on **Switch Mode**

Context menu options:

- Compare With
- Switch Mode (highlighted)
- Page Up
- Page Down
- Page Left
- Page Right
- Copy Records
- Find/Replace
- Locate Column
- Sort Records
- Hex on/off
- Show Options
- Exclude Records
- Reset Excludes
- Save Records
- SaveAs Records
- Validate
- Software Analysis
- Team
- Replace With

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.

The screenshot shows the CBL CUSTREC.cpy editor window. The data is displayed in character mode, where fields like names and addresses are shown as single strings without field separators. The layout is set to 'CHARACTER' mode, which is highlighted with a red oval. The data includes records for various customers with their names, addresses, and contact information.

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

The screenshot shows the CONTACT-REC layout editor. A dropdown menu under 'View Mode' is open, showing options: Single Mode, Dump Mode, DB2 Single Mode, Structure Mode, and Unstructured Mode. 'Dump Mode' is selected and highlighted with a blue background. Below the dropdown, there are buttons for 'Insert Mode' and 'Insert'. The main table displays field definitions for the CONTACT-REC layout.

The screenshot shows the CONTACT-REC layout editor in dump mode. It displays the raw binary data for the layout structure. The table shows offsets from +0 to +30, and the corresponding hex values for each byte in the record structure.

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

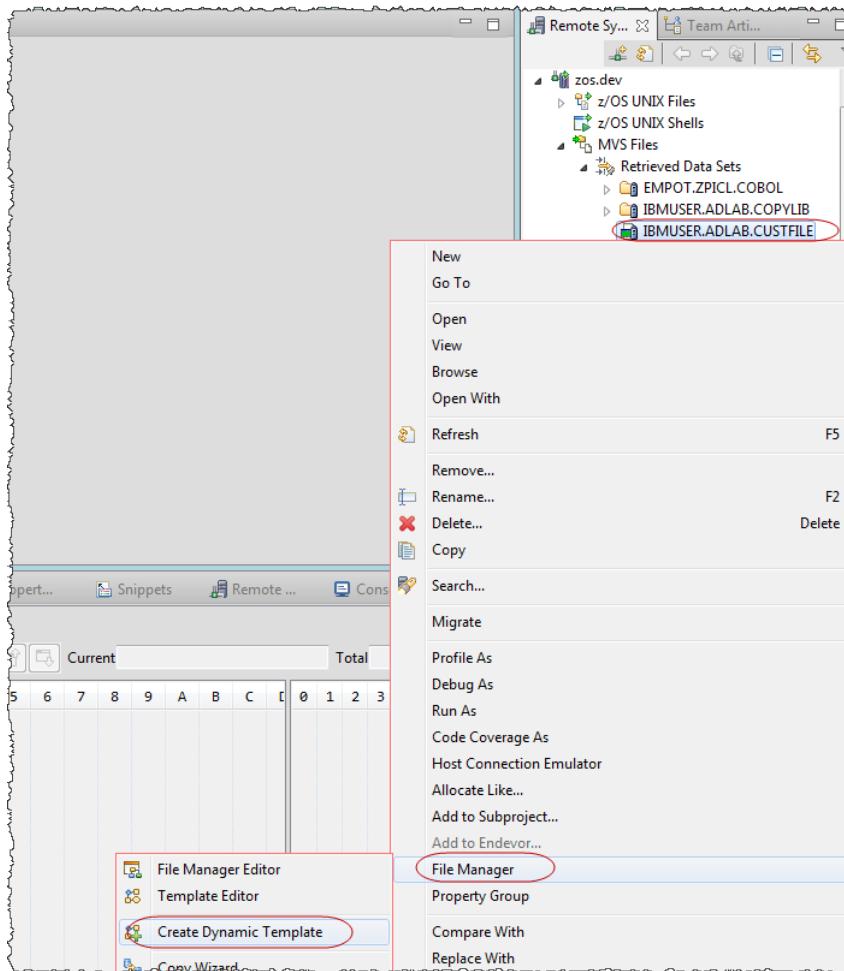
When you use the File Manager editor or viewer, you can specify a layout in the form of either a copybook or a template. If you specify a copybook, the editor/viewer can display records formatted according to the fields in the copybook. While a copybook defines the record layout, it cannot be used to 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.

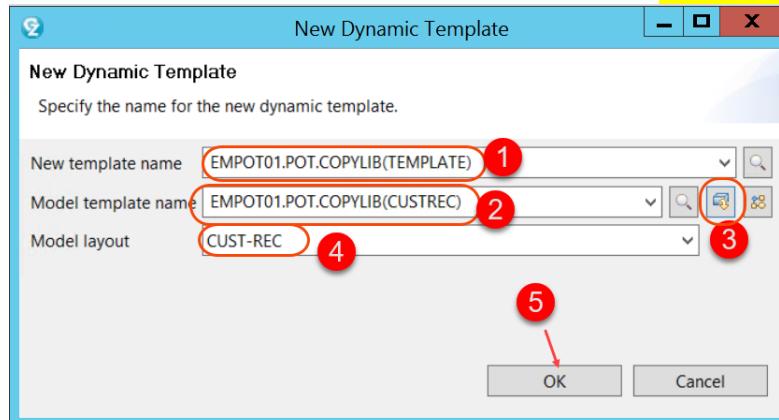
8.4.1  Go back to **z/OS Projects** Perspective and using *Remote Systems Explorer* view, right click **IBMUSER.ADLAB.CUSTFILE** file and select **File Manager -> Create Dynamic Template**



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

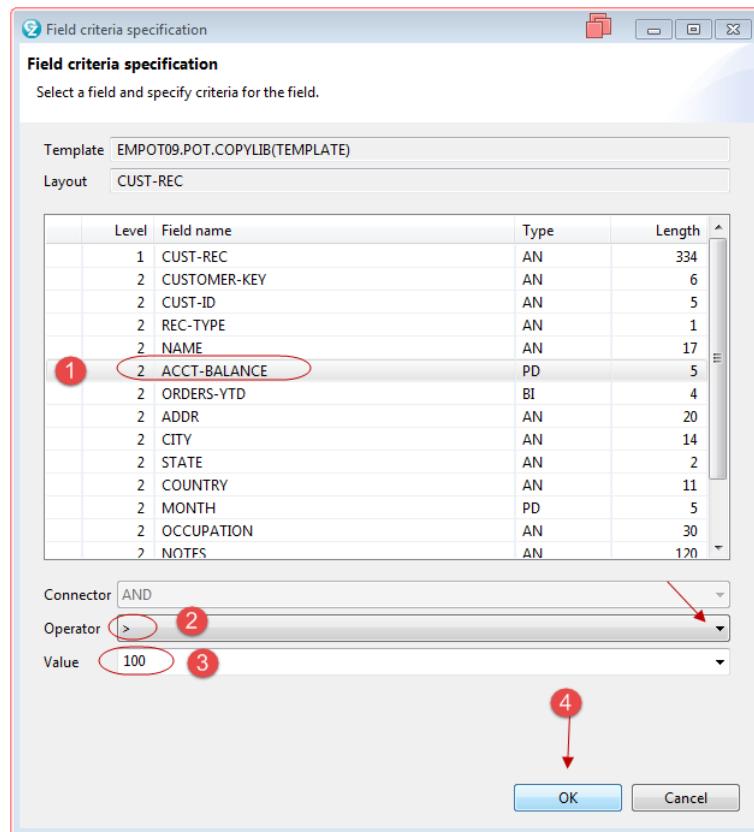
► 2 Using **By-Field Criteria Builder** click on the icon

The screenshot shows the 'File Manager Template Editor' interface. The main window displays a layout for 'CUST-REC'. In the 'Selection criteria' section, there is a grid icon (step 1). Below it, the 'By-Field Criteria Builder' icon (step 2) is highlighted with a red circle. A red arrow points from the 'Selection criteria' icon to the 'By-Field Criteria Builder' icon. The 'By-Field Criteria Builder' dialog is open, showing a table of fields:

Level	Field name	Type	Length
1	CUST-REC	AN	334
2	CUSTOMER-KEY	AN	6
2	CUST-ID	AN	5

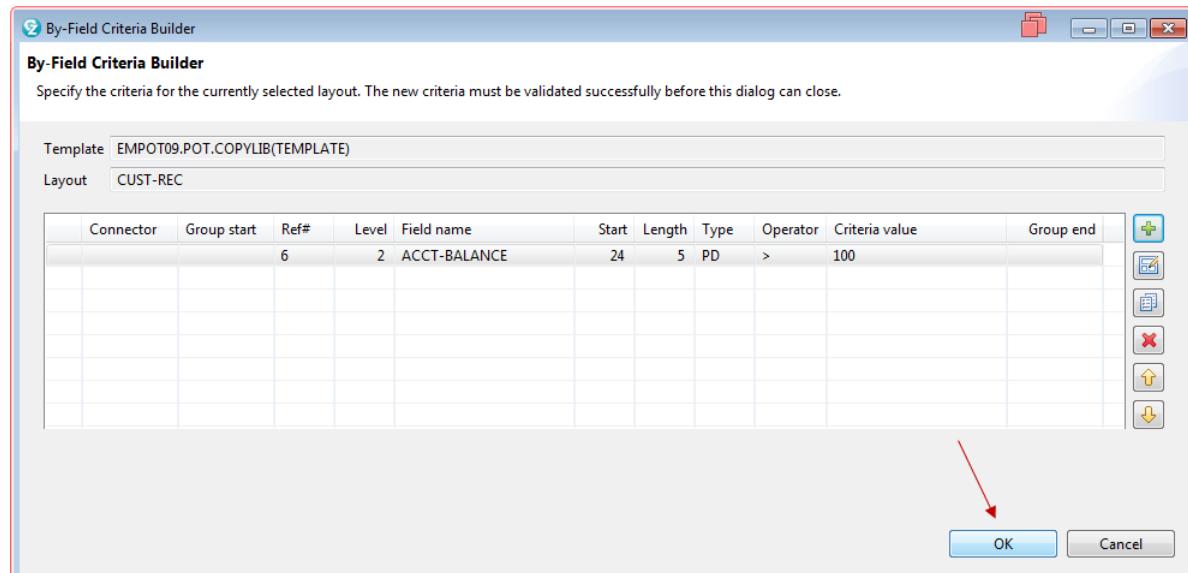
8.4.4 1 On the *Field criteria specification* select **ACCT-BALANCE**

2 On the *Operator* using the drop down select **>** and 3 on the *Value* type **100** and 4 click **OK**.



8.4.5 The result is shown below..

Click **OK**



8.4.6 ► Using the column **Selected** double-click on the boxes

① CUST-ID, ② NAME and ③ ACCT-BALANCE.

► Click **OK**.

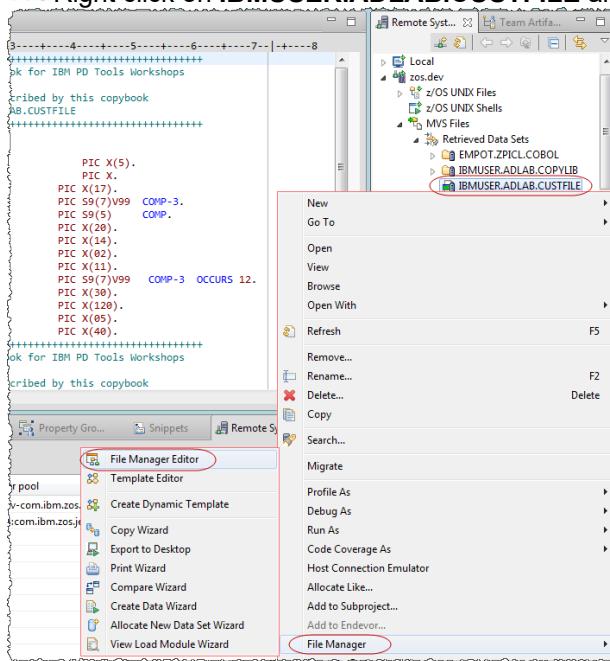
The template is created. At any time, you can edit and modify it.

The screenshot shows the 'File Manager Template Editor' window. The title bar indicates a connection to '(RDz connection: zos.dev) EMPT01@zos.dev-2800:EMPOT01.POT.COPYLIB(YTEMPLAT)'. The main area displays a table of fields with columns for Sequence, Hold, Selected, Level, Name, Additional Clause, Picture, Type, Start, Length, and REDEFINES Crit... . The 'Selected' column contains checkboxes. Red circles numbered 1, 2, and 3 highlight the checkboxes for CUST-ID, NAME, and ACCT-BALANCE respectively, which are all checked. A red circle numbered 4 highlights the 'OK' button at the bottom right of the editor. A red arrow points from the number 4 to the 'OK' button.

	Sequence	Hold	Selected	Level	Name	Additional Clause	Picture	Type	Start	Length	REDEFINES Crit...
1				1	CUST-REC			Alphanumeric	1	334	
2				2	CUSTOMER-KEY			Alphanumeric	1	6	
3			1	2	CUST-ID			Alphanumeric	1	5	
4				2	REC-TYPE			Alphanumeric	6	1	
5			2	2	NAME			Alphanumeric	7	17	
6			3	2	ACCT-BALANCE			Packed decimal	24	5	
7				2	ORDERS-YTD			Binary	29	4	
8				2	ADDR			Alphanumeric	33	20	
9				2	CITY			Alphanumeric	53	14	
10				2	STATE			Alphanumeric	67	2	
11				2	COUNTRY			Alphanumeric	69	11	
12				2	MONTH			Packed decimal	80	5	
13				2	OCCUPATION			Alphanumeric	140	30	
14				2	NOTES			Alphanumeric	170	120	
15				2	LAB-DATA-1			Alphanumeric	290	5	
16				2	LAB-DATA-2			Alphanumeric	295	40	

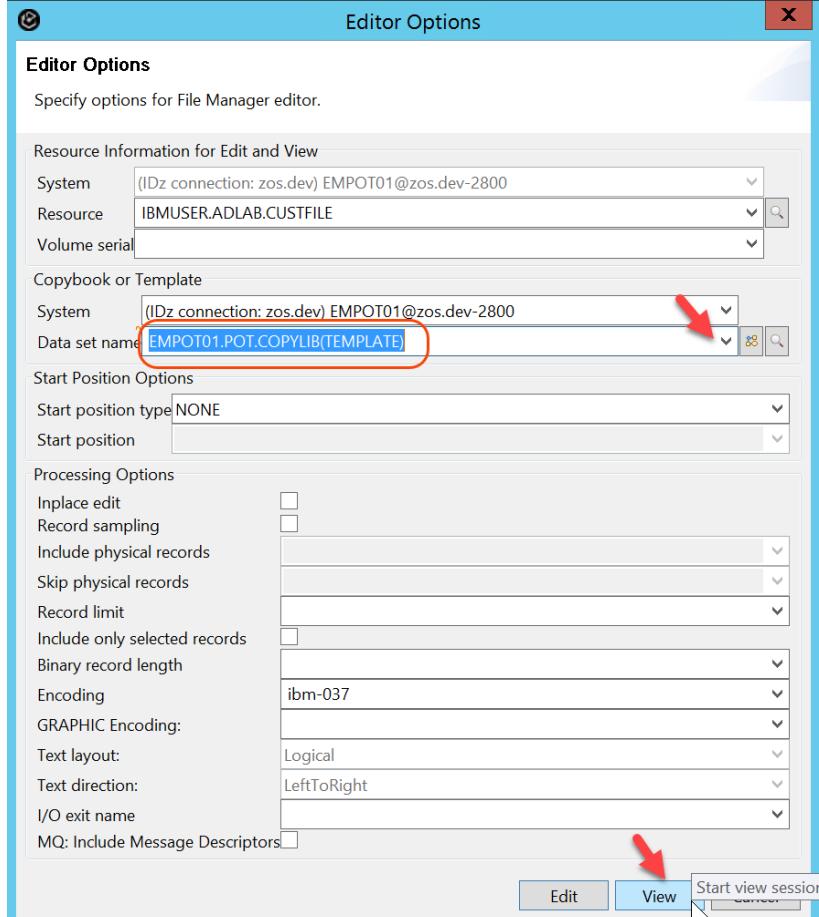
8.4.7 Now that the template is created you can use against the data set.

► 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.

The screenshot shows the Rational Developer for z/OS (RDz) interface. On the left, there is a data editor titled 'CUST-R' showing a list of customers with columns for CUST-ID, NAME, and ACCT-BALANCE. The data includes rows for Graham, Holly, Moore, Adeline, and others. Below this is another data editor titled 'CUST-REC' showing field definitions for CUST-ID, NAME, and ACCT-BALANCE. On the right, a 'Remote System Det...' browser window is open, showing a tree view of datasets on 'zos.dev' such as z/OS UNIX Files, z/OS UNIX Shells, and MVS Files, including 'IBMUSER.ADLAB.CUSTFILE' and 'IBMUSER.ADLAB.*'.

	CUST-ID	NAME	ACCT-BALANCE
1	03115	Graham, Holly	25453
2 CONTA...			
3 CONTA...			
4	05580	Moore, Adeline	49895
5 CONTA...			
6 CONTA...			
7 CONTA...			
8	06075	Dubree, Dustin	19298
9	06927	Buchs, Jillian	9999
10	07008	Houston, Roger	29697
11	07025	Marx, Audrey	45051
12 CONT...			
13 CONT...			
14 CONT...			

Field	Picture	Type	Start	Length	Data
CUST-ID		AN	1	5	03115
NAME		AN	7	17	Graham, Holly
ACCT-BALANCE		PD	24	5	25453

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 2D -(OPTIONAL) z/OS Connect EE Toolkit : Create an API from Catalog Manager Application (EGUI)

Created by Wilbert, updated April 12 2021 by Regi



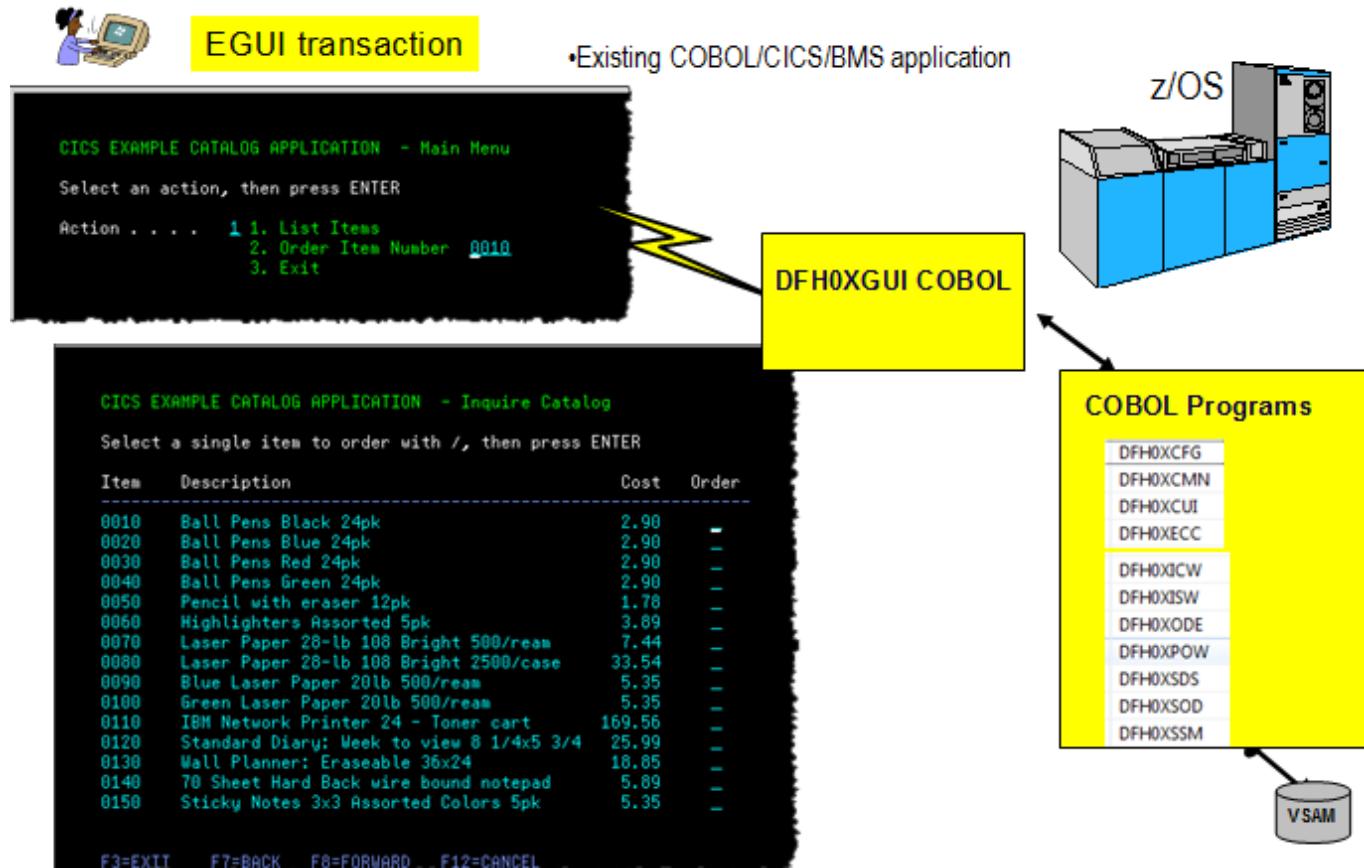
Acknowledgments:

We would like to thank the original authors of this exercises:
Aymeric Affouard, Eric Phan, Paul Pilotto & Nigel Williams.

Overview of development tasks

In this lab you will go through the process of creating an API that allows REST clients to access the application.
The different steps of the lab are:

1. Create your team services using the z/OS Connect EE API Toolkit
2. Create your team API using the z/OS Connect EE API Editor
3. Test your team API using a REST client:





Each time you see a symbol ➡ it means that you have to “do” something on your computer – not merely read the document.

z/OS Connect EE

IBM z/OS Connect EE provides a framework that enables z/OS based programs and data to participate fully in the new API economy for mobile and cloud applications. It provides REST API access to z/OS subsystems, such as CICS, IMS, WebSphere MQ, DB2, and Batch.

The z/OS Connect EE API toolkit is an Eclipse-based workstation tool that you install into IBM Explorer for z/OS to create services and REST APIs for accessing z/OS resources. In the API toolkit, you can create two types of projects: service projects and API projects

Section 1 – Create and configure a z/OS Connect Service Project

You will now create a z/OS Connect EE Service project and use the COBOL copybook to create a service. This copybook was identified as a candidate to invoke an API from the existing EGUI transaction.

To create the z/OS Connect EE Service you need to use the “z/OS Connect Enterprise Edition” perspective

1.1 Create the z/OS Connect service project

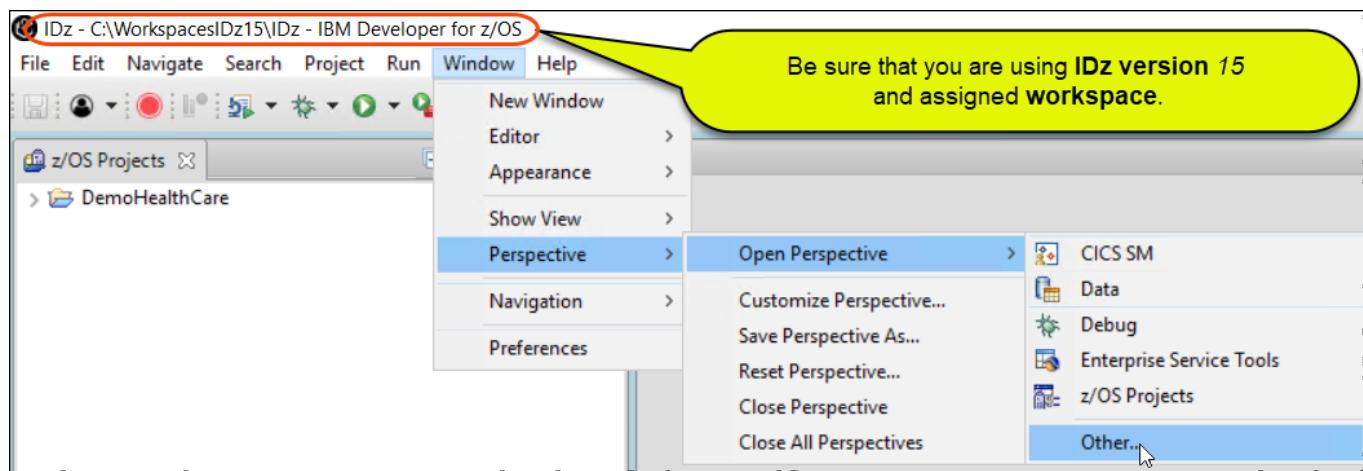
1.1.1 Start IBM Developer for z Systems version 15 if not already started

➡ Using the desktop double click on **IDz V15** icon. Verify that the message indicates that it is Version **15.0.1**.

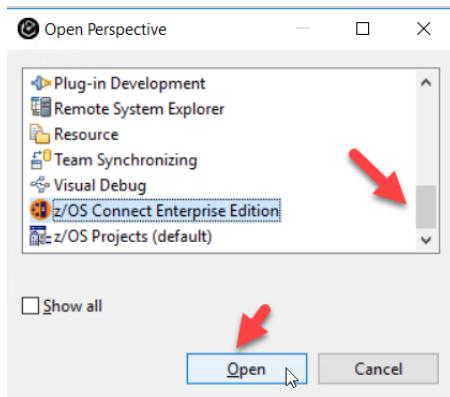
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.



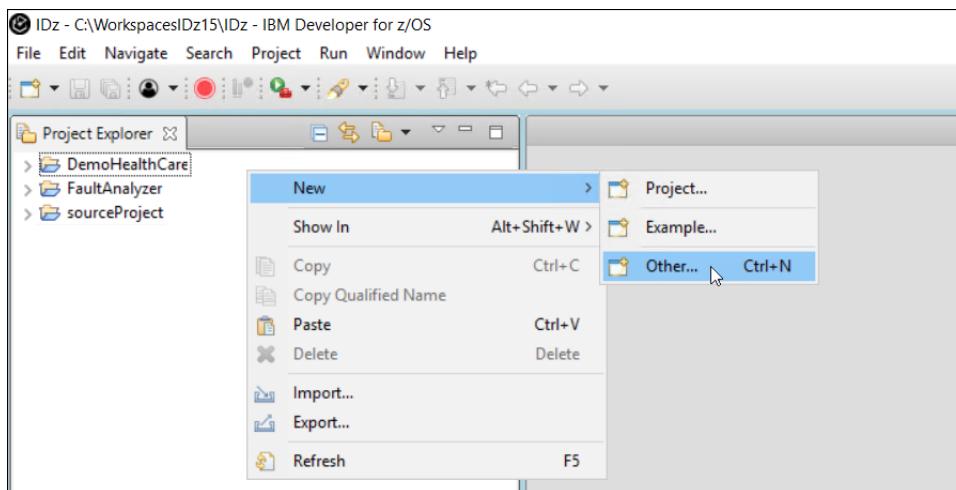
1.1.2 ➡ Open the z/OS Projects perspective by selecting Window > Perspective > Open Perspective > Other...



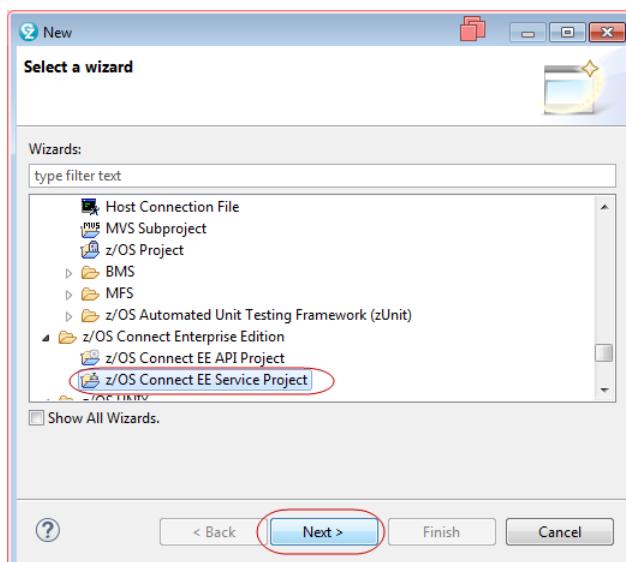
1.1.3 ➡ Scroll down, select **z/OS Connect Enterprise Edition** and click **Open**



1.1.4 ➡ Using the *Project Explorer* view, right click on the blank space and select **New > Other..**



1.1.5 Scroll down, select **z/OS Connect EE Service Project** and click **Next**.

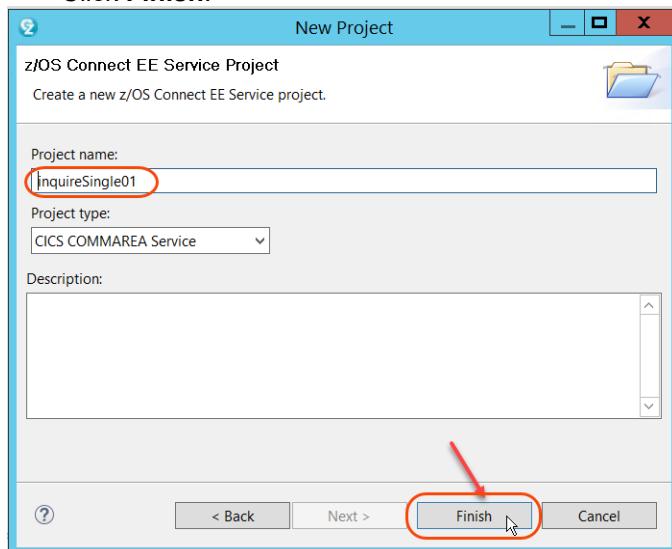


1.1.6 ► Fill in the Project name, this name will be the name of the service.

Use **inquireSingle01**

We know from the Application Discovery lab that the project should be based on a CICS Commarea

► Click **Finish**.

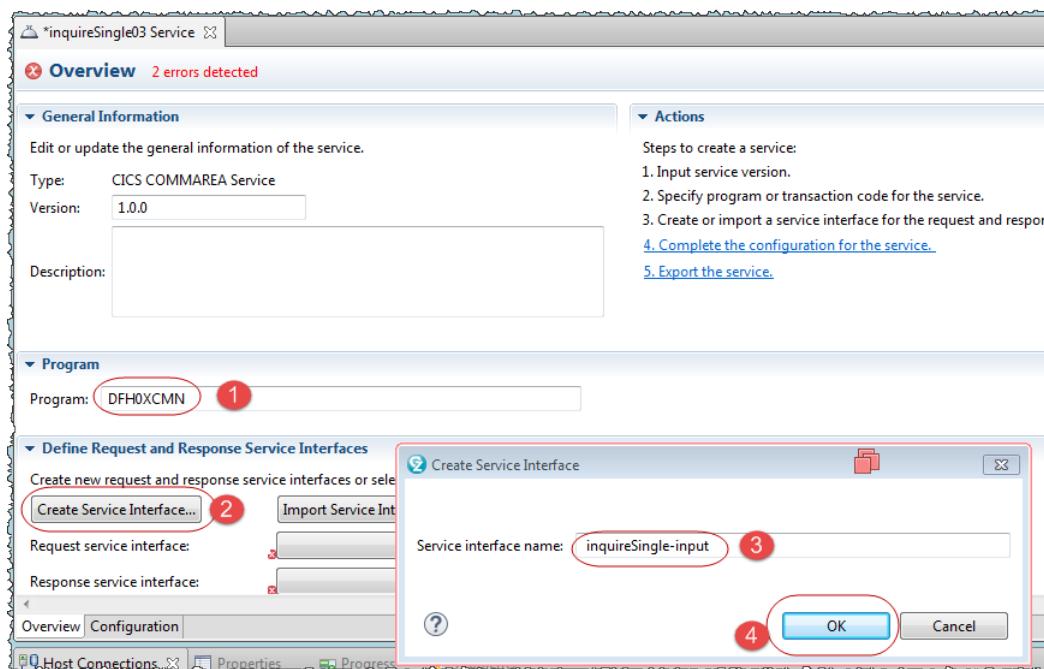


1.2 Service configuration

Once the project is created, there are 4 steps to perform: specify the CICS program to call, import copybook, define service interfaces and specify the IPIC connection reference

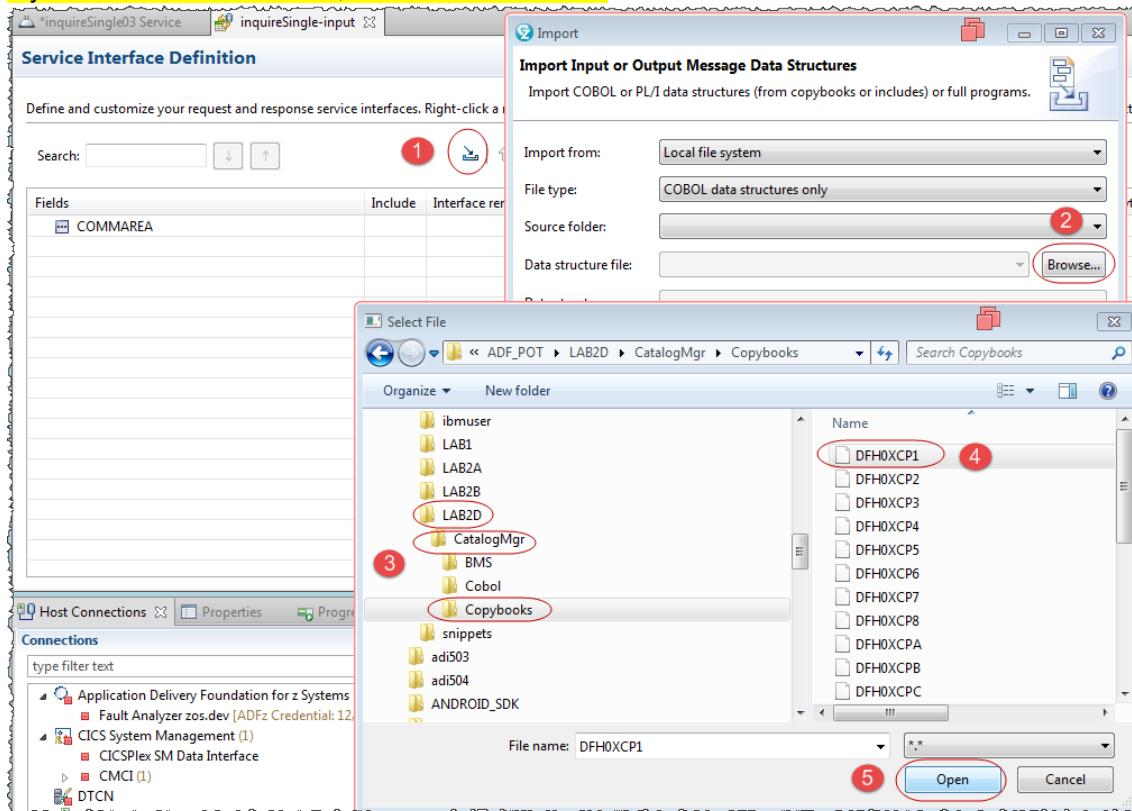
1.2.1 ► Fill in the Program information. the program name **DFH0XCMN** (where 0 is number zero).

► Click on **Create Service Interface...** and call it **inquireSingle-input** and click **OK**

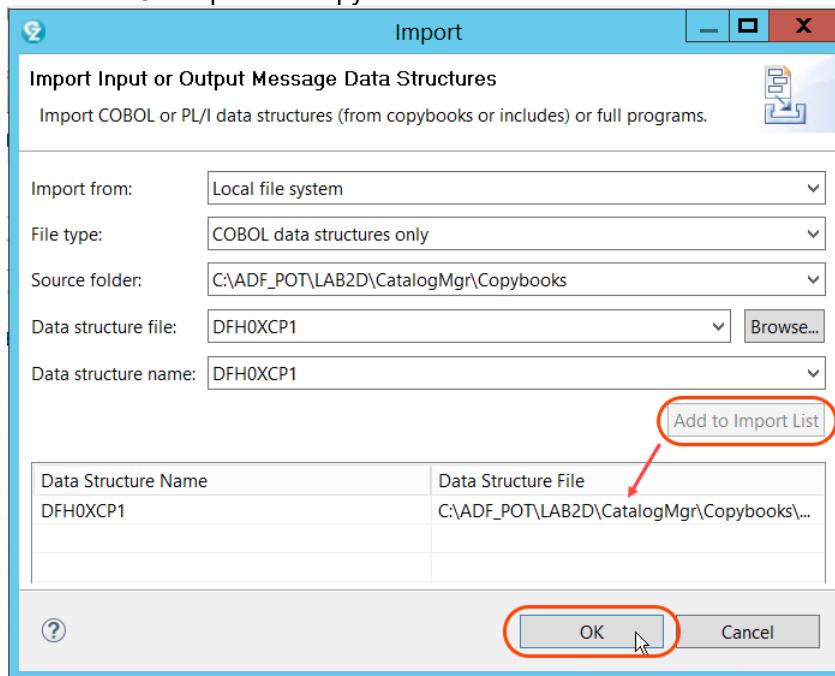


1.2.2 You will arrive in the Service Interface Definition window and you need to Import the copybook **DFH0XCP1**.

- Click on icon  and using the **Browse..** button navigate to **c:\ADF_POT\LAB2D\CatalogMgr\Copybooks\DFH0XCP1** and click **Open**.
If you DO not find this folder, contact the instructor.



- 1.2.3 ► With "DFH0XCP1" specified as the Data structure name, click on **Add to Import List**. Then click **OK** import the copybook.



1.2.4 ►| Expand the data structure to get a better view of the different fields

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	<input checked="" type="checkbox"/>	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefined)	<input checked="" type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL	4	88
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL	4	92
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIMAL	3	96
CA_INQUIRY_RESPONSE_DATA (Redefined)	<input type="checkbox"/>	CA_INQUIRY_RESPONSE_DATA		CHAR	900	99
CA_CAT_ITEM redefines CA_INQUIRY,	<input type="checkbox"/>	CA_CAT_ITEM		ARRAY	900	99
CA_INQUIRE_SINGLE redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_SINGLE		STRUCT	911	88
CA_ITEM_REF_REQ	<input type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60	99
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911	88
CA_USERID	<input type="checkbox"/>	CA_USERID		CHAR	8	88
CA_CHARGE_DEPT	<input type="checkbox"/>	CA_CHARGE_DEPT		CHAR	8	96
CA_ITEM_REF_NUMBER	<input type="checkbox"/>	CA_ITEM_REF_NUMBER		DECIMAL	4	104
CA_QUANTITY_REQ	<input type="checkbox"/>	CA_QUANTITY_REQ		DECIMAL	3	108
FILL_3	<input type="checkbox"/>	FILL_3		CHAR	888	111

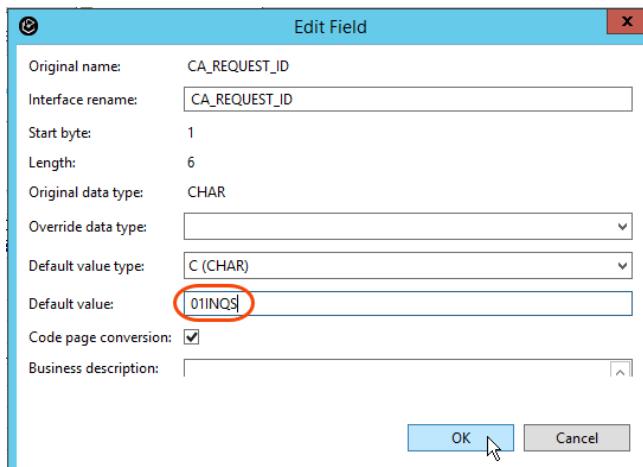
For the **inquireSingle-input** service, define a mapping that:

1.2.5 You will exclude **CA_REQUEST_ID** and assign the default value **01INQS**
(Identified at the Application Discovery Lab)

►| To assign a default value to a field, right-click the field, then choose **Edit field**,

Fields	Include	Interface rename	Default Field Value	Data Type
COMMAREA				
DFH0XCP1				
CA_REQUEST_ID	<input checked="" type="checkbox"/>	CA_REQUEST_ID	<input type="button" value="Edit field"/>	CHAR
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE	<input type="checkbox"/> Exclude field from interface	CHAR
CA_REQUEST_SPECIFIC (Redefined)	<input checked="" type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR

1.2.6 ►| Specify **01INQS** as default value and click **OK**



1.2.7 ► To exclude **CA_REQUEST_ID** you need to uncheck the include box.

The screenshot shows the 'Service Interface Definition' interface. In the 'Fields' table, under the 'Include' column, the checkbox for 'CA_REQUEST_ID' is unchecked (indicated by a red circle). Other fields like 'CA_RETURN_CODE' and 'CA_RESPONSE_MESSAGE' have their checkboxes checked.

Fields	Include	Interface rename	Default Field Value
COMMAREA			
DFH0XCP1			
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE	
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE	
CA_REQUEST_SPECIFIC (Redefined)	<input checked="" type="checkbox"/>	CA_REQUEST_SPECIFIC	

1.2.8 You will also exclude **CA_RETURN_CODE**, **CA_RESPONSE_MESSAGE** (these are response fields). Also will exclude **CA_REQUEST_SPECIFIC** (this field is redefined by CA_INQUIRE_SINGLE)

► To exclude you need to uncheck the include box.

The screenshot shows the 'Service Interface Definition' interface. Several fields have their checkboxes unchecked, indicated by red circles: 'CA_REQUEST_ID', 'CA_RETURN_CODE', 'CA_RESPONSE_MESSAGE', and 'CA_REQUEST_SPECIFIC'. A tooltip 'To import COBOL or PL/I data structure' is visible over the 'CA_REQUEST_ID' row.

Fields	Include	Interface rename	Default Field Value
COMMAREA			
DFH0XCP1			
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE	
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE	
CA_REQUEST_SPECIFIC (Redefined)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC	
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST	
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF	

1.2.9 You will include **CA_ITEM_REF_REQ** that is under **CA_INQUIRE_SINGLE**. This field allow to specify which item information to retrieve.

► Check the include box of **CA_ITEM_REF_REQ**.

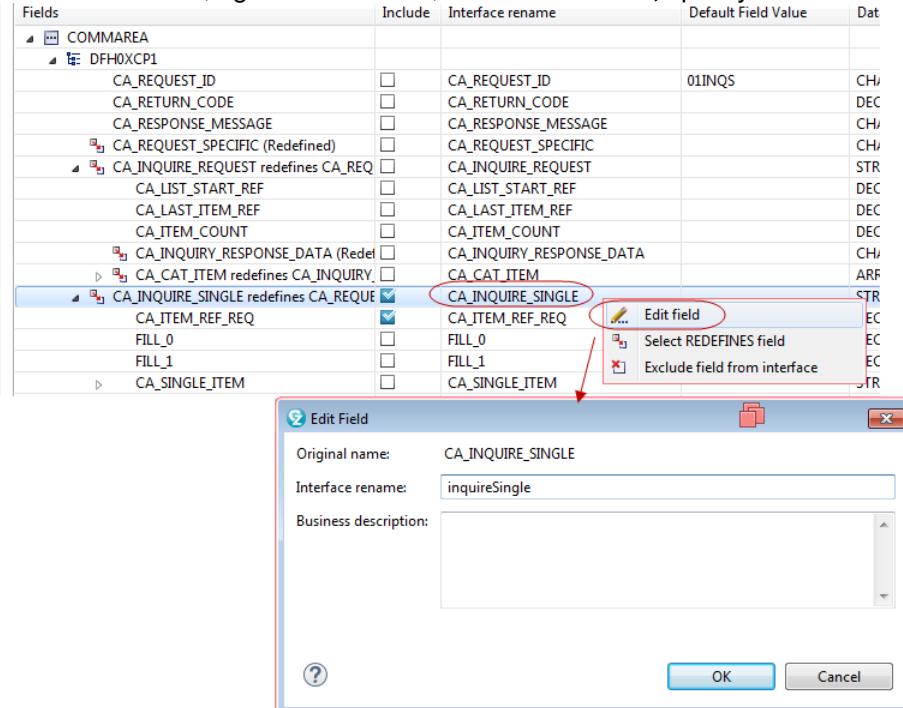
The screenshot shows the 'Service Interface Definition' interface. Under the 'DFH0XCP1' section, the 'CA_INQUIRE_SINGLE' row has its checkbox checked (indicated by a red circle), and the 'CA_ITEM_REF_REQ' row below it also has its checkbox checked (also indicated by a red circle).

Fields	Include	Interface rename	Default Field Value	Data Type
COMMAREA				
DFH0XCP1				
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS	CHAR
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE		DECIM
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR
CA_REQUEST_SPECIFIC (Redefined)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIM
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIM
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIM
CA_INQUIRY_RESPONSE_DATA (Redefined)	<input type="checkbox"/>	CA_INQUIRY_RESPONSE_DATA		CHAR
CA_CAT_ITEM redefines CA_INQUIRY,	<input type="checkbox"/>	CA_CAT_ITEM		ARRAY
CA_INQUIRE_SINGLE redefines CA_REQ	<input checked="" type="checkbox"/>	CA_INQUIRE_SINGLE		STRUCT
CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	CA_ITEM_REF_REQ		DECIM
FILL_0	<input type="checkbox"/>	FILL_0		DECIM
FILL_1	<input type="checkbox"/>	FILL_1		DECIM
CA_SINGLE_ITEM	<input type="checkbox"/>	CA_SINGLE_ITEM		STRUCT
FILL_2	<input type="checkbox"/>	FILL_2		CHAR

1.2.10 It is also a good idea to rename the fields that are exposed by the service interface .

► Rename the **CA_INQUIRE_SINGLE** as **inquireSingle**,

To rename a field, right-click the field, choose **Edit field**, specify a different interface rename and click **OK**.



► Rename the **CA_ITEM_REF_REQ** field as **itemID** (follow the lower/upper case letters)

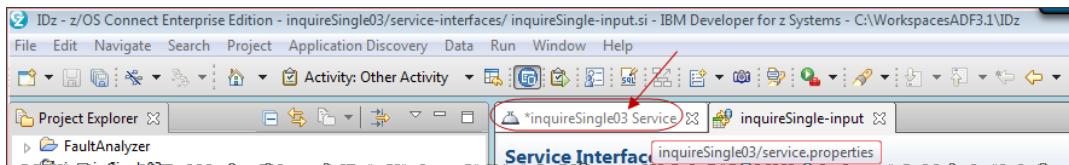
1.2.11 ► Your service interface should look similar to the mapping below

► Save the service interface (**CTRL S**)

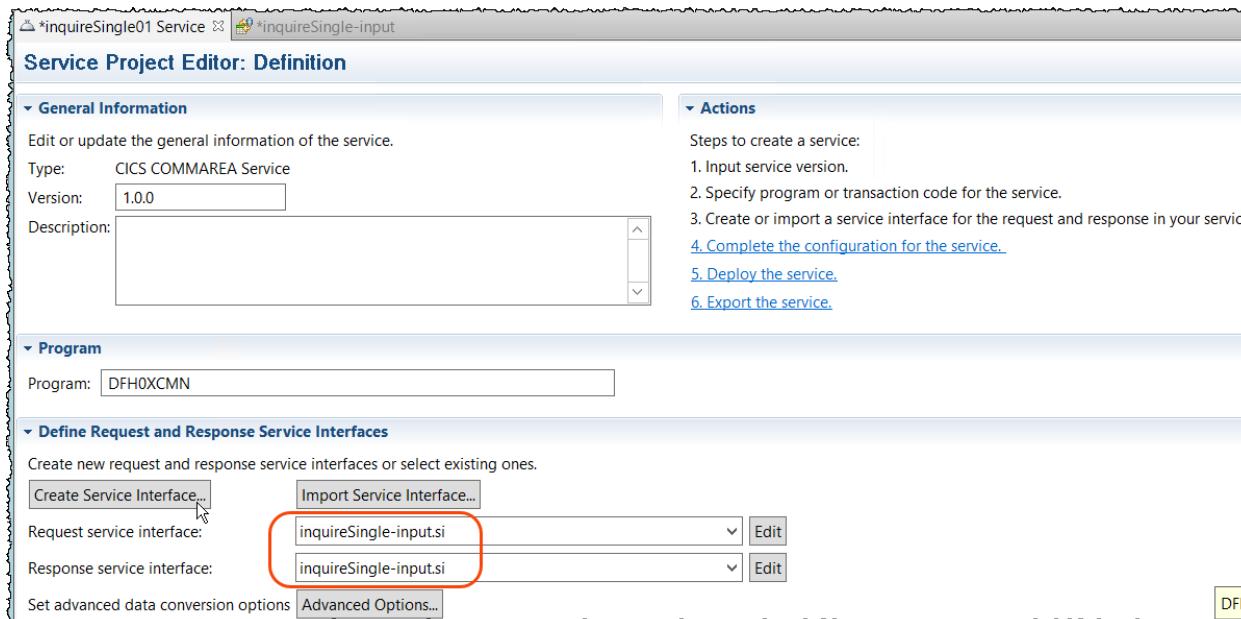
The screenshot shows the 'Service Interface Definition' table in the Catalog Manager application. The 'CA_ITEM_REF_REQ' field has been renamed to 'itemID'. The 'Interface rename' column for this row contains 'itemID'.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length
COMMAREA					
DFH0XCP1					
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS	CHAR	6
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79
CA_REQUEST_SPECIFIC (Redefined)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL	4
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL	4
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIMAL	3
CA_INQUIRY_RESPONSE_DATA (Redefine)	<input type="checkbox"/>	CA_INQUIRY_RESPONSE_DATA		CHAR	900
CA_CAT_ITEM redefines CA_INQUIRY	<input type="checkbox"/>	CA_CAT_ITEM		ARRAY	900
CA_INQUIRE_SINGLE redefines CA_REQ	<input checked="" type="checkbox"/>	CA_INQUIRE_SINGLE		STRUCT	911
CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	itemID		DECIMAL	4
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3
CA_SINGLE_ITEM	<input type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840
CA_ORDER_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911
CA_USERID	<input type="checkbox"/>	CA_USERID		CHAR	8
CA_CHARGE_DEPT	<input type="checkbox"/>	CA_CHARGE_DEPT		CHAR	8
CA_ITEM_REF_NUMBER	<input type="checkbox"/>	CA_ITEM_REF_NUMBER		DECIMAL	4
CA_QUANTITY_REQ	<input type="checkbox"/>	CA_QUANTITY_REQ		DECIMAL	3
FILL_3	<input type="checkbox"/>	FILL_3		CHAR	888

1.2.12 ► Go back to the **inquireSingle01** Service tab clicking on it



1.2.13 Note that the service interface you just defined is used for request and response

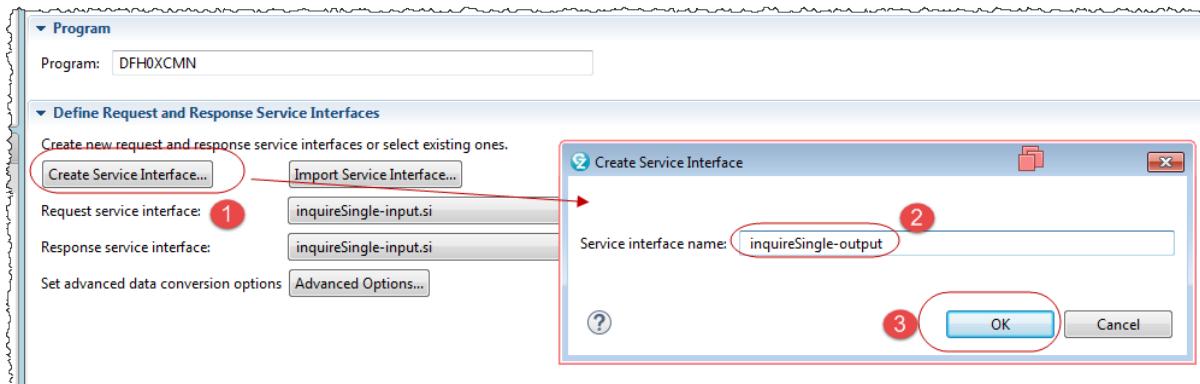


Let's create a new service interface for the output: **inquireSingle-output**.

1.2.14 In the new service interface, you will need to:

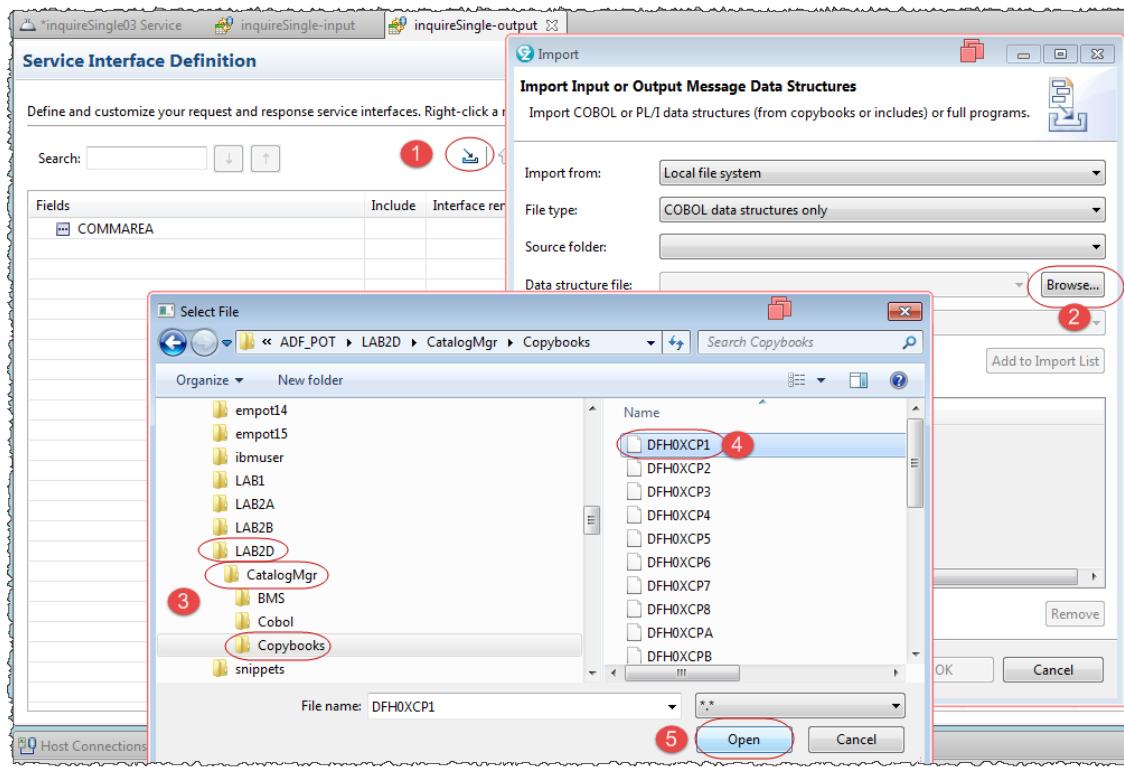
- Import the **DFH0XCP1** copybook
- Include and exclude fields
- Optionally rename exposed fields to give names that make more sense

► Click on **Create Service Interface**, call it **inquireSingle-output** and click **OK**

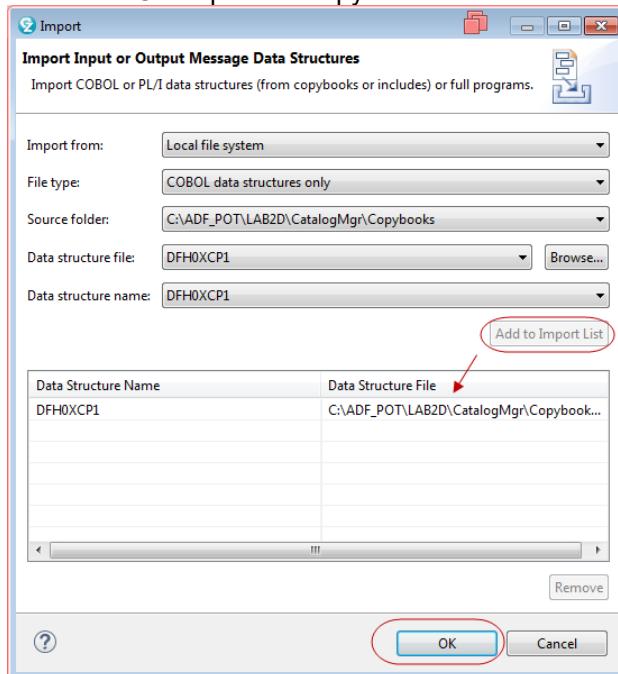


1.2.15 You will arrive in the Service Interface Definition window and you need to Import again the copybook DFH0XCP1.

► Click on icon click on source folder to select the value already entered there and using the **Browse..** button navigate to c:\ADF_POT\LAB2D\CatalogMgr\Copybooks\DFH0XCP1 and click **Open**.



1.2.16 ► With "DFH0XCP1" specified as the Data structure name, click on **Add to Import List**. Then click **OK** import the copybook.



1.2.17 ► Expand the data structure to get a better view of the different fields

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	<input checked="" type="checkbox"/>	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefined)	<input checked="" type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL	4	88
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL	4	92
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIMAL	3	96

For the inquireSingle-output service, define a mapping that:

1.2.18 Excludes CA_REQUEST_ID and CA_REQUEST_SPECIFIC

► To exclude you need to uncheck the include box.

Fields	Include	Interface rename	Default Field Value	Data Type
COMMAREA				
DFH0XCP1				
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID		CHAR
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR
CA_REQUEST_SPECIFIC (Redefined)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIMAL

1.2.19 Include CA_INQUIRE_SINGLE, only the high level record. and CA_SINGLE_ITEM

► To include you need to check the include box.

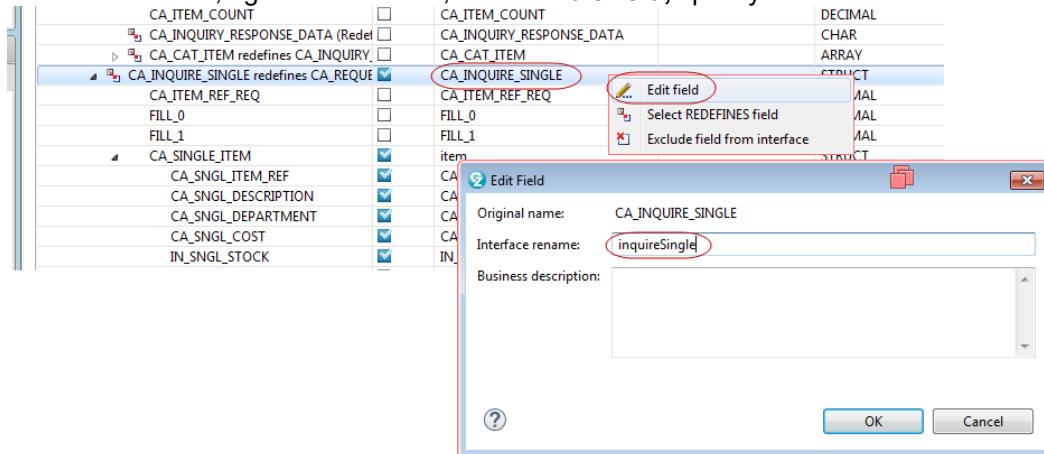
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length
COMMAREA					
DFH0XCP1					
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID		CHAR	6
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79
CA_REQUEST_SPECIFIC (Redefined)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911
CA_INQUIRE_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911
CA_LIST_START_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL	4
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL	4
CA_ITEM_COUNT	<input type="checkbox"/>	CA_ITEM_COUNT		DECIMAL	3
CA_INQUIRY_RESPONSE_DATA (Redefined)	<input type="checkbox"/>	CA_INQUIRY_RESPONSE_DATA		CHAR	900
CA_CAT_ITEM redefines CA_INQUIRY	<input type="checkbox"/>	CA_CAT_ITEM		ARRAY	900
CA_INQUIRE_SINGLE redefines CA_REQ	<input checked="" type="checkbox"/>	CA_INQUIRE_SINGLE		STRUCT	911
CA_ITEM_REF_REQ	<input type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3
CA_SINGLE_ITEM	<input checked="" type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60
CA_SNGL_ITEM_REF	<input checked="" type="checkbox"/>	CA_SNGL_ITEM_REF		DECIMAL	4
CA_SNGL_DESCRIPTION	<input checked="" type="checkbox"/>	CA_SNGL_DESCRIPTION		CHAR	40
CA_SNGL_DEPARTMENT	<input checked="" type="checkbox"/>	CA_SNGL_DEPARTMENT		DECIMAL	3
CA_SNGL_COST	<input checked="" type="checkbox"/>	CA_SNGL_COST		CHAR	6
IN_SNGL_STOCK	<input checked="" type="checkbox"/>	IN_SNGL_STOCK		DECIMAL	4
ON_SNGL_ORDER	<input checked="" type="checkbox"/>	ON_SNGL_ORDER		DECIMAL	3
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840
CA_ORDER_REQUEST redefines CA_REQ	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911
CA_USERID	<input type="checkbox"/>	CA_USERID		CHAR	8

1.2.20 The best practice is to rename the fields to better expose the service interface..
But if you are running late or don't want to spend time here just jump to 1.2.22

(Optional) Rename the fields that are exposed by the service interface .

►► Rename the **CA_INQUIRE_SINGLE** as **inquireSingle**,

To rename a field, right-click the field, choose **Edit field**, specify a different interface rename and click **OK**.



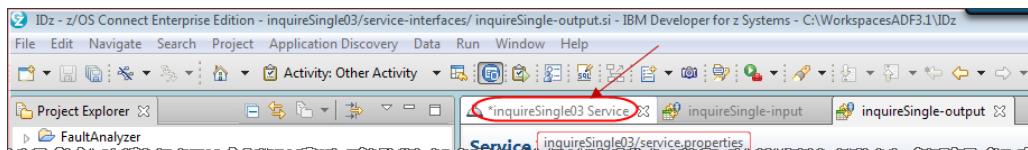
1.2.21 ►► (Optional) Rename the **CA_RETURN_CODE** as **resultCode**, **CA_RESPONSE_MESSAGE** as **responseMessage** **CA_SINGLE_ITEM** as **item**, **CA_SNGL_ITEM_REF** as **itemID**, **CA_SNGL_ITEM_DESCRIPTION** as **description**, **CA_SNGL_ITEM_DEPARTMENT** as **department**, **CA_SNGL_COST** as **cost**, **IN_SNGL_STOCK** as **stock** and **ON_SNGL_ORDER** as **onOrder**

►► Your service interface should look similar to the mapping below

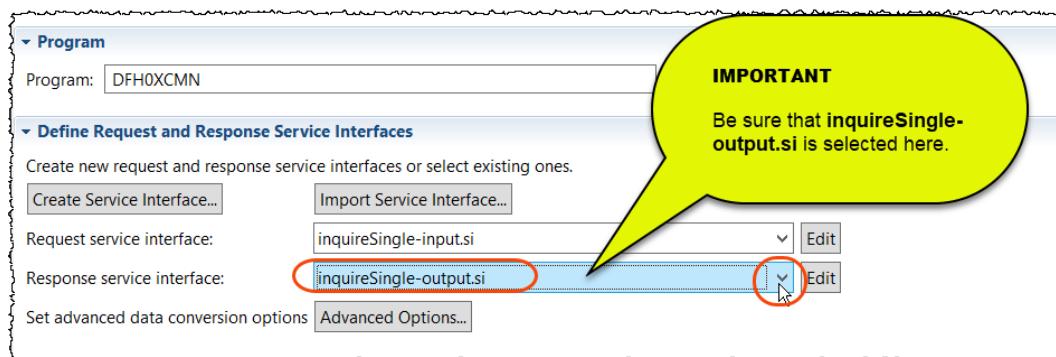
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length
COMMAREA					
DFH0XCP1					
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID		CHAR	6
CA_RETURN_CODE	<input checked="" type="checkbox"/>	resultCode		DECIMAL	2
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	responseMessage		CHAR	79
CA_REQUEST_SPECIFIC (Redefined)					
CA_INQUIRE_REQUEST redefines CA_REQ					
CA_LIST_START_REF	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911
CA_LAST_ITEM_REF	<input type="checkbox"/>	CA_LIST_START_REF		DECIMAL	4
CA_ITEM_COUNT	<input type="checkbox"/>	CA_LAST_ITEM_REF		DECIMAL	4
CA_INQUIRY_RESPONSE_DATA (Redefined)					
CA_CAT_ITEM redefines CA_INQUIRY					
CA_INQUIRE_SINGLE redefines CA_REQ	<input checked="" type="checkbox"/>	inquireSingle		STRUCT	911
CA_ITEM_REF_REQ	<input type="checkbox"/>	CA_INQUIRE_SINGLE		DECIMAL	4
FILL_0	<input type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4
FILL_1	<input type="checkbox"/>	FILL_0		DECIMAL	3
CA_SINGLE_ITEM	<input checked="" type="checkbox"/>	item		STRUCT	60
CA_SNGL_ITEM_REF	<input checked="" type="checkbox"/>	itemID		DECIMAL	4
CA_SNGL_DESCRIPTION	<input checked="" type="checkbox"/>	description		CHAR	40
CA_SNGL_DEPARTMENT	<input checked="" type="checkbox"/>	department		DECIMAL	3
CA_SNGL_COST	<input checked="" type="checkbox"/>	cost		CHAR	6
IN_SNGL_STOCK	<input checked="" type="checkbox"/>	stock		DECIMAL	4
ON_SNGL_ORDER	<input checked="" type="checkbox"/>	onOrder		DECIMAL	3
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840
CA_ORDER_REQUEST redefines CA_REQ					
CA_USERID		CA_ORDER_REQUEST		STRUCT	911
		CA_USERID		CHAR	8

1.2.22 ► Save the **inquireSingle-output** service interface (**CTRL S**)

1.2.23 ► Go back to the **inquireSingle01 Service** tab by clicking on it



1.2.24 ► Click on the second drop down ("Response service interface") and select **inquireSingle-output.si**

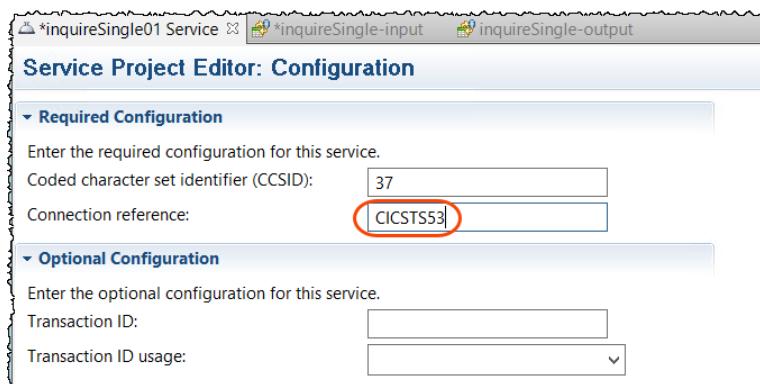


1.2.25 ► On the **inquireSingle Service** tab, switch to the "Configuration" part by clicking the sub-tab (bottom of the window).



1.2.26 ► Type **CICSTS53** for the **Connection reference**

This connection name is a logical name that represents the IPIC connection to a CICS region.



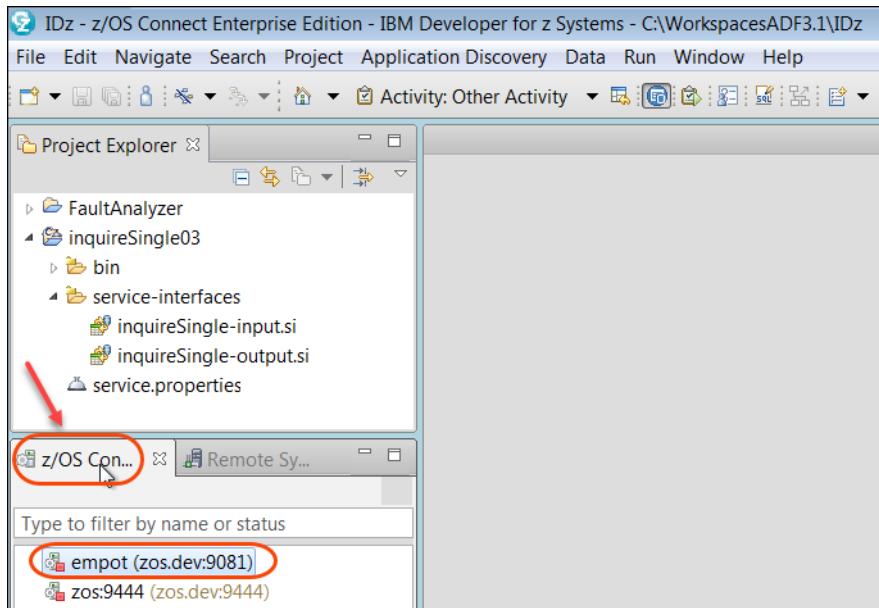
1.2.27 ► Save the service interface (**CTRL S**)

1.2.28 ► Use **Ctrl + Shift + F4** to close all opened editors

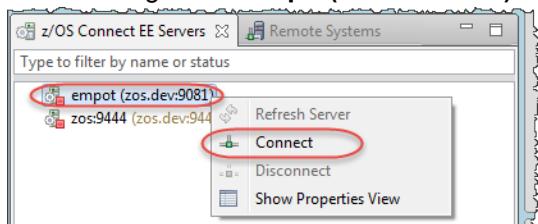
1.3 Connecting to a z/OS Connect EE server

You will connect to the z/OS Connect EE server to deploy the service.

1.3.1 ► Still using the *z/OS Connect Enterprise Edition* perspective click on **z/OS Connect EE Servers** view.
You should see the connection **zos.dev:9081** already defined



1.3.2 ► Right click **empot(zos.dev:9081)** and select **Connect**.



In case of connection errors...

If you have errors during the connection it could be because z/OS connect is not running.
Call the instructor.

Instructor: use the z/OS master console or SDSF DA and verify that you have the **BBGZANGL** and **BAQSTRT** started tasks running.

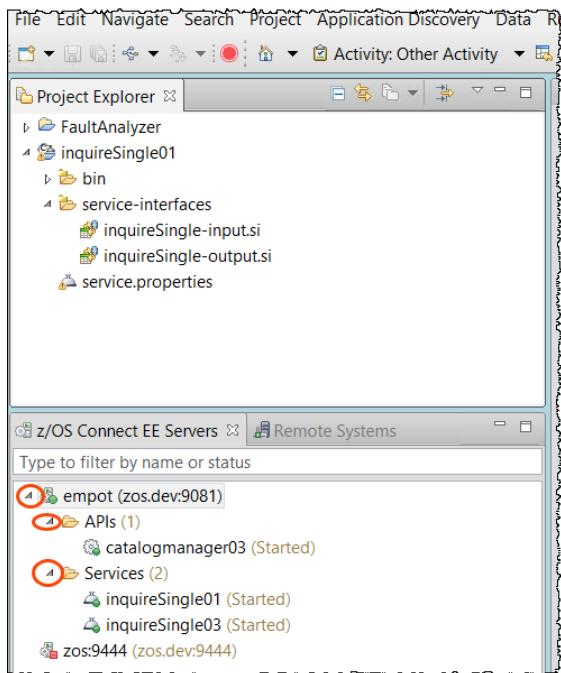
If not: issue a **S BBGZANGL**, then **S BAQSTRT**

For BAQSTRT, make sure you see the message "The server server1 is ready to run a smarter planet"



```
Launching Server30 (z/OS Connect 03.00.00, WebSphere Application Server 7.0.0.
YAUDIT  " CWWKE0001I: The server server30 has been launched.
YAUDIT  " BAQR7000I: z/OS Connect API package catalogmanager03 installed succe
YAUDIT  " BAQR0000I: z/OS Connect Enterprise Edition version 3.0.0.0 (20170602)
YAUDIT  " BAQR7043I: z/OS Connect EE service archive inquireSingle03 installed
YAUDIT  " CWWKF0015I: The server has the following interim fixes active in the
YAUDIT  " CWWKF0012I: The server installed the following features: Yssl-1_0, a
YAUDIT  " CWWKF0011I: The server server30 is ready to run a smarter planet
YAUDIT  " CWWKT0016I: web application available (default_host): http://cloud.p
```

1.3.3 ► Using z/OS Connect EE Servers view on left/bottom, expand **empot/API** and **-Services** and you should see something similar to the following screen capture below.
(it will depend on how many services and APIs have already been deployed there).



1.4 Deploying the Service to z/OS Connect

You can deploy your service to the server directly from the API toolkit if the server connection is already properly configured.

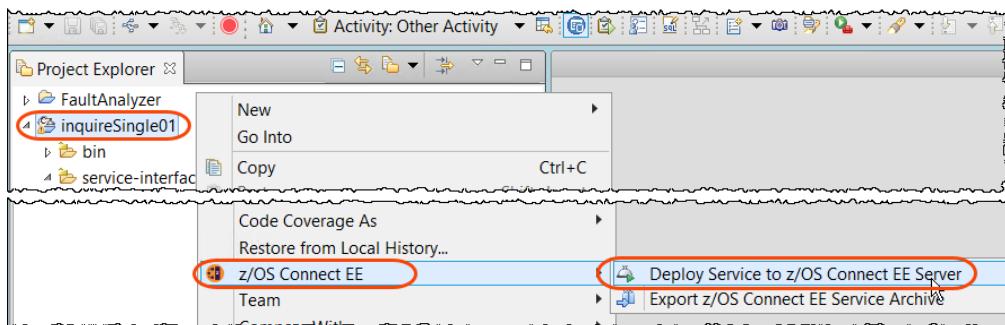


Automatic Deploy versus Manual deploy

Depending on the level of the z/OS Connect EE V3 installed you could do the Automatic deploy.

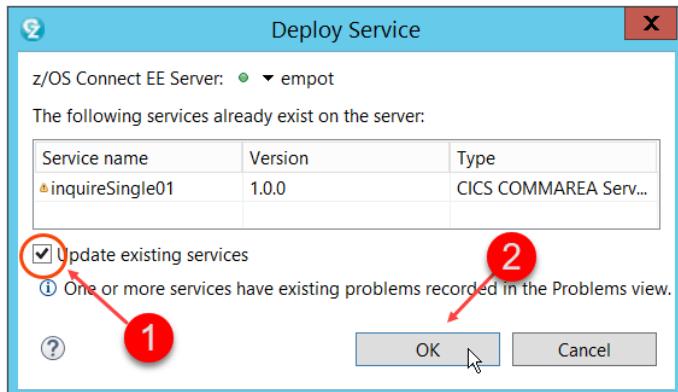
1.4.1 We will deploy the service to the z/OS Connect EE server.

► Using *Project Explorer* view, right-click on the project **inquireSingle01**, then **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**.



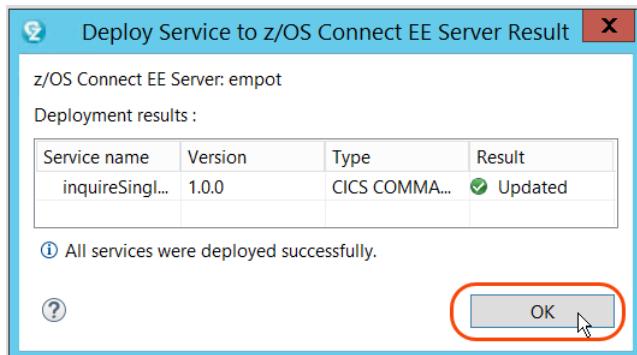
1.4.2 You will get the following *Deploy Service* dialog.

► Check **Update existing services** and click **OK**.



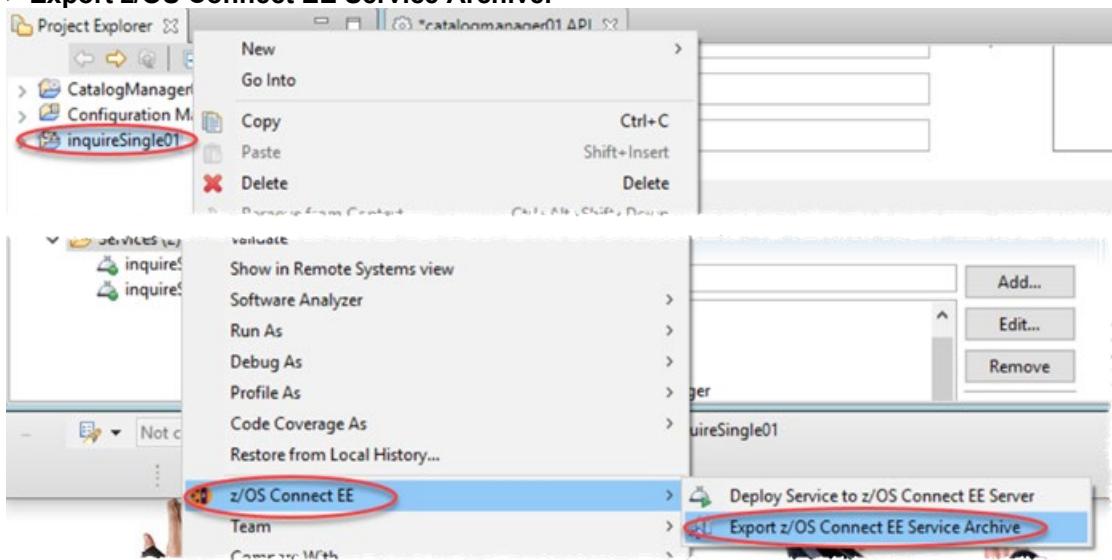
1.4.3 You will get the *Deploy Service ... Result* dialog. The result could be “Created” or “Updated”.

► Click **OK**.

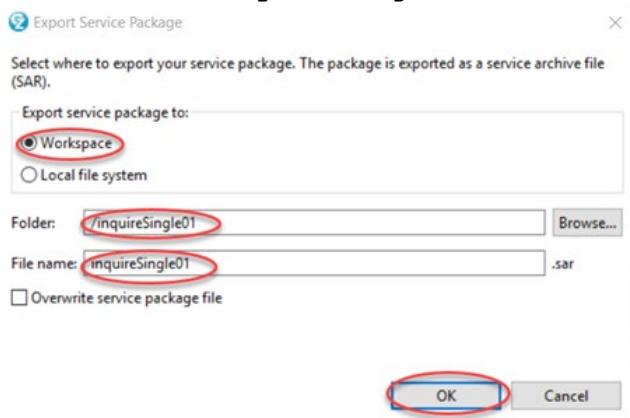


1.4.4 You will now export the service archive file for use in the next section.

► Using the *Project Explorer* view, right-click on the project **inquireSingle01**, then **z/OS Connect EE** > **Export z/OS Connect EE Service Archive**.



- In the *Export Service Package* dialog, select **Workspace** as the export target, with `/inquireSingle01` as the *Folder* and `inquireSingle01` as the *File name*. Click **OK**.

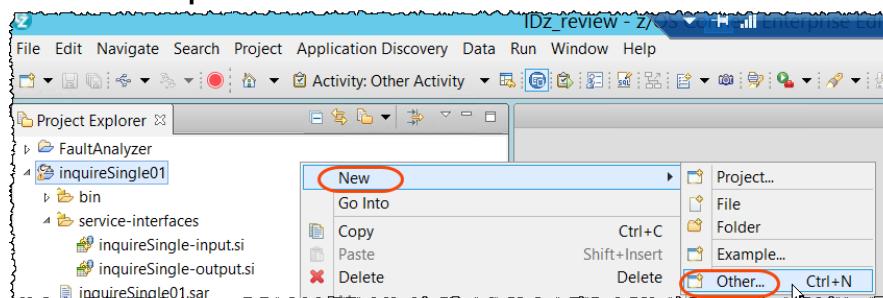


Section 2 – Create and configure a z/OS Connect API Project

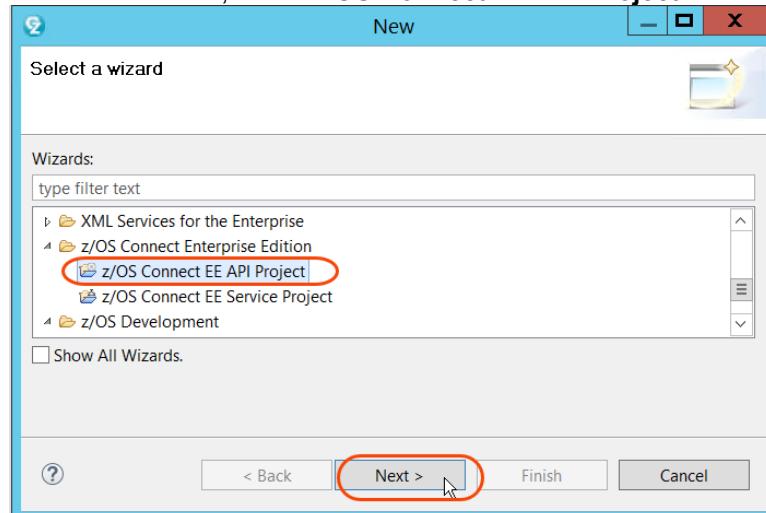
In this step, you will create and API to invoke a z/OS Connect Service created before.

2.1 Create the z/OS Connect API project and import the service

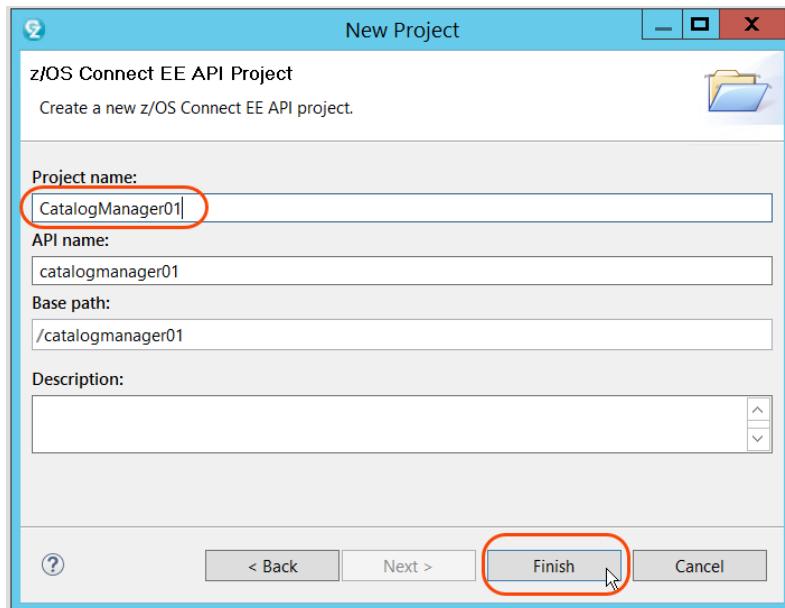
- 2.1.1. ► Still under *z/OS Connect Enterprise Edition* perspective and using the *Project Explorer* view, right click on the blank space and select **New > Other..**



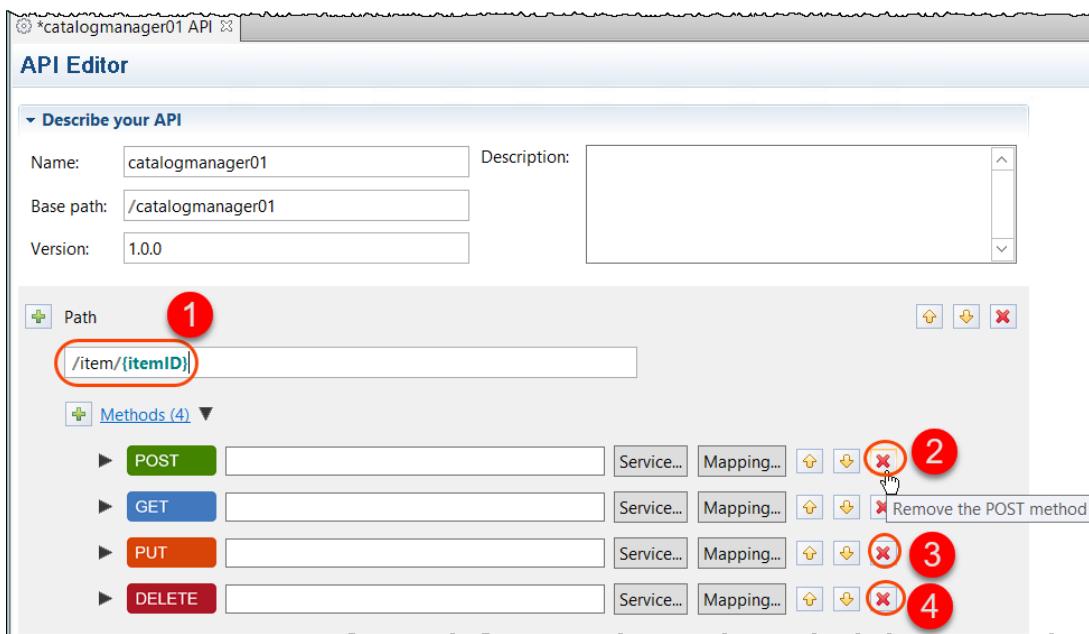
- 2.1.2. Scroll down, select **z/OS Connect EE API Project** and click **Next**



- 2.1.3 ► Fill in the Project name, this name will be the name of the API in lowercase.
 Use **CatalogManager01**
 ► Click **Finish**.

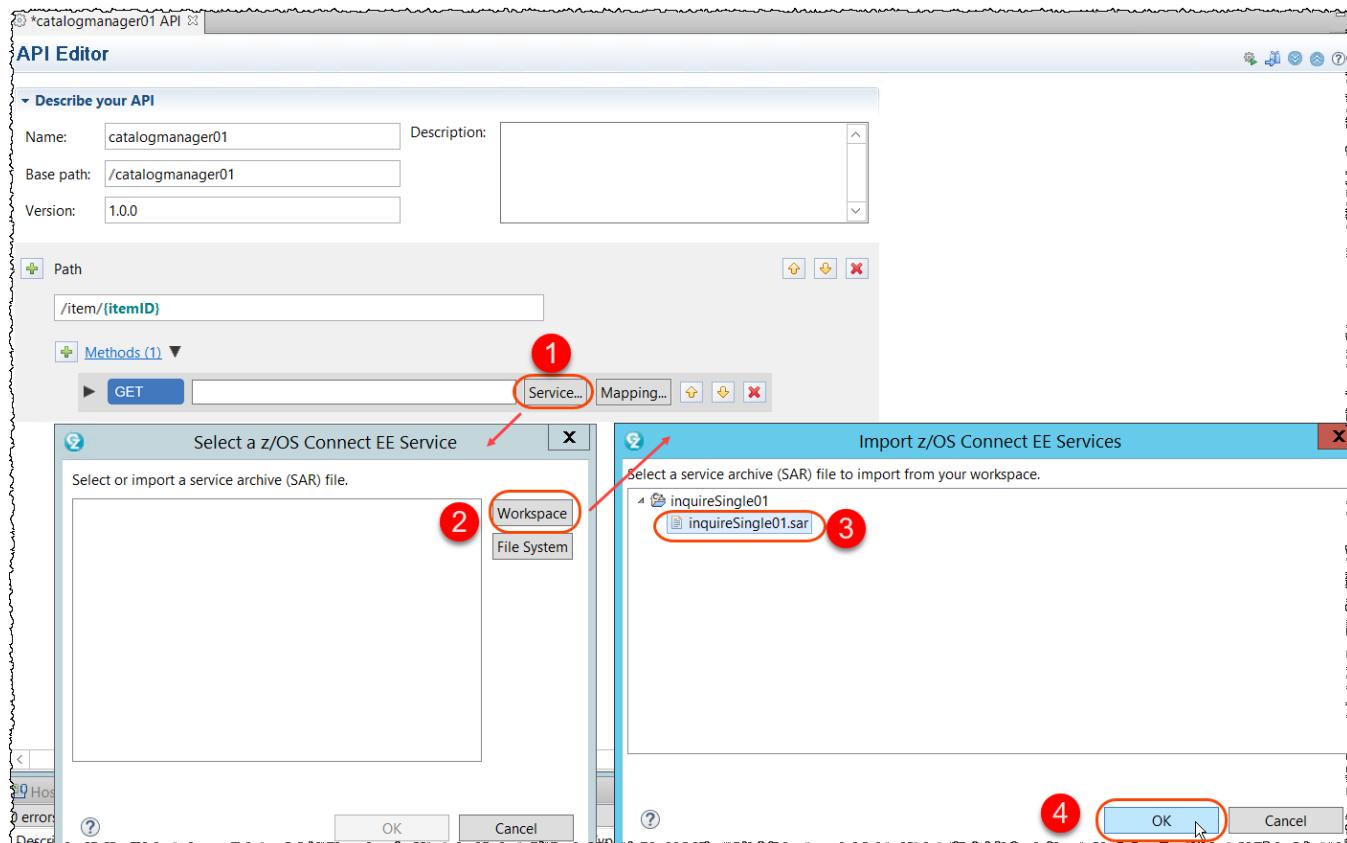


- 2.1.4 You will arrive in the z/OS Connect EE API Editor
 ► in Path specify **/item/{itemID}**
 Tip: (itemID was the interface renamed on step 1.2.10)
 ► You want only GET a value from the service. Use the icon to delete the methods **POST**, **PUT** and **DELETE**



- 2.1.5 ► Click **OK** for the dialogs confirming the delete.

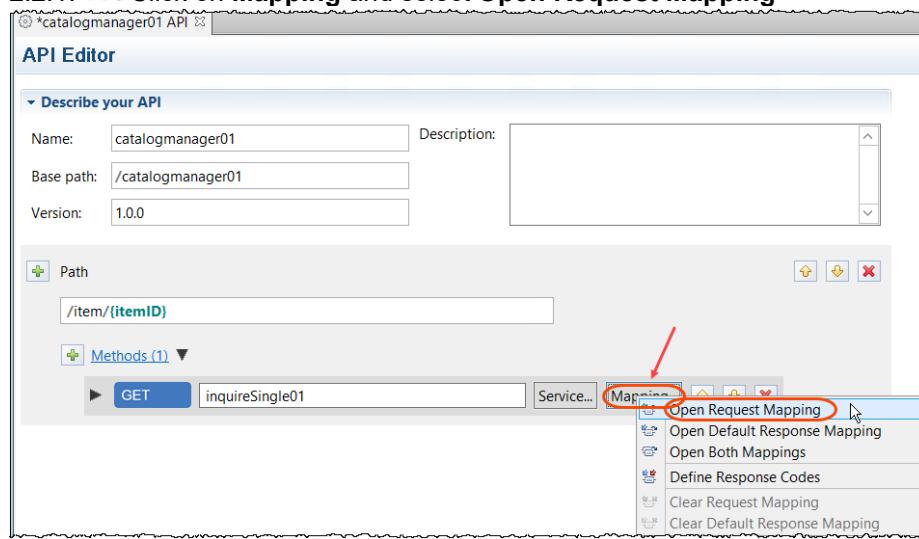
- 2.1.6 ► Click on **Service...** button and on **Select a z/OS Connect EE Service** click **Workspace** button.
 Navigate to your Service Project and find the SAR file (**inquireSingle01.sar**) that you exported before and
 ► Click **OK** three times.



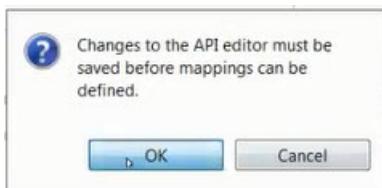
2.2 Mapping the API to the service

Now we need to do the mappings

- 2.2.1. ► Click on **Mapping** and select **Open Request Mapping**

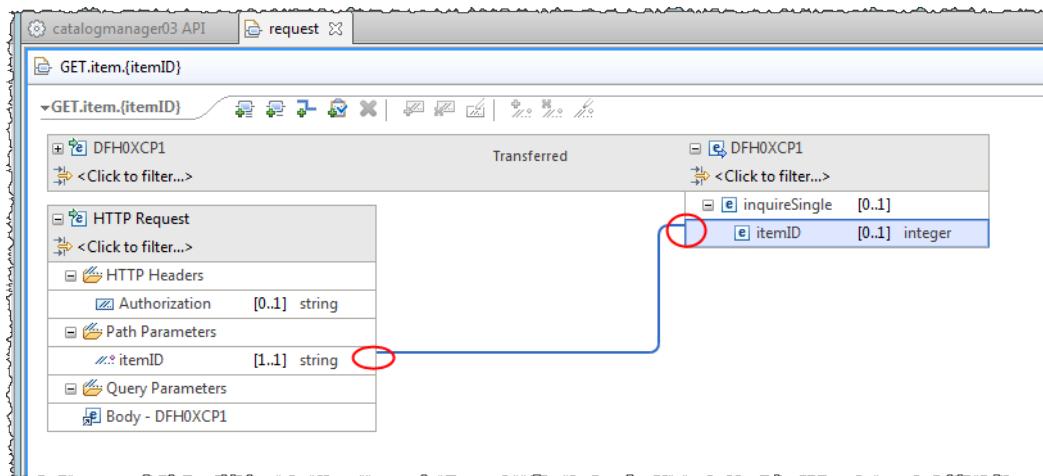


2.2.2 ► A dialog will ask for save the changes. Click **OK** to save it..

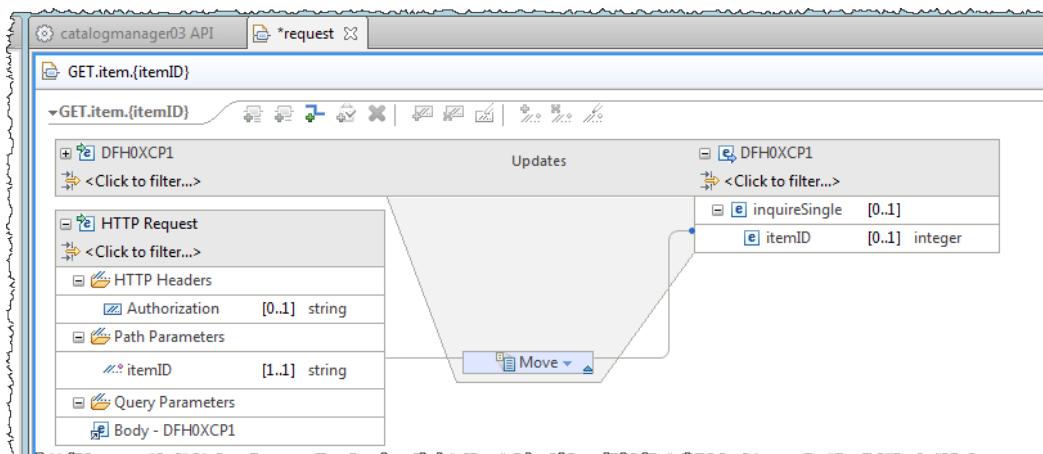


2.2.3 The Mapping editor opens

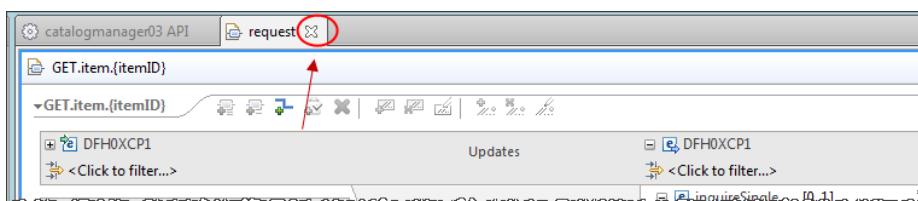
► Expand **inquireSingle** on right.
Use the mouse to drag **itemID** and drop to **itemID** under **Path Parameters**



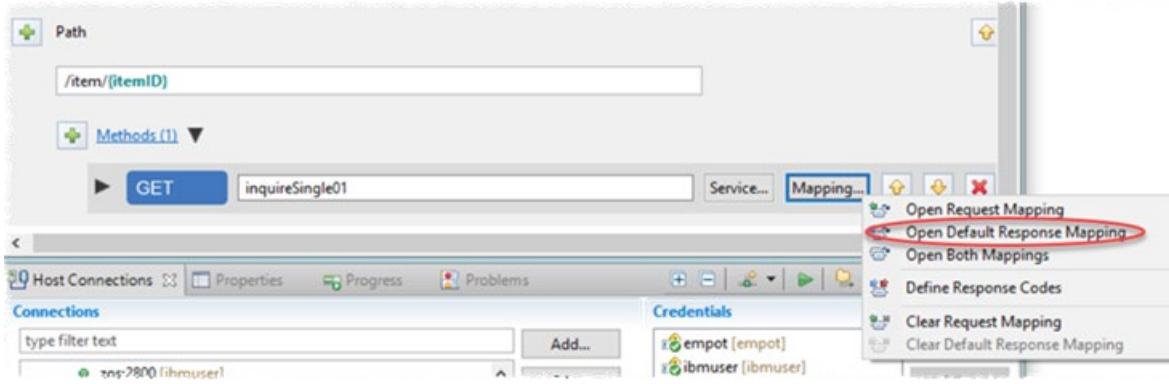
It shows as:



2.2.4 ► Use **Ctrl + S** to save this mapping and **close** this mapping editor



2.2.5 ➡ Back to API editor click on **Mapping...** and select **Open Default Response Mapping**



2.2.6 ➡ On the response expand the fields that we will map on the left and on the right. We could map each one as we did, but we can keep as is so it will do an **implicit mappings** and **maps everything**. It will save time.

Transferred		Body - DFH0XCP1
CA_RETURN_CODE [0..1] integer	<Click to filter...>	CA_RETURN_CODE [0..1] integer
CA_RESPONSE_MESSAGE [0..1] string	<Click to filter...>	CA_RESPONSE_MESSAGE [0..1] string
CA_INQUIRE_SINGLE [0..1]	<Click to filter...>	CA_INQUIRE_SINGLE [0..1]
CA_SINGLE_ITEM [0..1]	<Click to filter...>	CA_SINGLE_ITEM [0..1]
CA_SNGL_ITEM_REF [0..1] integer	<Click to filter...>	CA_SNGL_ITEM_REF [0..1] integer
CA_SNGL_DESCRIPTION [0..1] string	<Click to filter...>	CA_SNGL_DESCRIPTION [0..1] string
CA_SNGL_DEPARTMENT [0..1] integer	<Click to filter...>	CA_SNGL_DEPARTMENT [0..1] integer
CA_SNGL_COST [0..1] string	<Click to filter...>	CA_SNGL_COST [0..1] string
IN_SNGL_STOCK [0..1] integer	<Click to filter...>	IN_SNGL_STOCK [0..1] integer
ON_SNGL_ORDER [0..1] integer	<Click to filter...>	ON_SNGL_ORDER [0..1] integer

2.2.7 ➡ Use **Ctrl + Shift + F4** to close all editors

Section 3 – Deploy the API

In this step, you would deploy the API to z/OS Connect EE Server. Your API may only contain a single path, but for an initial experience with the API Editor and the APIs deployment, it is sufficient.

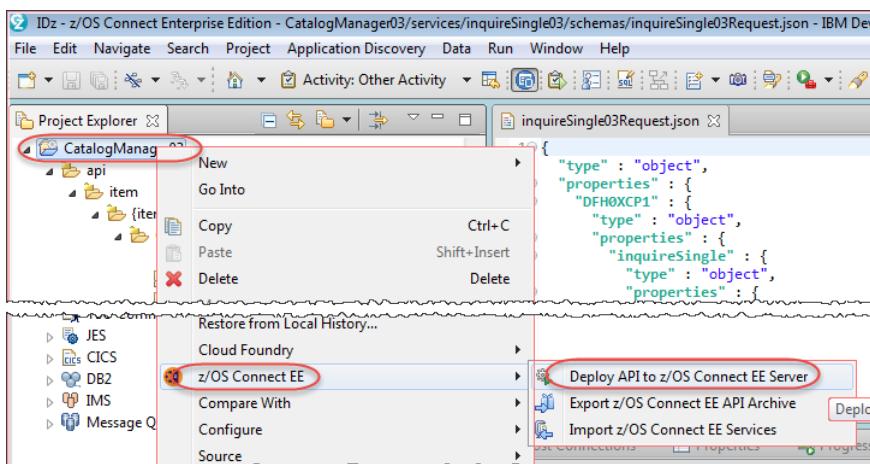
3.1 Deploy API to z/OS

- 3.1.1. Still under *z/OS Connect Enterprise Edition* perspective and using the *Project Explorer* view, fully expand the **CatalogManager01** project and double click on **inquireSingle01Request.json**. You can see the json request created

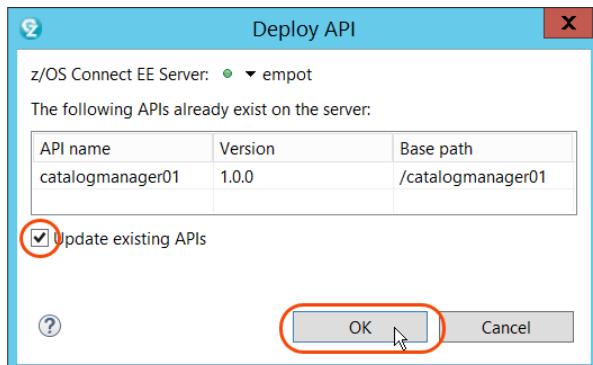
```

1<[{"type": "object", "properties": {"DFH0XCP1": {"type": "object", "properties": {"inquireSingle": {"type": "object", "properties": {"itemID": {"minimum": 0, "maximum": 9999, "type": "integer"}}, "type": "array", "minItems": 1, "maxItems": 1}}}, "type": "array", "minItems": 1, "maxItems": 1}}
```

- 3.1.2. To deploy this API, right click on **CatalogManager01** and select *z/OS Connect EE -> Deploy API to z/OS Connect EE Server*

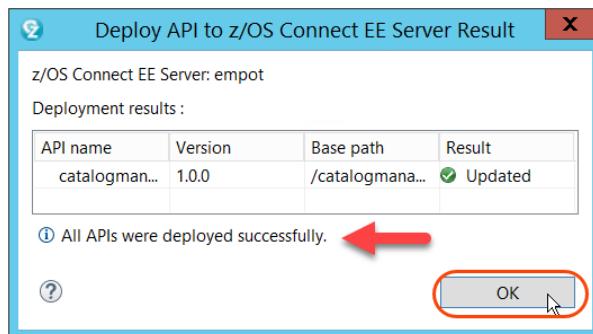


3.1.3 ➡ Select Update existing APIs and click **OK**

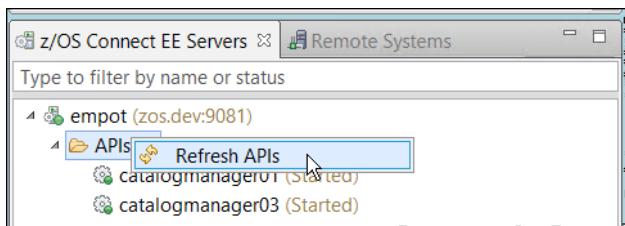


3.1.4 ➡ Once deployed, the result is displayed

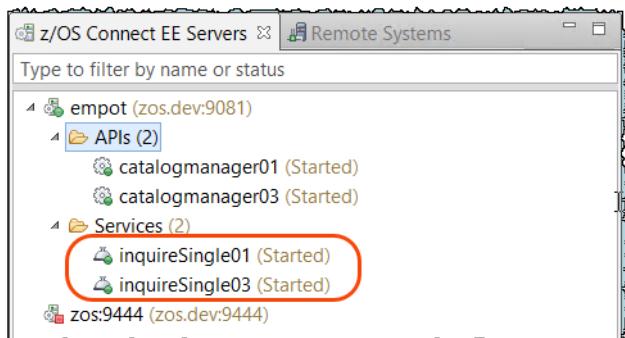
Check that “*All APIs were deployed successfully*” then click on **OK**:



3.1.5 ➡ Using the *z/OS Connect EE Servers* view (left bottom), right click on **APIs** and select **Refresh APIs**.



3.1.6 You should see the list of APIs already deployed to the *z/OS Connect EE Server*



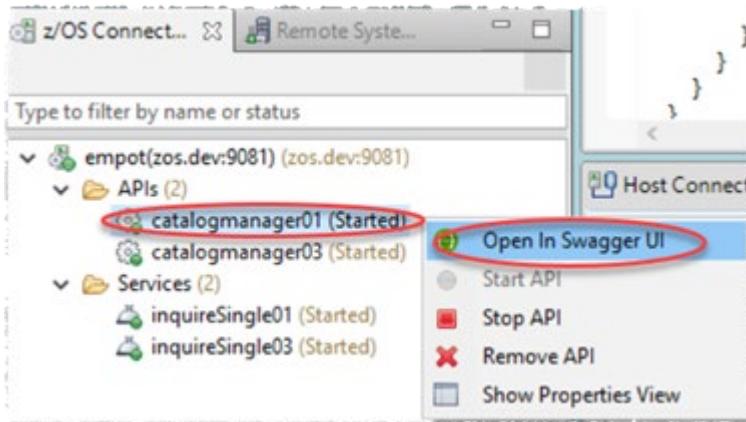
Section 4 – Testing deployed API using Swagger

In this step, you will test the deployed API using Swagger.

Swagger is the world's largest framework of API developer tools for the OpenAPI Specification(OAS), enabling development across the entire API lifecycle, from design and documentation, to test and deployment.

4.1 Invoking Swagger to test the API

- 4.1.1. ► Using the z/OS Connect EE Servers view, right click on the API **catalogmanager01** that you deployed and select **Open In Swagger UI**.



4.1.2. It will open the Swagger UI inside IDz.

- Click on **catalogmanager01** to see the **GET API**



You will see:



4.1.3 ► Click GET /items/{itemID}

You will see the description for the operation: the elements to send for the request and the format of the response. Test your inquire single operation by filling in the itemID (**10, 20, 30, ..., 210**), then click “**Try it out!**”.

catalogmanager01

catalogmanager01

GET /item/{itemID}

Show/Hide | List Operations | Expand O;

Response Class (Status 200)

OK

Model Example Value

```
{
  "DFH0XCP1": {
    "returnCode": 0,
    "department": 0,
    "cost": "string",
  }
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
itemID	10		path	string

Try it out!

[BASE URL: /catalogmanager01 , API VERSION: 1.0.0]

4.1.4 ►| Scroll down and check the results

The screenshot shows the Postman interface with two tabs open: 'inquireSingle03Request.json' and 'catalogmanager03.json'. In the 'catalogmanager03.json' tab, the 'itemID' field is set to '10' (highlighted with a red arrow). Below it, the 'Authorization' field is empty. A 'Try it out!' button is visible. The 'Curl' section contains a command: `curl -X GET --header 'Accept: application/json' 'http://zos.dev:9081/catalogmanager03/item/10'`. The 'Request URL' is `http://zos.dev:9081/catalogmanager03/item/10`. The 'Request Headers' field contains: `{ "Accept": "application/json" }`. The 'Response Body' section displays a JSON response with a red box highlighting the item details:

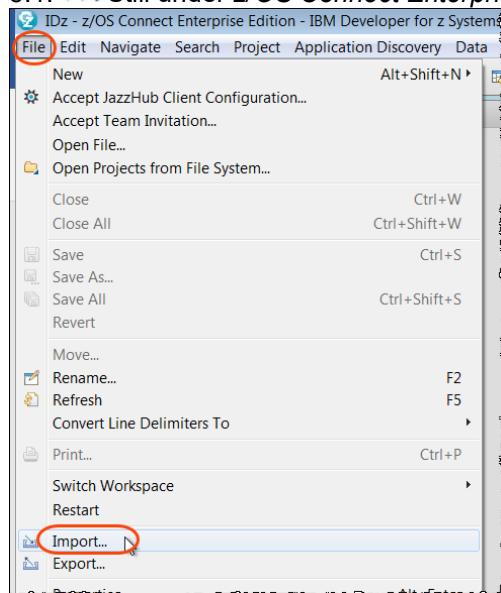
```
{
  "DFH0XCP1": {
    "responseMessage": "RETURNED ITEM: REF =0010",
    "returnCode": 0,
    "inquireSingle": {
      "item": {
        "stock": 5,
        "itemID": 10,
        "department": 10,
        "description": "Ball Pens Black 24pk",
        "onOrder": 0,
        "cost": "002.90"
      }
    }
  }
}
```

4.1.5 ►| Use Ctrl + Shift + F4 to close all opened editors.

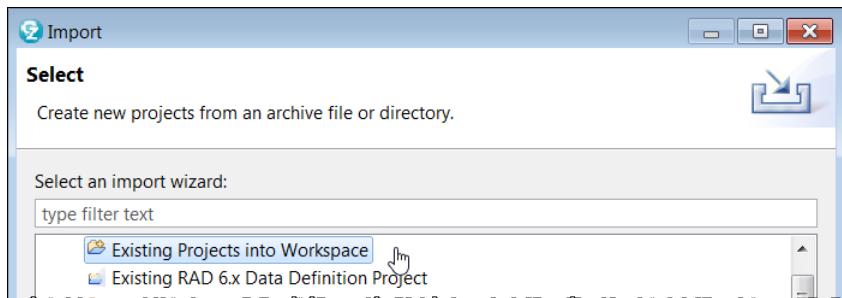
Section 5 –(OPTIONAL) Import the solution

In case you did not complete the lab you can import a solution for an userid (empot03). Even if you are other user id this solution will work for you.

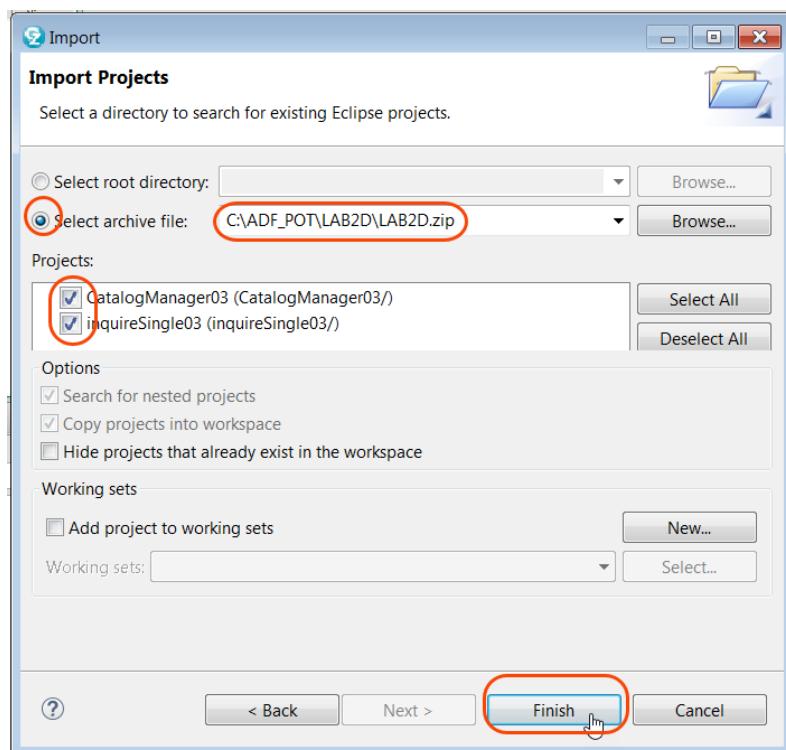
5.1. ►| Still under z/OS Connect Enterprise Edition perspective select File > Import...



5.2.  **Select Existing Projects into Workspace** and click **Next**



5.3.  **Select archive file** and use Browse to find the exported version **LAB2D.zip** and select **CatalogManager03** and **inquireSingle03** Click **Finish**



5.4.  Proceed to item 1.4.1 to deploy the Service to z/OS and then go to Section 3 to deploy the API.I

LAB 5 (OPTIONAL) Create UrbanCode Deploy infrastructure and deploy to z/OS

Updated April 12 2021 by Regi. (Reviewed by Wilbert Kho)

Abstract:

You will create and deploy an existing COBOL CICS application using UrbanCode Deploy to z/OS

This Lab has 3 parts.

Part 1 – Create the UrbanCode Deploy application infrastructure.

In this section, you will design the actual deployment process; you need to prepare the application infrastructure in UrbanCode Deploy (UCD). First, you will create a component and add component version, then you will define resources and map them to an environment, and finally you will create an application and add environment and component to it.

Part 2 - Create the UrbanCode Deploy deployment processes.

On this part you will create a deployment process for your component and the application process that uses the component process to deploy the component.

Part 3 – Deploy the application to z/OS CICS

On this part you will deploy the Application to the z/OS CICS.



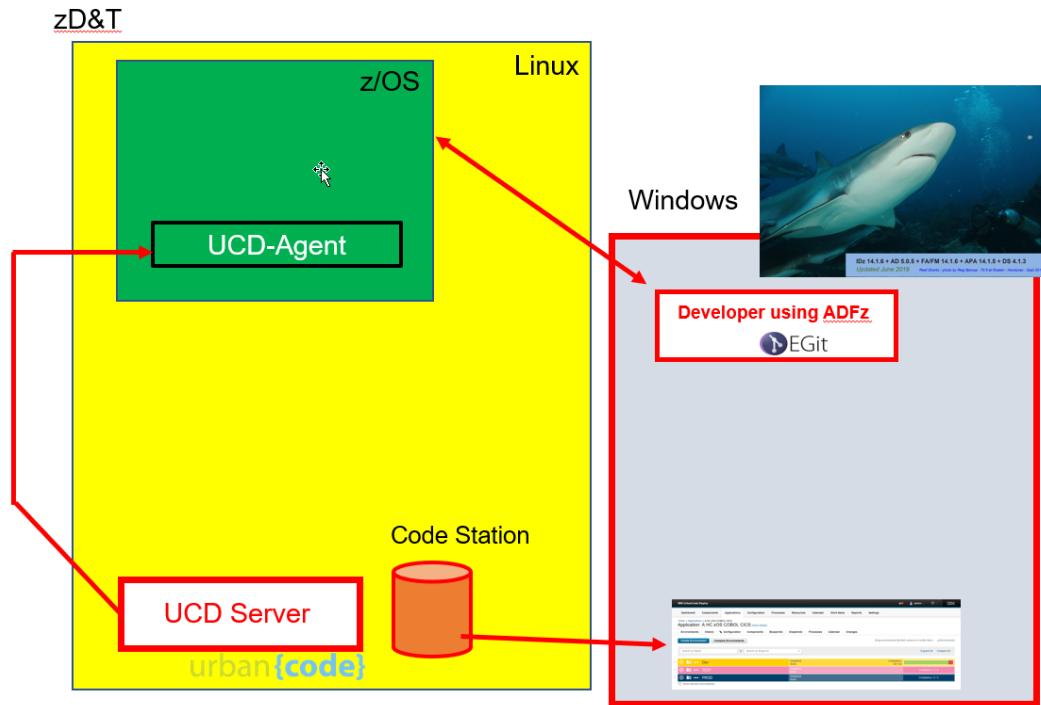
Each time you see the symbol ➡ it means that you have to “do” something on your computer – not merely read the document.

Lab Architecture and proposed scenario

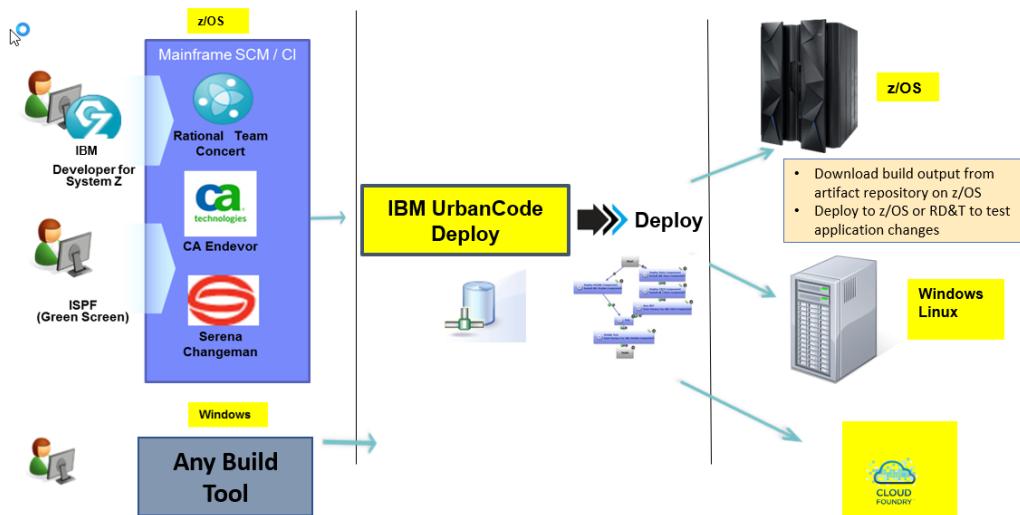
The architecture used in this lab consists of.

1. The **z/OS Server** running on the ZDT that has CICS running and the UCD z/OS agent.
2. The **UrbanCode Deploy Server** that is running on Linux and on the same machine that has the z/OS (ZDT)

Below the architecture of each cloud instance



Below is a picture that describes UCD:

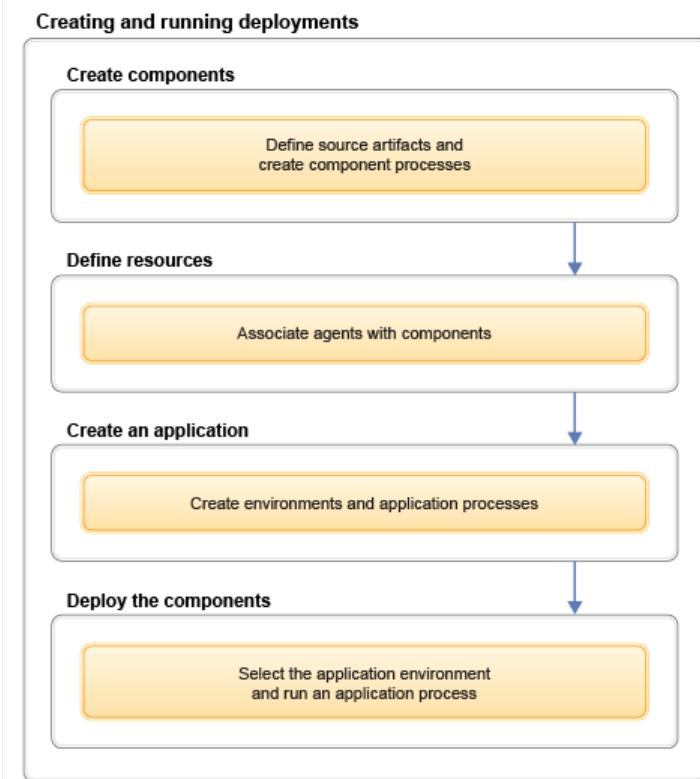


PART 1 – Create the UrbanCode application infrastructure

In this section, you will design the actual deployment process; you need to prepare the application infrastructure in UCD.

First, you will create a component and add component version, then you will define resources and map them to an environment, and finally you will create an application and add environment and component to it

The main steps are as follows:



Task 1 – Logon to UCD server running on Linux and verify that the UrbanCode agents are running

You will now connect to the UrbanCode Deploy (UCD) server running on a LINUX (same Linux that ZDT is running) and that each student have access to it

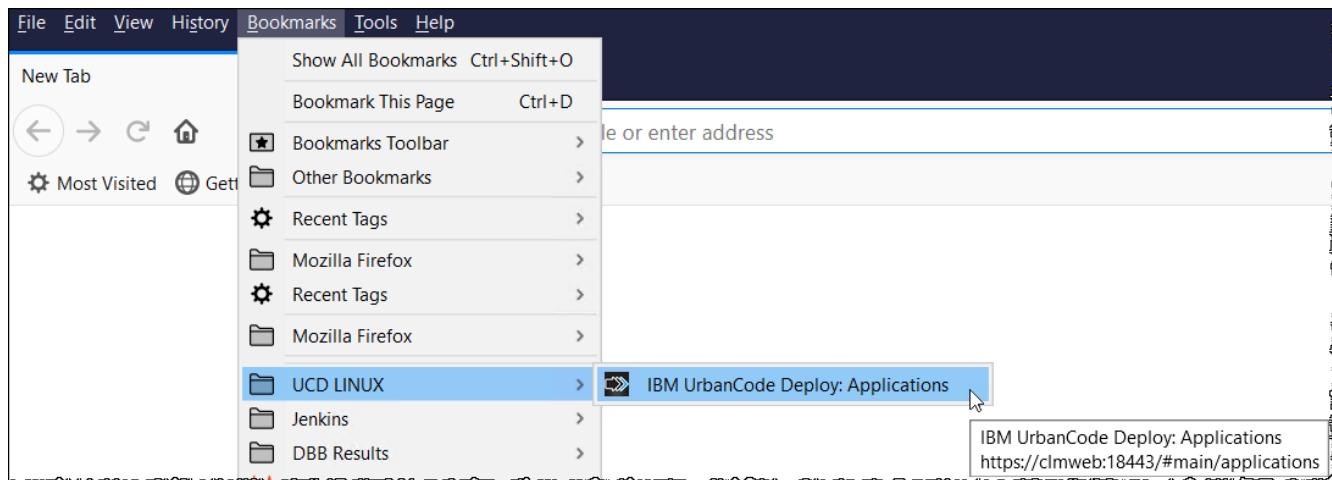
Here you will verify the UCD agents that the UCD server has access to.

You will use the user id **empot02** and password **empot02**. Remember that here the Userid and password is **lower case**.

1.1  **Start the Firefox browser** to go to UCD server on Linux You can use the icon on the base of your screen Move the mouse to the base of your screen if you do not see this bar



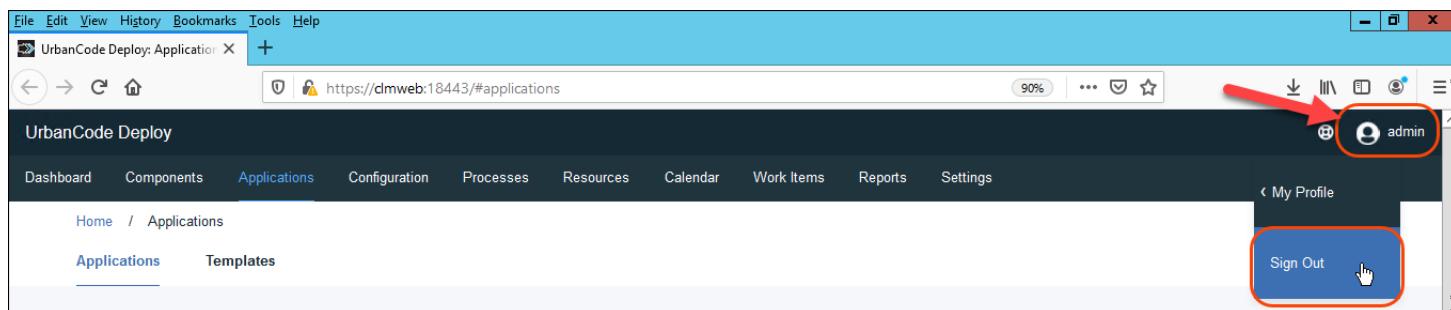
 Look for the **bookmark** below



Or use: <https://clmweb:18443/>

1.2. **If you are already logged on**, you must first logoff and logon with your right user id.

 Click on **admin** (or whatever user id is logged in) and select **Sign Out**



- 1.3. ► Login with **empot02** and password **empot02**
 Remember that this is case sensitive here and lower case

UrbanCode Deploy
v. 7.0.2.0.1011801

Username
empot02

Password

Keep me logged in

Login

© Copyright 2017 IBM Corporation.
© Copyright 2018 HCL Technologies Ltd.

- 1.4. ► Verify that the UCD agents are running Th by clicking **Resources > Agents**.

There are 2 running UCD agents on our z/OS.

- One is version 6 (**zdtagent**) identified by the blue icon ●
- The other is v7 (**zdtagentv7**) identified by the green icon ●

Notice a warning recommending an upgrade of the v6 agent This UCD version 6 agent will point to our DEV environment (CICSTS53).

On this lab you will use only the z/OS V6 agent (**zdtagent**)

UrbanCode Deploy

Dashboard Components Applications Processes **Resources** Calendar Work Items Reports Settings

Home / Resources

Resource Tree Resource Templates **Agents** Agent Configuration Templates Agent Relays Agent Pools Cloud Connections

Flat list Actions... Select All... Discover Available Network Hosts Install New Agent

Name	Description	Status	Date Created	Last Contact	License Type	Licensed	Type	Version	Relay
clmweb		● Offline	Not Available	5/16/2016, 8:03 PM	Unlicensed	false	JMS	6.2.1.0.748085	
linagent		● Offline	3/5/2018, 12:00 PM	5/16/2018, 2:23 PM	Unlicensed	false	JMS	6.2.1.0.748085	
zdtagent		● Offline	1/16/2018, 12:43 AM	4/20/2020, 5:15 PM	RDT	false	JMS	6.2.6.0.932486	
zdtagentv7		● Online	4/21/2020, 12:42 PM	5/6/2020, 3:38 PM	RDT	false	Web	7.0.2.0.1011769	

Items per page: 10 4 records - Refresh Print 1 of 1 pages

The z/OS V6 agent must be running

Upgrade Recommended

Task 2 – Create a UCD component

Components are groups of deployable artifacts that make up an application. You will later create your UCD application named “**J02Mortgage**” that will include only one component that you will create here.

UCD: What is a Component?



A component is a logical representation of deployable artifact along with user-defined processes that operate on them. The physical artifacts are specified in a Component Version. A component has one or more component versions. Each component version has one or more files.

In UrbanCode Deploy, components represent deployable items along with user-defined processes that operate on them. Deployable items, or artifacts, can be files, PDS members, images, databases, etc. In our example, component will include **PDS** members **J02CMORT** and **J02MPMT**.

z/OS: What is a PDS?

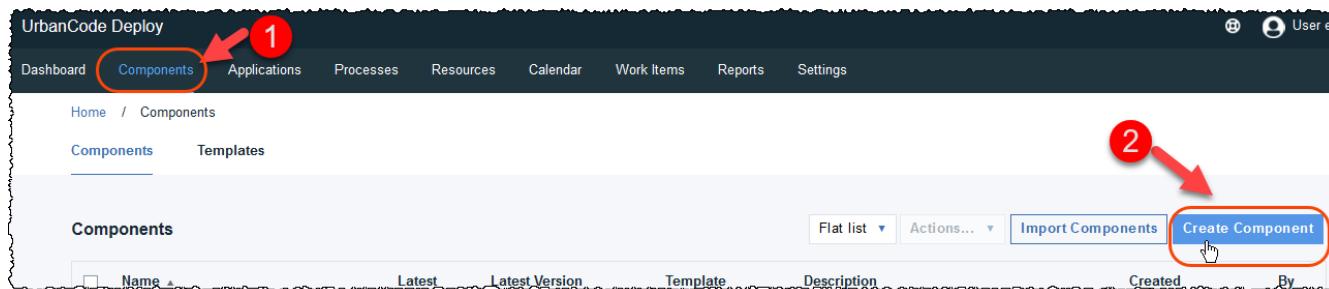


A partitioned data set or PDS consists of a *directory* and *members*. The directory holds the address of each member, or “file”, and thus makes it possible for programs or the operating system to access each member directly. Each member, however, consists of sequentially stored records.

Partitioned data sets are often called *libraries*. Programs, or other content, are stored as members of partitioned data sets. Generally, the operating system loads the members of a PDS into storage sequentially, but it can access members directly when selecting a program for execution.

A PDS also contains a directory. The directory contains an entry for each member in the PDS with a reference (or pointer) to the member. Member names are listed alphabetically in the directory, but members themselves can appear in any order in the library. The directory allows the system to retrieve a particular member in the data set.

2.1 ► On the **Components** page, click **Create Component**:



2.2 Provide values for the required fields on the **Create Component** dialog.

- Specify component Name as **J02Mortgage**
(Be careful since the name is case sensitive).
- Select **MVSCOMPONENT** in the **Template** field (Templates serve as models for various groups of resources. MVSCOMPONENT template, preinstalled with UCD, includes basic z/OS deploy processes and other z/OS related settings)
- Scroll down and click **Save**.

UCD: What is a Component Template?

A component template, stores component processes and properties so you can create components from them; template-based components inherit the template's properties and process.

Create Component ⓘ

Name * 1

Description 3

Teams
Team 02 x

Component Template
MVSCOMPONENT 2

Template Version
Always Use Latest

Component Type
z/OS

High Level Qualifier Length
0

ucd.repository.host

Use the system's default version import agent/tag.

Import new component versions using a single agent.

Import new component versions using any agent with the specified tag.

Cleanup Configuration

Inherit Cleanup Settings

Run Process after a Version is Created

4

2.3 The component **J02Mortgage** is created

Dashboard Components Applications Processes Resources Calendar Work Items Reports Settings

Home / Components / J02Mortgage

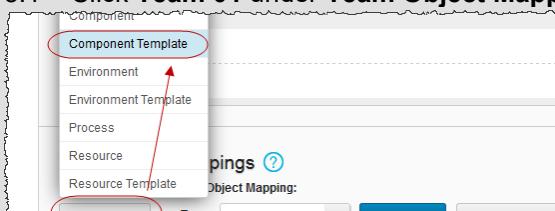
Component: J02Mortgage Show details

Dashboard Usage Configuration Calendar Versions Processes Changes

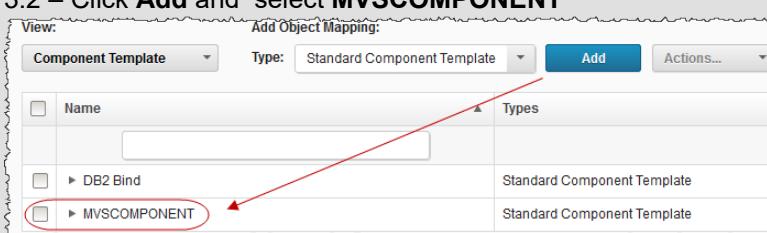
The MVSCOMPONENT is not in the drop down?
 Your UCD server must have this template defined there. If NOT call the instructor..

1. Logon ad **admin/admin**
2. Click **Settings > Teams (under Security)**
3. For EACH team perform as below.

3.1 – Click Team 01 under Team Object Mappings/View: select Component Template



3.2 – Click Add and select MVSCOMPONENT



Task 3 – Create a ship list file and z/OS component version

The z/OS **ship list file** specifies which files from the build to include in the new component version to deploy. You must create a ship list file before you run z/OS deployment tools.

Ship list files are XML files that contain a list of files to be deployed on z/OS. For more information, refer to the UCD documentation:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.doc/topics/zos_shiplistfiles.html

To create a ship list file for the component to be deployed you could use many ways, including ISPF. To make this easy you will use IDz.

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

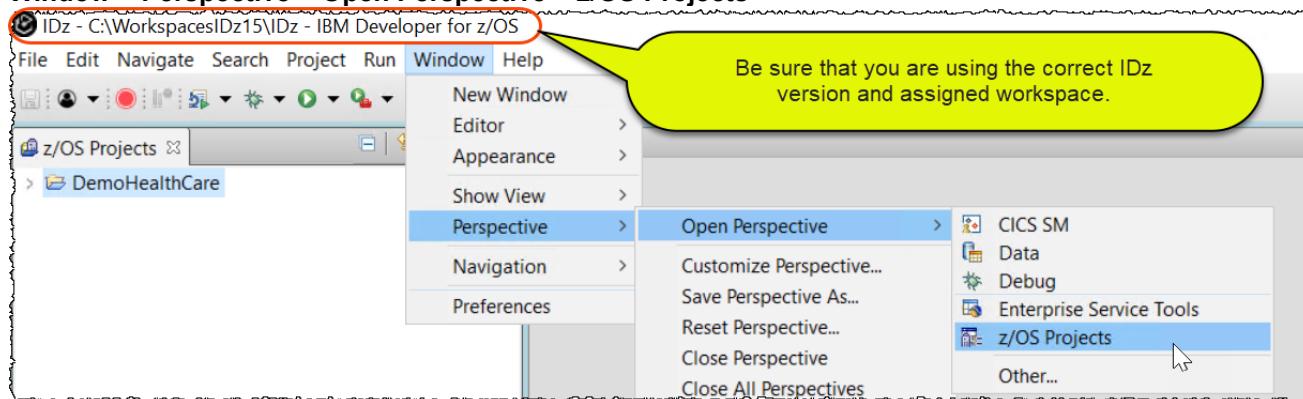
►► Using the desktop double click on **IDz V15** icon.

►► 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.

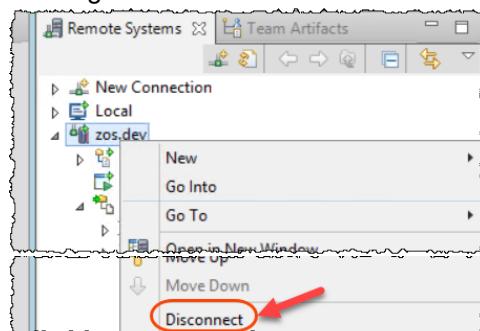


3.2  Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

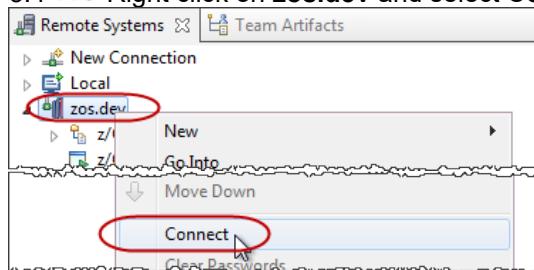


3.3 If you used userid **empot01** before you need to disconnect. Your new userid now will be **empot02**.

 Right click on **zos.dev** and select **Disconnect**

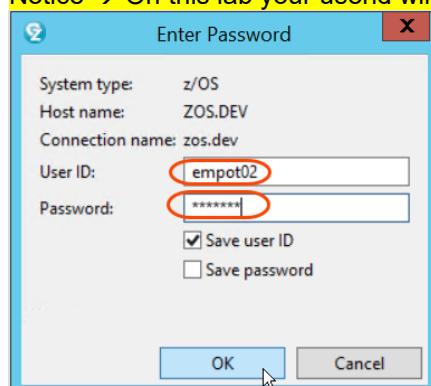


3.4  Right click on **zos.dev** and select **Connect**

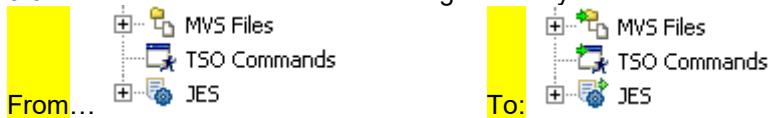


3.5  Type **empot02** as userid (might be already there) and **empot02** as password.
Click **OK** to connect to z/OS.

Notice → On this lab your userid will be **empot02** (not empot01 used in previous labs)



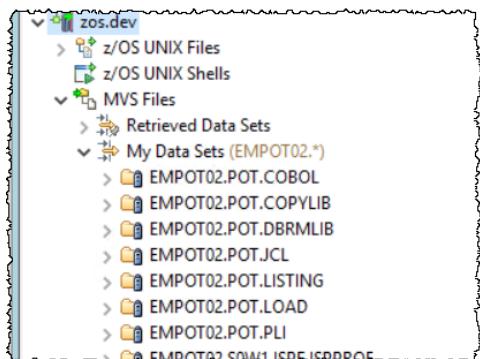
3.6 Notice that the folder icons changed once you are connected to z/OS. A small green arrow is added.,,



3.7 ► Expand MVS Files and My Data Sets to see all your MVS data sets

Note that you do not have the same PDSs shown below (EMPOT02.*) this is just an example..

Depending on which ID you are using you may have no data sets, the IDs are reused and we have no control what is there.



You are connected to a z/OS remote system. Now you will create a Ship List and submit JCL to create the executables to be deployed.

3.8 You now can create the UCD ship list. An XML file that has the list of the load modules that will be deployed. This xml could be created from an SCM build tool like RTC, Endevor etc..

For this lab we will create it manually..

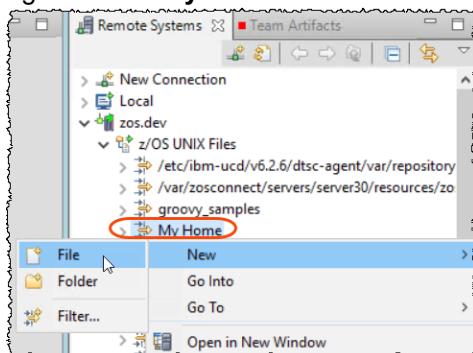
What is the Ship list file?

The ship list file specifies which files from the build to include in the new component version to deploy. You must create a ship list file before you run the IBM® z/OS® deployment tools.

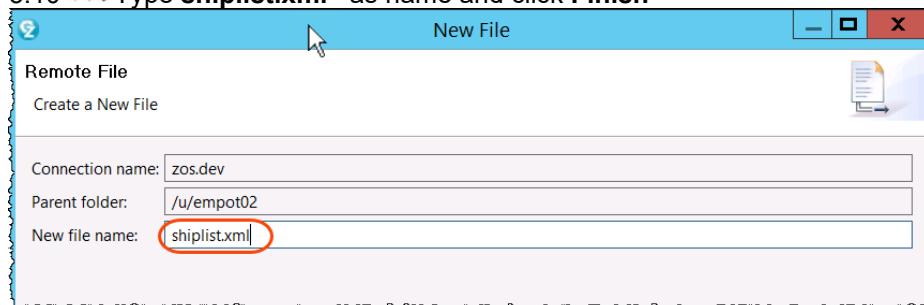


Ship list files are XML files that contain a list of files. The container type that is used to identify partitioned data set (PDS) resources is PDS and the resource type is **PDSmember**. You can use an asterisk (*) as a wildcard for the resource name, if you want all members in a partitioned data set (PDS) to be included in a package. Typically, you write a script that works with your build engine to create a ship list file from the build output.

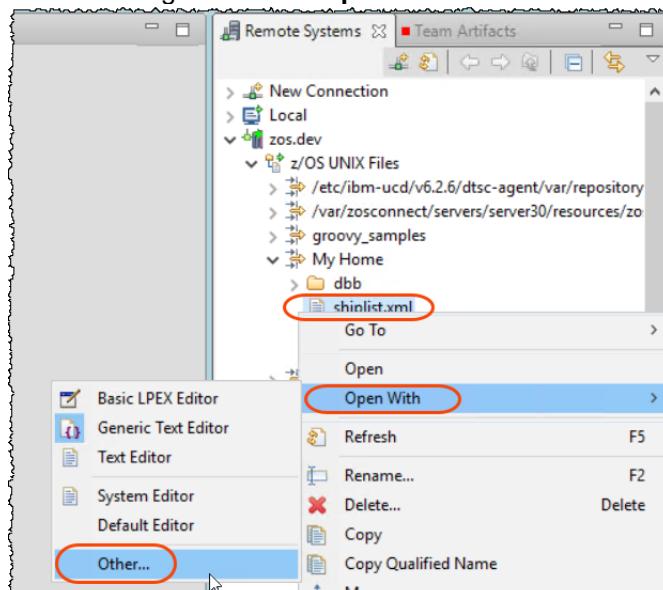
3.9 ► Using the Remote Systems view (on top right), expand z/OS UNIX Files, right click on My Home → New → File



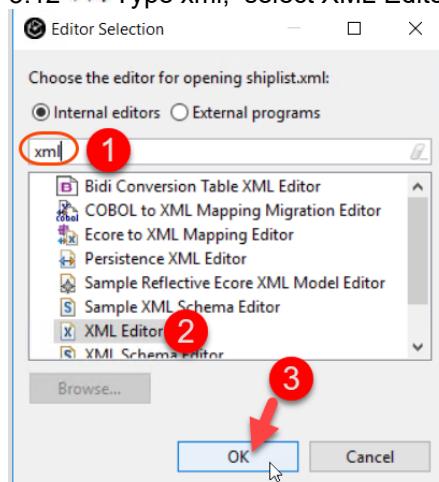
3.10 ► Type **shiplist.xml** as name and click **Finish**



3.11 ► Right click on **shiplist.xml** and select it to edit, or right click and select **Open With Other...**



3.12 ► Type **xml**, select **XML Editor** and click **OK**.



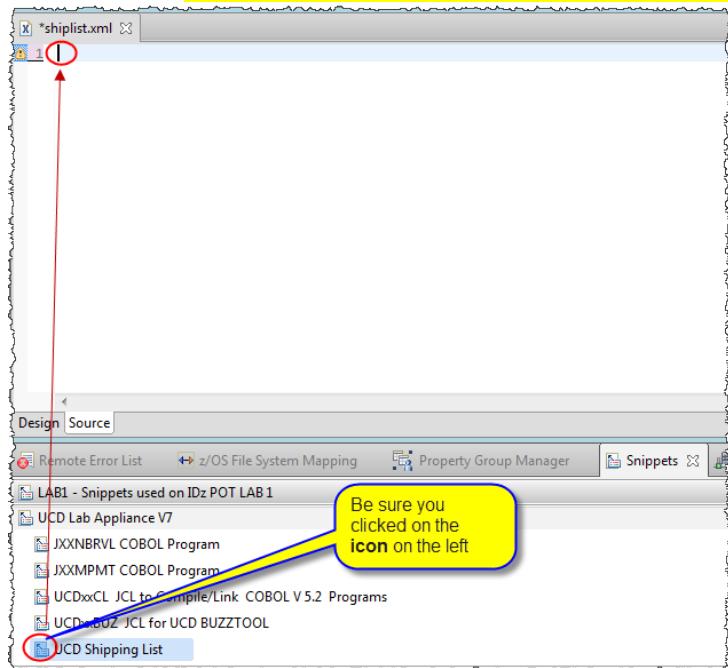
3.13 ► Click on **Source** tab



3.14 ► Click on **Snippets** tab on the bottom.

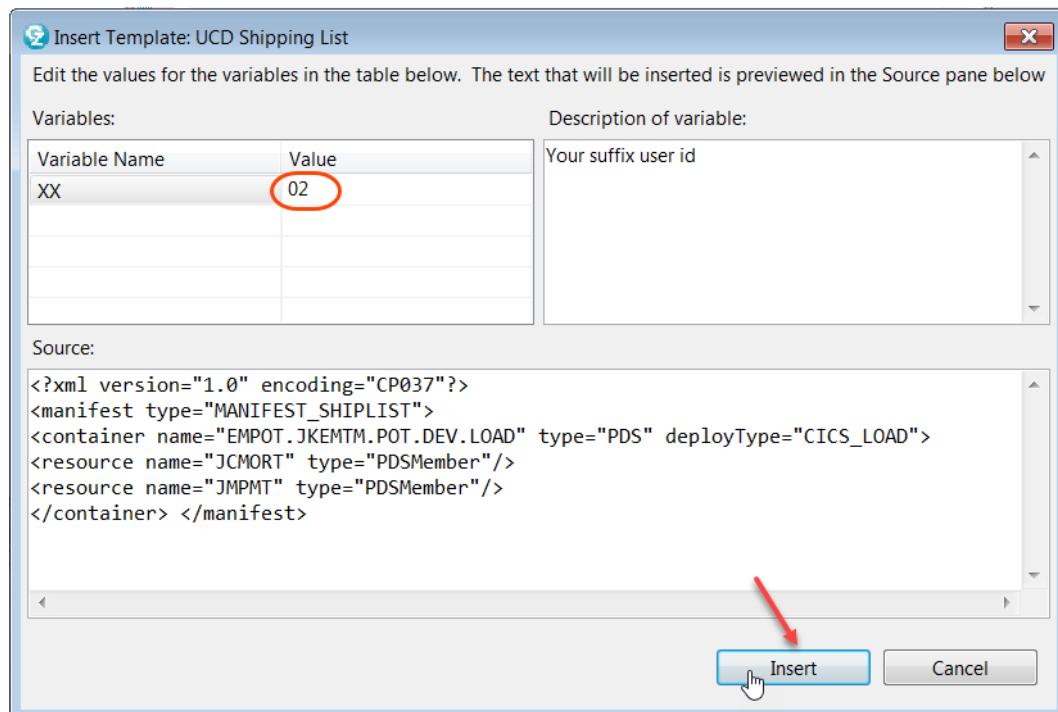
► Scroll down and use the **UCD Shipping List** snippets to drag and drop to the **FIRST position** of the file

IMPORTANT: You must click on the icon  to be able to drag/drop

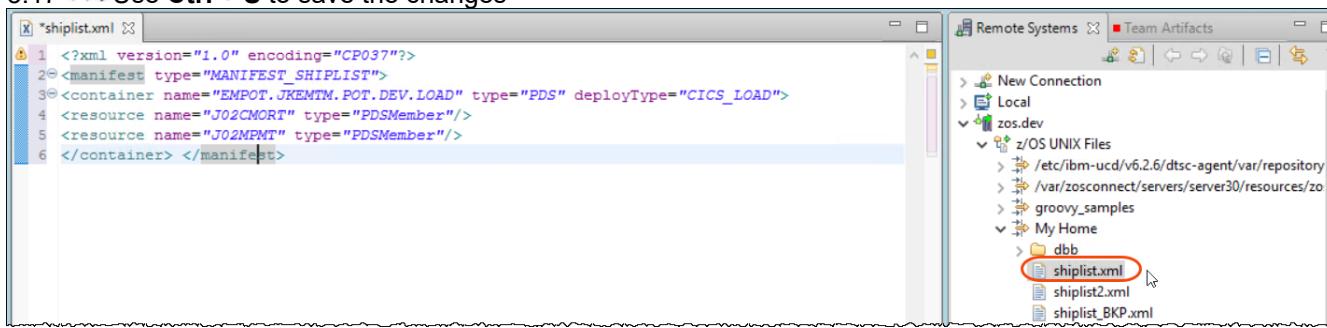


3.15 ► When the dialog opens type **02** ..

3.16 ► Click **Insert** to insert the lines. Notice that the variables are replaced



3.17 ►| Use Ctrl + S to save the changes



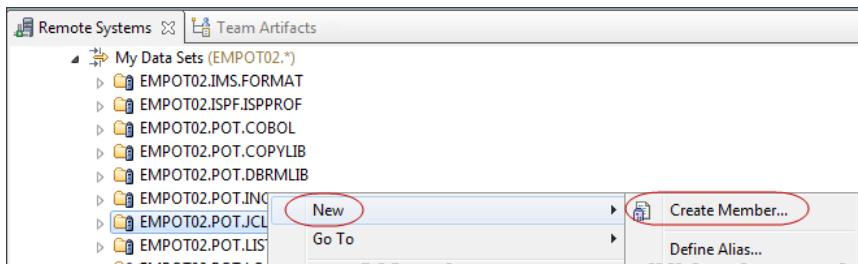
Task 4 - Create the UCD component version from JCL

On this task, you will now use the z/OS UCD tool named “BUZTOOL” to create a version of the modules that need to be deployed. The UCD Toolkit installed on z/OS used the UCD Client to communicate to the UCD server.

In this lab the UCD code station on z/OS is kept in the USS Files or on UCD server (installation decision).

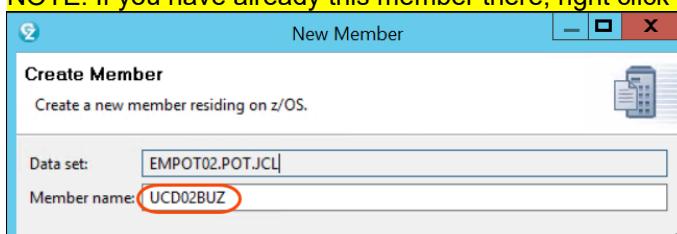
4.1 ►| Using Remote System view, expand **My Data Sets**, right click on **EMPOT02.POT.JCL** and create a member .

If you don't have this PDS call the instructor.

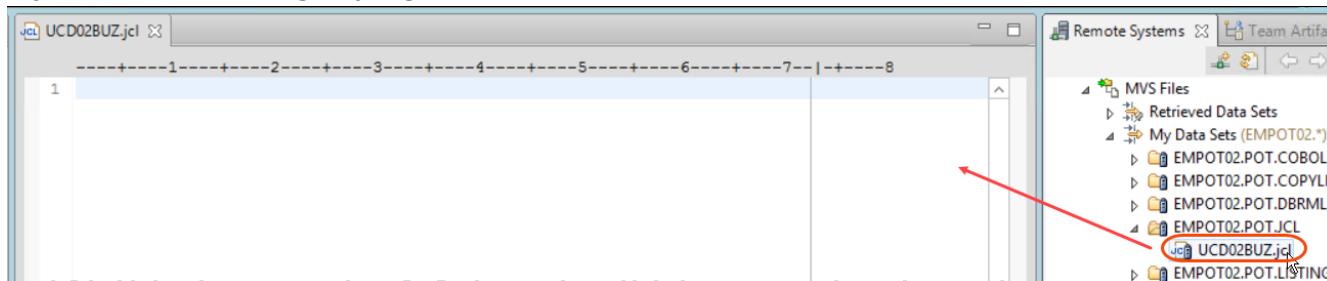


4.2 ►| Name it **UCD02BUZ** and click **Finish**

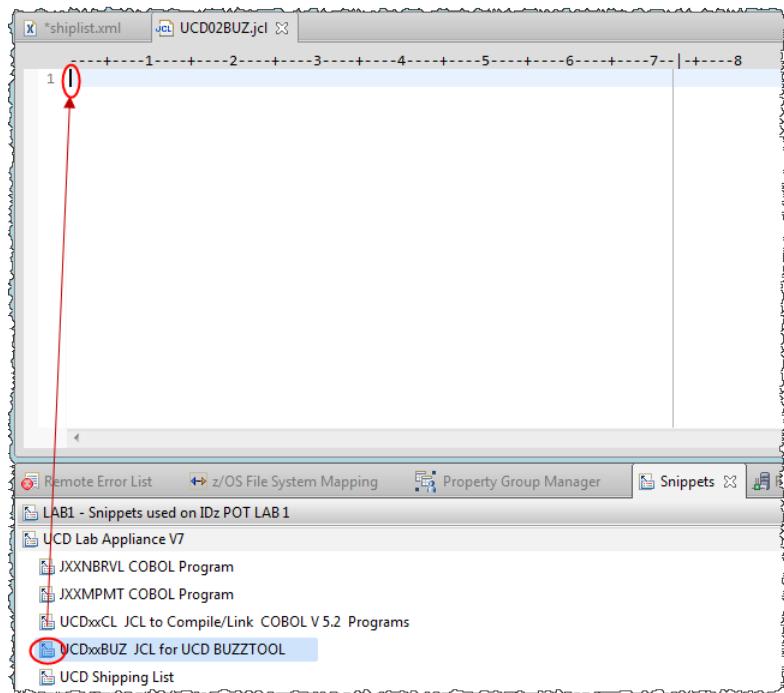
NOTE: If you have already this member there, right click and delete it.



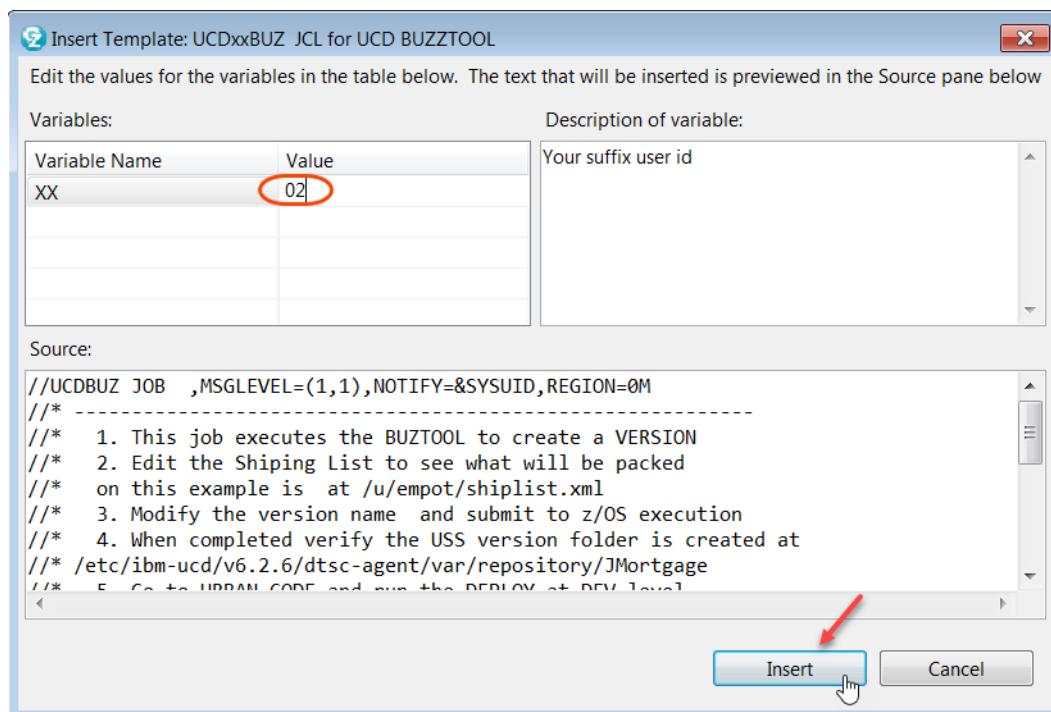
4.3 ►| Double click on **UCD02BUZ** to edit.



- 4.4 ►| Use the UCDxxBUZ snippets to drag and drop to the **FIRST POSITION** of the file
IMPORTANT: You must click on the icon  to be able to drag/drop



- 4.5 ►| When the dialog opens type **02**. ..
 ►| Click **Insert** to insert the lines. Note that the variables are replaced



This job uses the **buztool createzosversion** command to create the component version.

4.6 ► Scroll down and understand the parameters passed to the BUZTOOL

```

1 //UCD02BUZ JOB ,MSGLEVEL=(1,1),NOTIFY=&SYSUID,REGION=0M
2 /*
3 /* 1. This job executes the BUZTOOL to create a VERSION
4 /* 2. Edit the Shiping List to see what will be packed
5 /* on this example is at /u/empot02/shiplist.xml
6 /* 3. Modify the version name and submit to z/OS execution
7 /* 4. When completed verify the USS version folder is created
8 /* /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage
9 /* 5. Go to URBAN CODE and run the DEPLOY at DEV level
10 /*
11 //BUZTOOL EXEC PGM=BPXBATCH,REGION=0M
12 //STDOUT DD SYSOUT=*
13 //STDERR DD SYSOUT=*
14 //STDPARM DD *
15 SH /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh createzosversion
16     "-c" "J02Mortgage"
17     "-s" "/u/empot02/shiplist.xml"
18     "-v" "yyyymmdd"
19 /*
20 /* yyyyymmdd will be the name of your version
21 */

```

Possible arguments for the buztool **createzosversion** command

Parameter	Required	Description
-c	true	The name of the component in IBM UrbanCode Deploy. The component name can contain only letters, numbers, and spaces.
-v	false	The name of the version to create. If a version is not specified, a version name is generated from the current time stamp. The version name can contain only letters, numbers, and spaces.
-s	true	The location of the ship list file.
-verb	false	To display a trace log, set this parameter to true.

4.7 ► Name a version for this component:

Modify from "yyyymmdd" to today's date in the format of **year, month and day**

Example: I used **20180709**. This version name must be unique for this component to avoid errors of being already created on the code station.

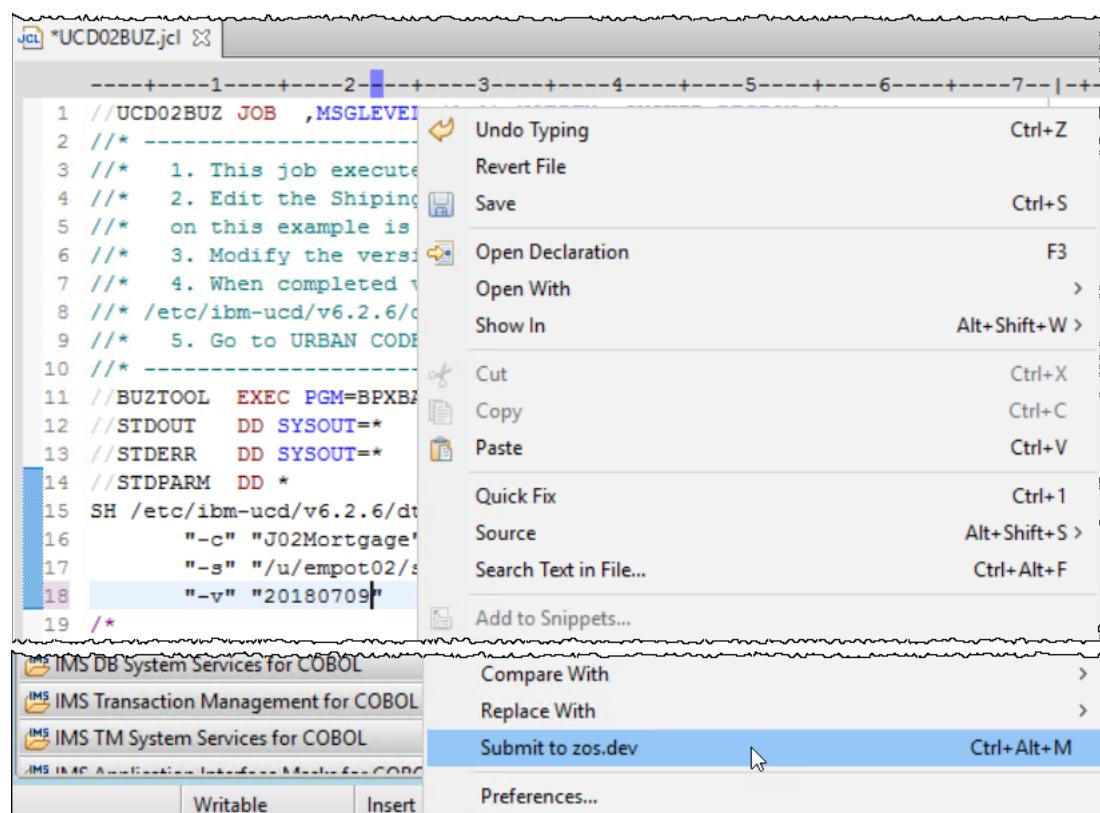
```

14 //STDPARM DD *
15 SH /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh createzosversion
16     "-c" "J02Mortgage"
17     "-s" "/u/empot02/shiplist.xml"
18     "-v" "20180709"

```

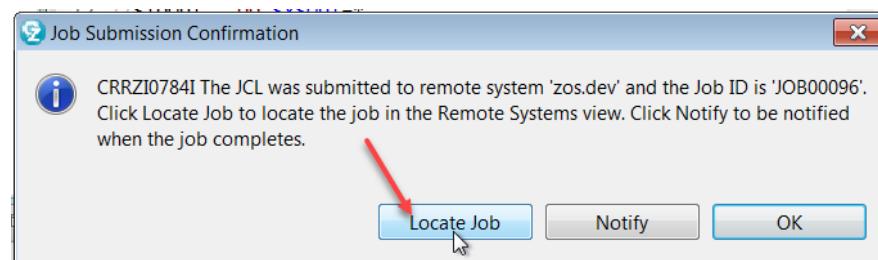
4.8 ►| Use **Ctrl + S** to save

►| Right click on the editor and select **Submit to zos.dev**



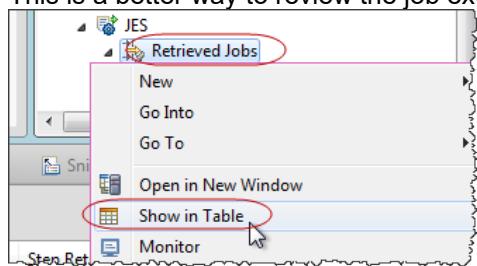
You do not find the option Submit?...

Be sure you are using the correct editor. Either LPEX or JCL editors will provide this capability when right clicking. Be sure you pasted the content on line 1 and column 1. If you still don't see, you can right click on the editor content and select **Open with → Other -> JCL Editor** and try again..

4.9 ►| Click **Locate Job** to get the job results

This job may take as long as 1 or 2 minutes.. Remember that your z/OS instance on cloud has limited power resources and is very slow.

4.10 ► On the right under *Remote System* view right click on **Retrieve Jobs** and select **Show in Table**. This is a better way to review the job execution



4.11 ► Use the refresh icon to see the job being executed. Once execution is completed you must have a *User Return Code*

Resource	Job ID	Job Name	Job Own...	Job Entr...	Return C...	Return In...	System ...	User Ret...	Return S...	Queue P...
UCD02BUZ:JOB0...	JOB00096	UCD02B...	EMPOT02	2018/07...	CC 0000	NORMAL		000	COMPLE...	10

The return code **must be 0** and the version must be created on UCD Server

4.12 ► Double click and scroll down to see the results

```

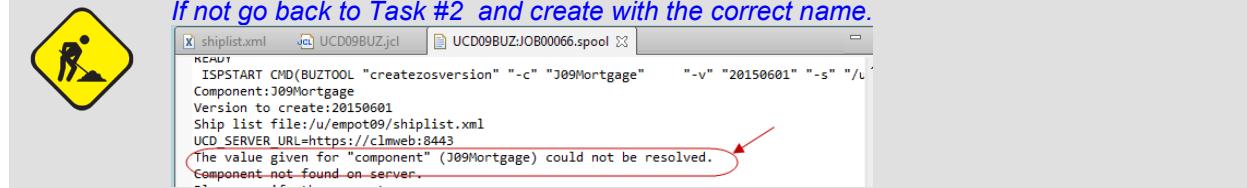
58      "-s" "/u/empot02/shiplist.xml"
59      "-v" "20180709"
60 zOS toolkit config  : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,2017
61 zOS toolkit binary  : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,2017
62 zOS toolkit data set : BUZ626 (6.2.6,20170907-0249)
63 Reading parameters:
64 ....Command : createzosversion
65 ....Component : J02Mortgage
66 ....Version : 20180709
67 ....Shiplist file : /u/empot02/shiplist.xml
68 Verifying version
69 ....Repository location : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repos
70 Pre-processing shiplist:
71 ....Shiplist after processing : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/
72 Packaging data sets:
73 ....Location to store zip : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/rep
74 ....Zip name : package.zip
75 ....EMPTO.JKEMTM.POT.DEV.LOAD.bin
76 ....Elapsed time for data set package or deploy operation : 1.6954
77 Create version and store package:
78 ....Version artifacts stored to UCD server CodeStation
79 ....Version:20180709 created
80 Elapsed time 24.0 seconds.
81 FSUM1006 A shell was not specified. Processing continues using the
82

```

4.13 ► Use **Ctrl + Shift + F4** to close all editors.

Errors like the one below?

Be sure that you have created the component with the correct name (mixed case letters). If not go back to Task #2 and create with the correct name.

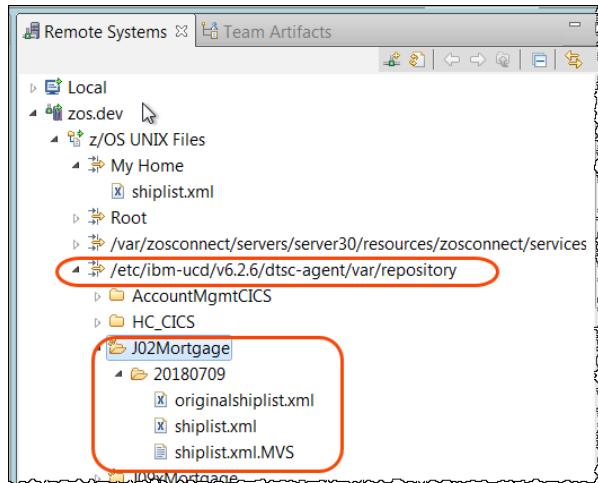


4.14 (Optional)

► Under **z/OS UNIX Files** navigate to the filter
/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository

Expand **J02Mortgage** to see the UCD metadata.

Notice that the zip file is located on the UCD CODE STATION on Linux (not on z/OS).



4.15 ► Close or minimize IDz, you will not need any more on this lab.

Notice that during version creating process, the **BUZTOOL** communicates with your UCD server to obtain component information and to store version artifacts. If component version was created successfully, you can verify version information using UCD server interface.

4.16 ► Now Using the UCD browser, click Components -> J02Mortgage

The screenshot shows the 'Components' tab in the IBM UrbanCode Deploy interface. Below it, there are five status indicators: Failed Version Import (0), Importing Version (0), No Version (1), Successful (0), and No Artifact (0). At the bottom, there are buttons for 'Create Component', 'Import Components', 'Actions...', and 'Flat list'. A table lists components, with the 'J02Mortgage' component highlighted in blue and circled in red with a red number 2. The table columns are: Name, Latest Import, Latest Version, Template, Description, and Created. The 'J02Mortgage' row shows: J02Mortgage, 20180709, 20180709, MVSCOMPONENT, HC Mobile app for Windows7, and 7/9/2018, 6:1 PM.

Name	Latest Import	Latest Version	Template	Description	Created
Component Name					
AccountMgmtCICS		work_item_287.20180608-0223530451	MVSCOMPONENT	Component used on PDTOLS demo	6/24/2015, 7 PM
HC_CICS		20180627-031937	MVSCOMPONENT	zMobile Health Care Appl - COBOL CICS assets	12/14/2017, 4:51 PM
HCMobileWindows7		V02 - HC for demo deploy		HC Mobile app for Windows7	12/14/2017, 4:51 PM
J02Mortgage	20180709	20180709	MVSCOMPONENT		7/9/2018, 6:1 PM

4.17 ► Click , ① Versions and the ② version that you created (yyyyymmdd).

Version	Statuses	Type	Created By	Date
20180709	Statuses	Any	admin	7/9/2018, 9:52 PM
testeRegi		Incremental	admin	7/9/2018, 9:28 PM

4.18 ► Click **Show Details**. You can verify that the repository is on UCD Server Code Station (on Linux, not z/OS USS Files)

UrbanCode Deploy

Components

Version: 20180709

Created By: admin
Created On: 5/2/2020, 4:05 PM
Repository Type: CodeStation
Links

4.19 ► Expand **EMPOT.JKEMT.POT.DEV.LOAD** and you should be able to see the list load modules that are packaged and stored in the Code Station (from the JCL that you submitted before). Optionally the Code Station could be on the z/OS under USS folders.

In our example the Code Station is on the UCD server (LINUX):

Name	Artifact Type	Deploy Type	Inputs
EMPOT.JKEMT.POT.DEV.LOAD	[PDS,ADD]	CICS_LOAD	
J02CMORT		CICS_LOAD	
J02MPMT		CICS_LOAD	

Task 5 - Create UCD Resources

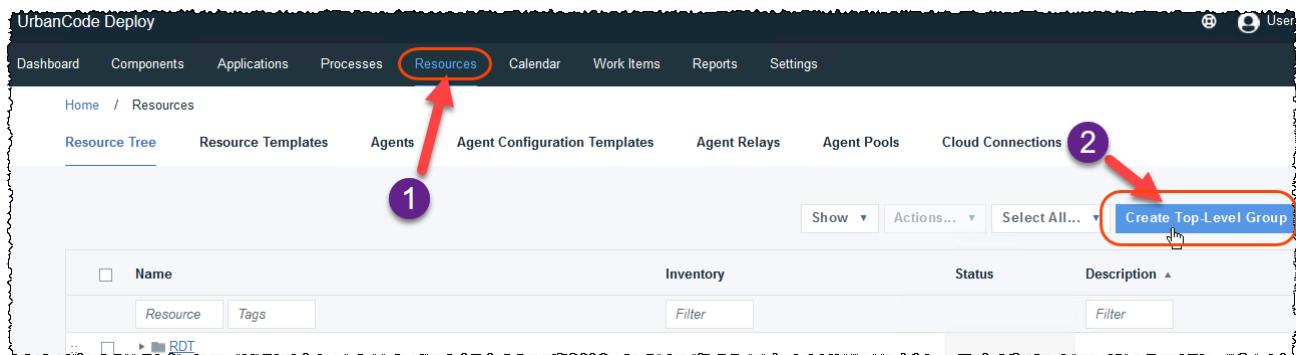
On this task you will create the UCD resources that is a user-defined construct.



UCD : What is a resource?

A resource is a logical deployment target that typically resolves to an agent and a user-defined construct that is based on the architectural model of UCD. Resources can contain other resources in a hierarchical tree structure (called also resource groups or folders) to represent complex targets. Resources are assigned to environments.

5.1 ► On the **Resources** tab, click **Create Top-Level Group**:



5.2 ► Type **zOS Resource Group 02** and click **Save**.

Create Resource

Name*
zOS Resource Group 02

Include Agents Automatically

Description

Teams
Team 02 x

Default Impersonation

Default impersonation can be configured here. Any steps which do not specify their own impersonation settings will fall back to the settings provided here.

Cancel Save

You should have:

Name	Inventory	Status	Description
Resource Tags Filter	Filter	Filter	
.. RDT			
.. zOS Resource Group 02			

5.3 ► Move the mouse to **ZOS Resource Group**, click the **...** (three dots) and using the drop-down menu click **Add Agent**:



5.4 ► In the **Agent** drop down dialog, select **zdtagent** and click **Save**

Create Resource

Agent: zdtagent (circled in red)

Name*: zdtagent

Description: (empty)

Inherit Teams From Parent

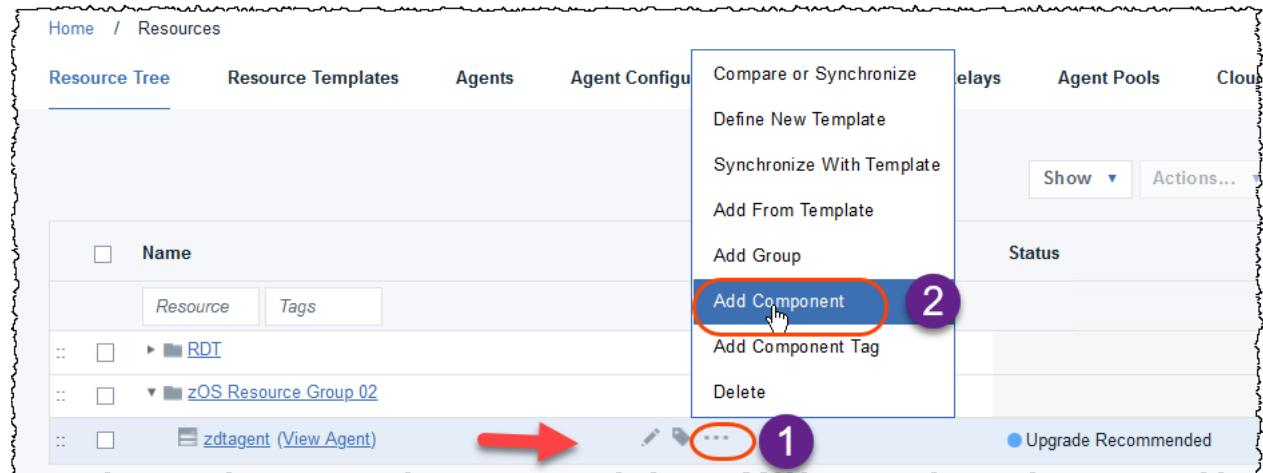
Teams: Team 02

Default Impersonation

Default impersonation can be configured here. Any steps which do not specify their own impersonation settings will fall back to the settings provided here.

Save (highlighted by a red arrow)

- 5.5 ► In the row **zdtagent**, click the ... (three dots) and using the drop-down menu click **Add Component**:



- 5.6 ► Select the component created earlier (**J02Mortgage**) and click **Save**.

Create Resource

Component*
J02Mortgage

Name*
J02Mortgage

Description
[empty]

Inherit Teams From Parent

Teams
Team 02

Default Impersonation

Default impersonation can be configured here. Any steps which do not specify their own impersonation settings will fall back to the settings provided here.

Role Properties: J02Mortgage

Property	Description	Value	Actions
No properties are available on this role.			

Refresh Print

Cancel **Save**

You should have:

The screenshot shows the 'Resource Tree' section of the UrbanCode Deploy interface. At the top, there are tabs for 'Resource Tree', 'Resource Templates', 'Agents', 'Agent Configuration Templates', 'Agent Relays', 'Agent Pools', and 'Cloud Groups'. Below the tabs is a search bar with 'Show' and 'Actions...' buttons. The main area displays a tree view of resources under 'zOS Resource Group 02', with 'zdagent (View Agent)' and 'J02Mortgage (View Component)' highlighted by a red box.

Task 6 - Create an UrbanCode Application

On this task you will create an UCD application.

Applications are responsible for bringing together all the components that must be deployed together.

UCD : What is a UCD Application?

Application is a module the contains the following elements:

- Components to be deployed
- Environments target and the resources to deploy the application.
- A process definition (note that application process runs on the server while component process runs on agent)
- Properties

UCD: Properties

Properties can be set in UCD for many different things, including components, environments, processes, and applications. You can also set global properties for the system.

You can refer to a property by scope:
 \${p:scope/propertyName} for example: \${p:environment/DBconnect}

or without scope:
 \${p:propertyName}

You define objects (Application, component, process, resource, etc) properties in the object's Configuration Tab of UCD browser interface.

For more details, refer to the full documentation on Properties:
http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.reference.doc/topics/ud_properties.html

6.1 ► On the Applications tab, click Create Application:

The screenshot shows the 'Applications' tab selected in the navigation bar. Below the navigation bar, there are two buttons: 'Applications' and 'Templates'. At the bottom right of the page, there is a blue 'Create Application' button with a red arrow pointing to it.

6.2 ► Provide the value for the **Name** field in the **Create Application** dialog
Name it **J02Mortgage COBOL** and click **Save**

The screenshot shows the 'Create Application' dialog box. The 'Name' field is highlighted with a red oval and contains the text 'J02Mortgage COBOL'. In the 'Teams' section, 'Team 02' is listed with a delete icon next to it. At the bottom of the dialog, there is a checkbox labeled 'Enforce Complete Snapshots' and two buttons: 'Cancel' and 'Save', with a red arrow pointing to the 'Save' button.

The result should be:

The screenshot shows the application details page for 'J02Mortgage COBOL'. The title 'Application: J02Mortgage COBOL' is highlighted with a red oval. The page has tabs for Environments, History, Configuration, Components, Blueprints, Snapshots, Processes, Calendar, and Changes. The Environments tab is active. On the right side, there is a 'Create Environment' button. Below the tabs, there is a search bar with 'Search by Name' and 'Search by Blueprint' options, and a checkbox for 'Show Inactive Environments'. A message at the bottom states 'No Environments Found'.

Task 7 - Create environment

On this task you will create The UCD environment that is a user-defined collection of resources that hosts an UCD application.

When you create an environment, you map resources to it that define where the parent application can run deployments.

UCD : What is a UCD Application Environment?

An environment is the application's mechanism for bringing together components with the agent that deploys them. Environments are typically modeled on some stage of the software project lifecycle, such as development, test, or production.

When you create an environment, you map resources to it and the application components that will be deployed.

You can also define environment specific properties.



7.1 ➔ Click Create Environment:

Home / Applications / J02Mortgage COBOL

Application: J02Mortgage COBOL Show details

Environments History Configuration Components Blueprints Snapshots Processes Calendar Changes

Drag environments by their names to re-order them. o Environments

Create Environment

7.2 ➔ Provide the value for the Name field in the Create Environment dialog (for example, **Test**), choose a color, and click **Save**:

Create Environment

Name* 1

Description

Blueprint

Teams

Team 02 x

+ Require Approvals

Exempt Processes

None

Lock Snapshots

Require Snapshot

Color

Cancel Save 3

7.3 ► Click the environment name (**Test** in our example) to open environment properties:

The screenshot shows the 'Environments' tab selected in the top navigation bar. Below it, a message says 'Drag environments by their names to re-order them.' followed by '1 Environment'. There are search fields for 'Search by Name' and 'Search by Blueprint'. A checkbox for 'Show Inactive Environments' is unchecked. The environment 'Test' is listed with a yellow background, indicating it is selected. To the right of 'Test' is a 'Snapshot' status showing 'None'. A blue button labeled 'Create Environment' is at the top right.

7.4 ► Click Add Base Resources:

The screenshot shows the 'Resources' tab selected in the top navigation bar. It displays a table with columns: Name, Inventory, Status, and Description. A message at the top says 'No Desired Inventory'. At the top right, there are buttons for 'Show', 'Actions...', 'Select All...', and a blue button labeled 'Add Base Resources' with a red arrow pointing to it. The table below shows a single row: 'No resources have been added yet.'

7.5 ► Expand zOS Resource Group 02, select Component **J02Mortgage**: and click **Save**:

Note: In order for a component to be deployed by an application, it must be added to the application and also mapped to an agent-type resource. A component that is added to an application but not mapped to an agent resource, cannot be deployed by that application.

Similarly, a component that is mapped to an agent resource but not added to an application, cannot be deployed by that application.

The screenshot shows the 'Add Resource to Environment' dialog. It has a table with columns: Name, Inventory, and Description. The 'Name' column is sorted by 'Name'. There are filter buttons for 'Name' and 'Tags' in the first row. Below is a tree view of resources. The 'zOS Resource Group 02' node is expanded, and its child 'zdtagent' node is also expanded. Under 'zdtagent', the 'J02Mortgage' component is selected and has a checked checkbox next to it. A red box highlights the 'zOS Resource Group 02' node. A purple circle labeled '1' points to the checked checkbox next to 'J02Mortgage'. A purple circle labeled '2' points to the 'Save' button at the bottom right. Other buttons in the bottom right include 'Cancel', 'Print', 'Refresh', and 'Help'.

You now must have:

Name	Inventory	Status	Description
zOS Resource Group 02 / zdagent (View Agent) / J02Mortgage			

PART 2 - Create the UrbanCode deployment processes.

On this part you will you create a deployment process for your component and the application process that uses the component process to deploy the component.

Processes are automated tasks that run on agents.

There are three types of processes:

- **Generic processes** run outside the context of components or applications. Not used on this lab.
- **Application processes** run within the context of applications. In many cases, application processes call component processes. For example, an application process can call the component processes that deploy those components.
- **Component processes** run tasks on a single component, such as deploying it, uninstalling it, or running configuration tasks on it.



Task 8 - Create a components process

A component process is a succession of commands that are called steps. Steps can manipulate files, run system commands, set properties, pass information to other steps, and run programs. Steps are provided by automation plug-ins. Processes are designed with the drag-and-drop process editor where you drag plug-in steps onto the design editor and configure them as you go. Several plug-ins come with the product and others are available, which work with many different types of software. In this lab you use z/OS plug-ins. A component can have any number of processes defined for it, but a component must have at least one process.

In this task you create a deployment process for your component. Later, you create an application process that uses the component process to deploy the component.

We are going to create a **component process** for this example. This process will automate deployment of the new version of our **J02Mortgage COBOL** application.

8.1 ► Click on tab **Components**, find the component created earlier (**J02Mortgage**) and click on it

The screenshot shows the 'Components' tab selected in the top navigation bar. Below the navigation, there's a breadcrumb trail: Home / Components. Under the 'Components' heading, there's a table listing several components. One component, 'J02Mortgage', is highlighted with a red oval and a purple circle labeled '2'. Another component, 'Windows', is also highlighted with a red oval and a purple circle labeled '1'.

Name	Latest Import	Latest Version	Template	Description	Created	By
AccountMgmtCICS	20180627-1257160725	MVSCOMPONENT	Component used on PDTOOLS demo	6/24/2015, 7:55 PM	admin	
HC_CICS	20200130-124347	MVSCOMPONENT	zMobile Health Care Appl - COBOL CICS assets	12/14/2017, 4:51 PM	admin	
HCMobileWindows7	V02 - HC for demo deploy		HC Mobile appl for Windows7	12/14/2017, 4:51 PM	admin	
J02Mortgage	20180709	MVSCOMPONENT		5/6/2020, 5:04 PM	User empt0 (empt02)	

8.2 ► Click the **Processes** tab

The screenshot shows the 'Processes' tab selected in the top navigation bar of the 'Component: J02Mortgage' details page. Below the navigation, there's a breadcrumb trail: Home / Components / J02Mortgage. The main content area lists various processes:

Process	Description
Deploy (MVSCOMPONENT)	Deploy data sets. Version artifacts are fetched from code station located in the same z/OS.
Deploy - get artifacts from CodeStation (MVSCOMPONENT)	Deploy data sets. Version artifacts are fetched from UrbanCode Deploy server CodeStation.
Deploy - get artifacts using FTP (MVSCOMPONENT)	Deploy data sets. Version artifacts are fetched from code station using FTP
Remove all versions (MVSCOMPONENT)	Remove all versions in an environment including the backup created during version deployment. Use this process when you want to start next round of development with a clean environment. Audit history is available even if versions have been removed from the environment.
Remove redundant versions (MVSCOMPONENT)	Remove redundant versions in an environment. Redundant versions are these versions which are replaced completely by later deployed versions.
Remove redundant versions with manual verification (MVSCOMPONENT)	Remove redundant versions in an environment. Redundant versions are these versions which are replaced completely by later deployed versions.
Sample JCL submission process (MVSCOMPONENT)	This process demonstrates usage of the JCL submission steps.
Uninstall (MVSCOMPONENT)	Uninstall a version and restore the backup data sets

You will notice that there is a small collection of processes that was created automatically for your component (**Deploy**, **Remove all versions**, **Uninstall**, and so on). Remember that those processes were created from a template.

These processes can be used by themselves to perform simple tasks or can be used as starting points for more complex processes.

We will next create our deployment process. We could slightly simplify this task by reusing the existing process called **Deploy**. However, for the purposes of this example, we are going to create a brand new one.

8.3 ► Click **Create Process** button to get started.



8.4 ► Enter Process Name (**Deploy JKE to CICS**)

Leave all other options at their default values and click **Save**:

The screenshot shows a 'Create Process' dialog box. It has several input fields: 'Name*' (with 'Deploy JKE to CICS' entered), 'Description' (empty), 'Process Type*' (set to 'Deployment'), 'Inventory Status*' (set to 'Active'), 'Default Working Directory*' (containing the placeholder `\${p:resource/work.dir}/\${p:component.name}`), and 'Required Role' (empty). At the bottom, there are 'Cancel' and 'Save' buttons. A red arrow points to the 'Save' button, which is circled with a purple marker labeled '2'. Another purple marker labeled '1' points to the 'Name' field.

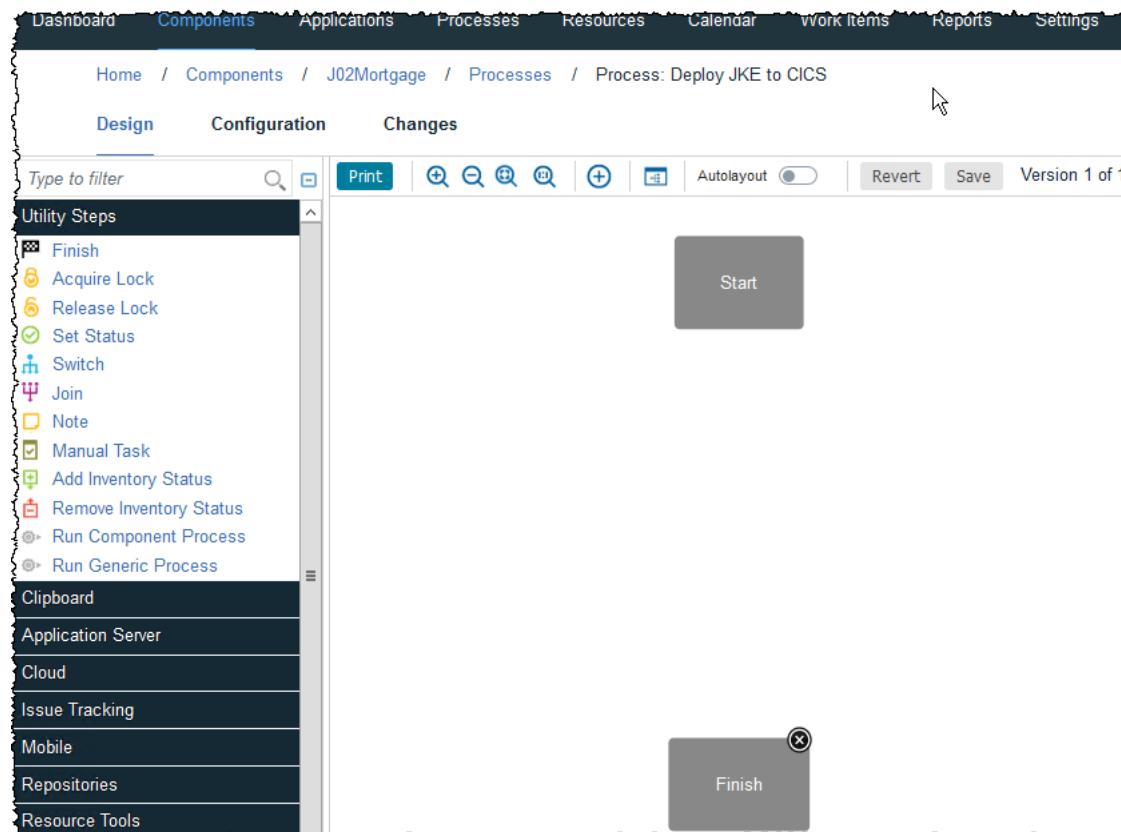
8.5 The Process Editor will open.

Here, you can organize the steps of a process, specify their properties, and connect them to each other.

As usual, you can refer to UCD documentation for detailed information on editing processes.

If you have internet access you can go to the link below for more details::

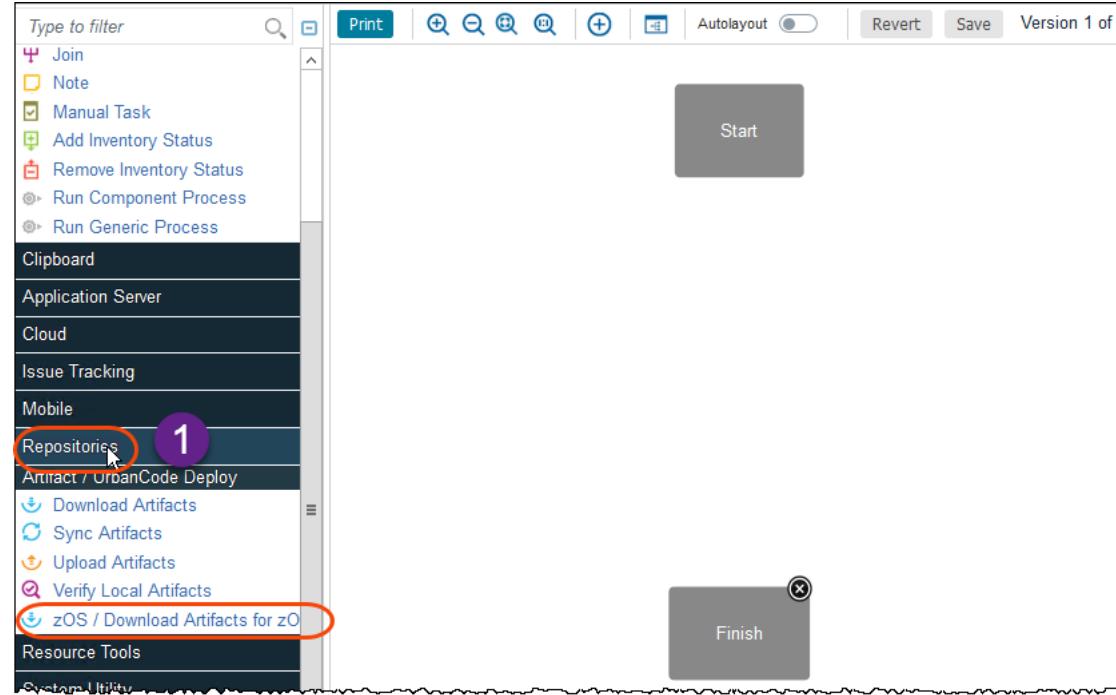
https://www.ibm.com/support/knowledgecenter/SS4GSP_7.0.2/com.ibm.udesigner.doc/topics/comp_workflow.html



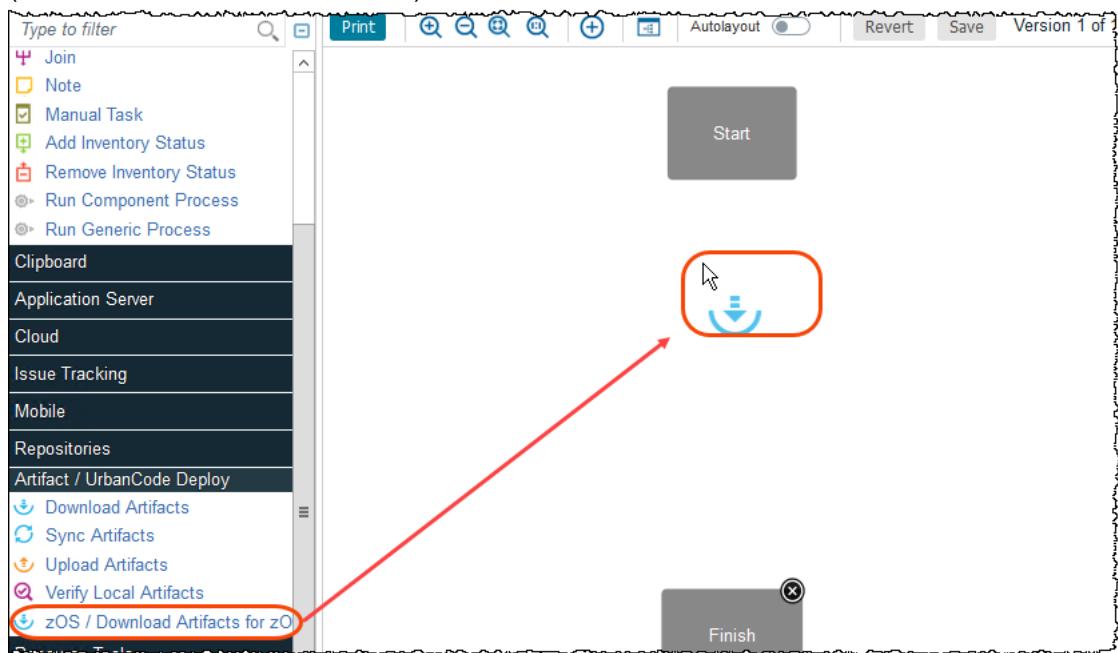
8.6 We are going to create a simple process for updating an existing CICS application.

- On the left hand side of the editor you will find an extensive **Palette** of process steps.
- Scroll down and expand **Repositories**.

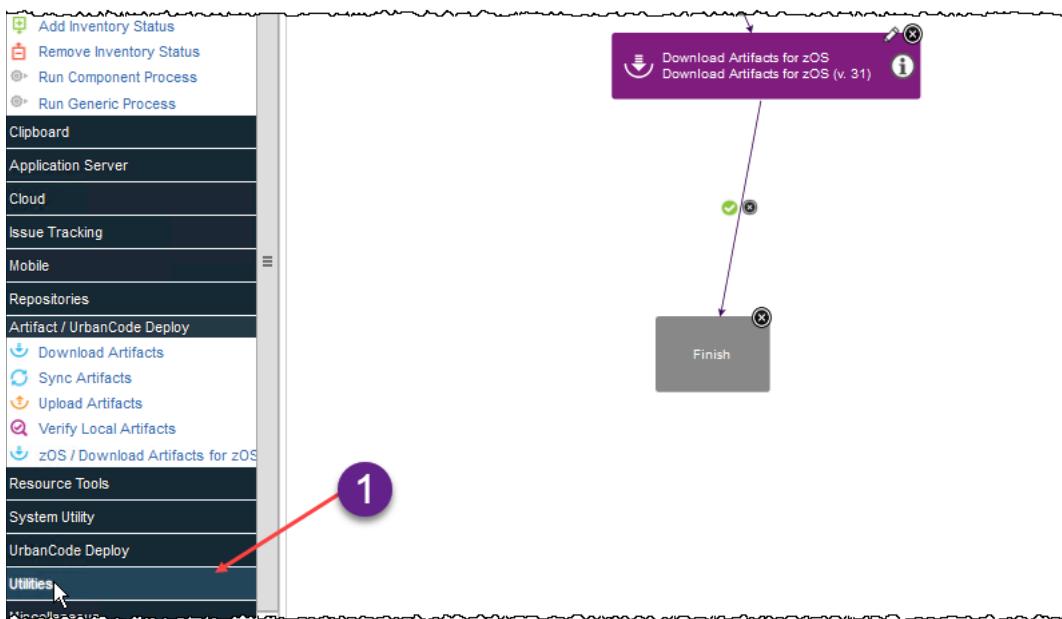
First you are going to use the step **zOS /Download Artifacts for zOS**:



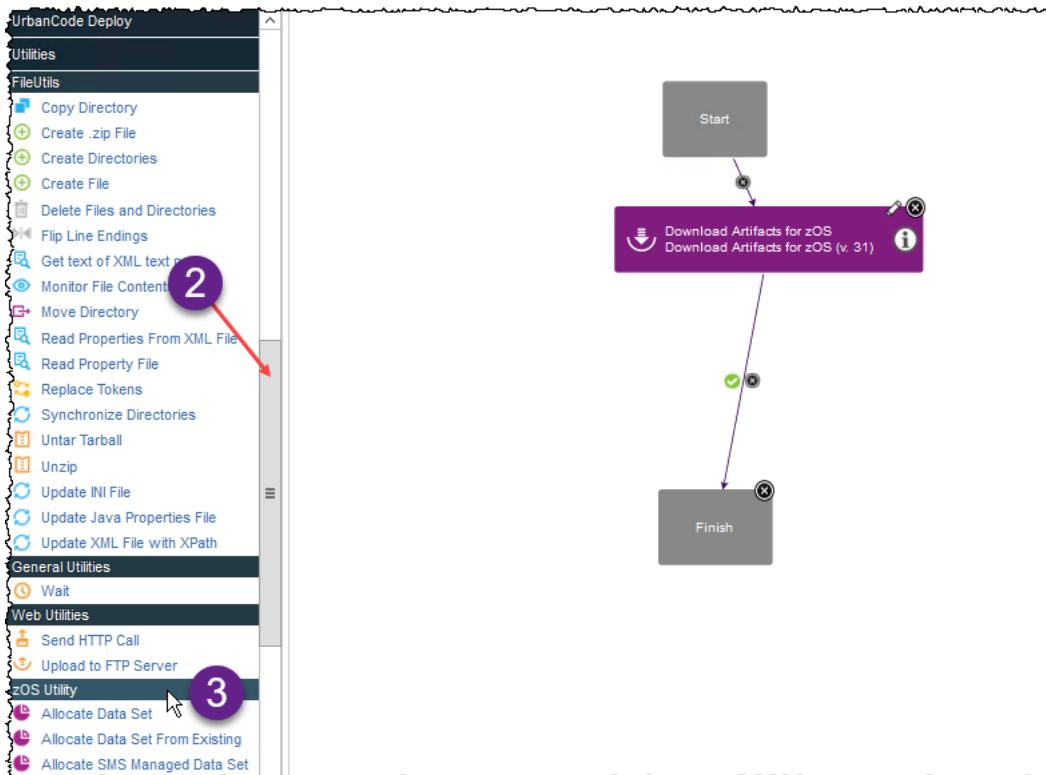
8.7 ► Drag and drop the first step, **zOS /Download Artifacts for zOS**, onto the design space (somewhere under the **Start** block).



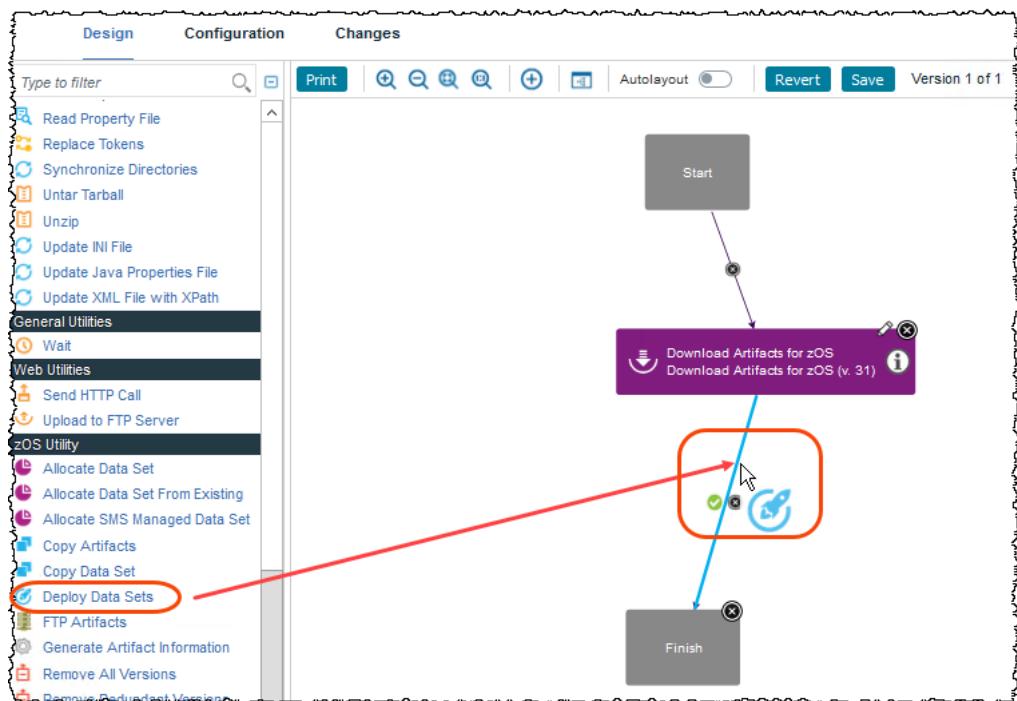
- 8.8 ► 1 Using the Design palette on left expand **Utilities**, clicking on it



- 2 Scroll down and 3 expand **zOS Utility**: The **Deploy Datasets** step will also be needed to load component artifacts from the z/OS repository and to bring them into z/OS work area for later deploy.



8.9 ► Scroll down and drag the **Deploy Data Sets** step onto the line that connects to the *Finish* as below



Properties

Properties can be set in UCD for many different things, including components, environments, processes, and applications. You can also set global properties for the system.

You can refer to a property by scope:

`${p:scope/propertyName}`

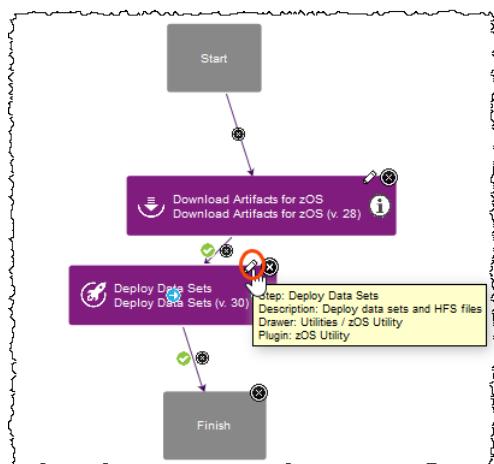
or without scope:

`${p:propertyName}`

For more details, refer to the full documentation on Properties:

http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.reference.doc/topics/ud_properties.html

8.10 ► Click on the pencil icon ☰ on **Deploy Data Sets** to change the UCD properties for this step.

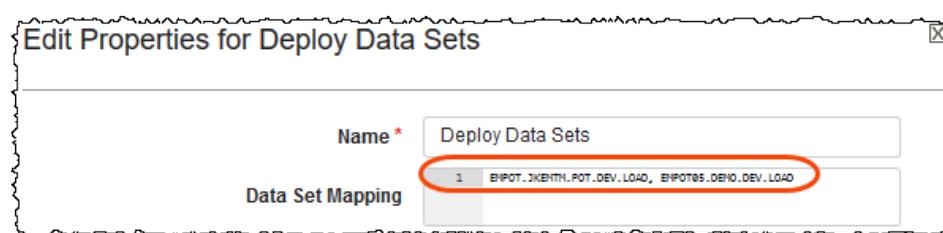
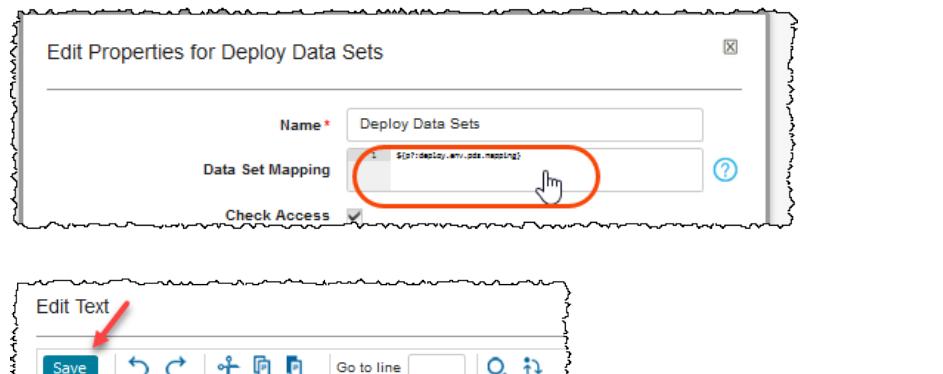


8.11 ► Enter the value for the *Data Set Mapping* property:
EMPOT.JKEMTM.POT.DEV.LOAD, EMPOT05.DEMO.DEV.LOAD

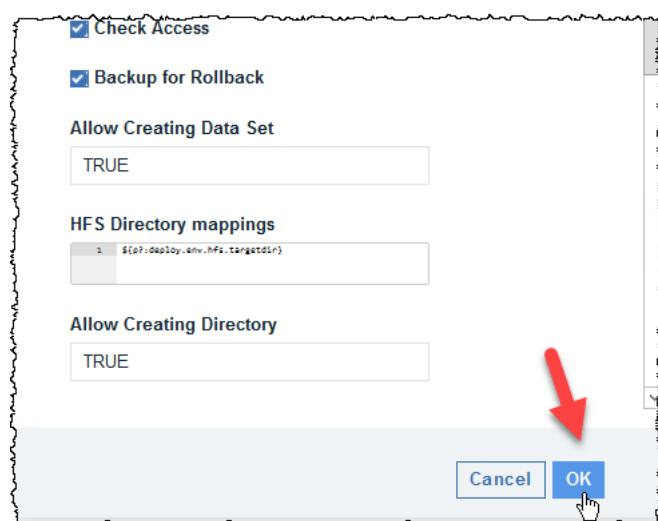
Data Set Mapping specifies which PDS members packaged with the component to deploy, and where to deploy them.

A mapping rule should follow format "From PDS, To PDS":

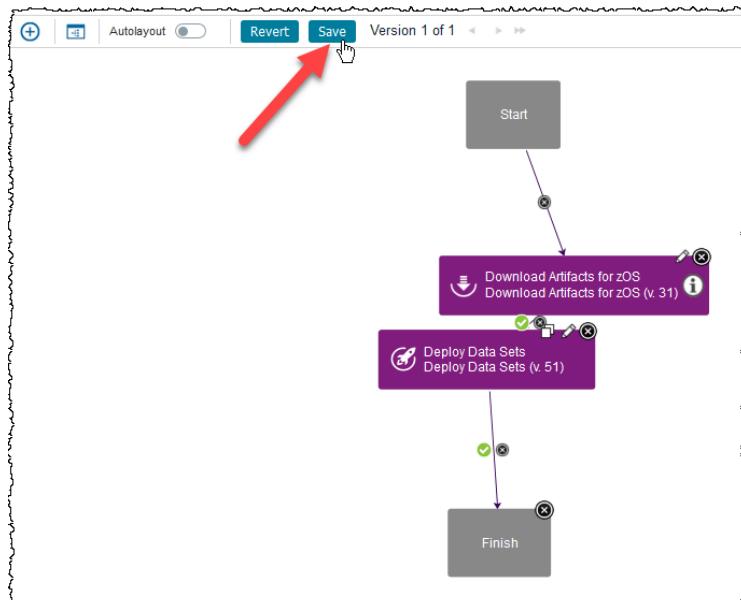
We could have that as variable but we will fix here to save time.



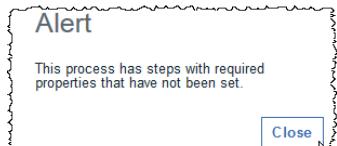
8.12 ► Click **OK** to save the properties updated



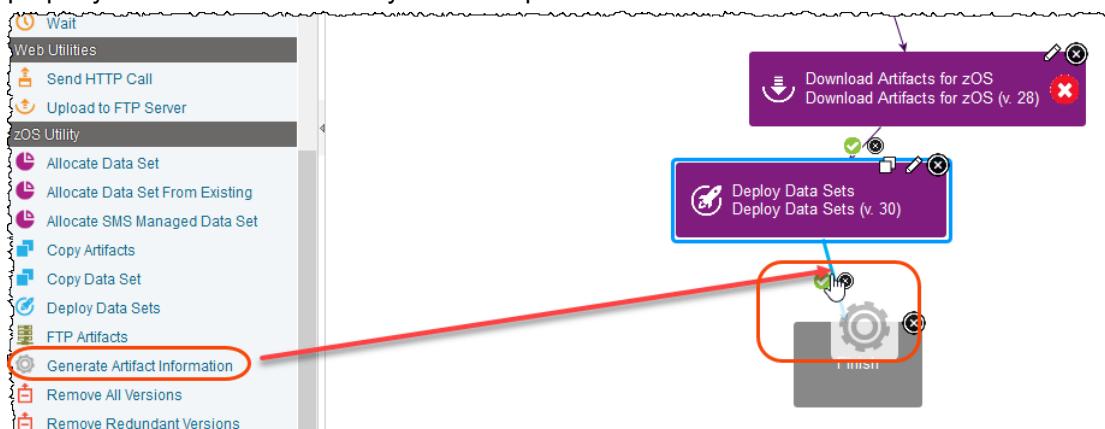
8.13 ► Click the **Save** button to save what you had done so far.



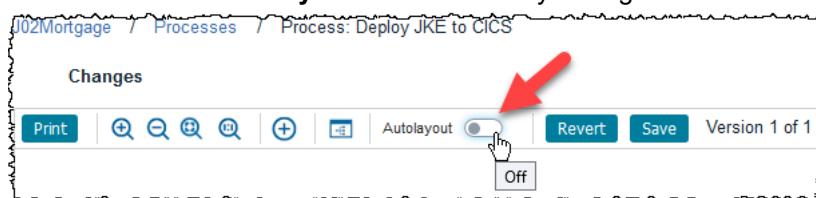
8.14 ► If there is a warning about missing property ignore it, we will fix that later.



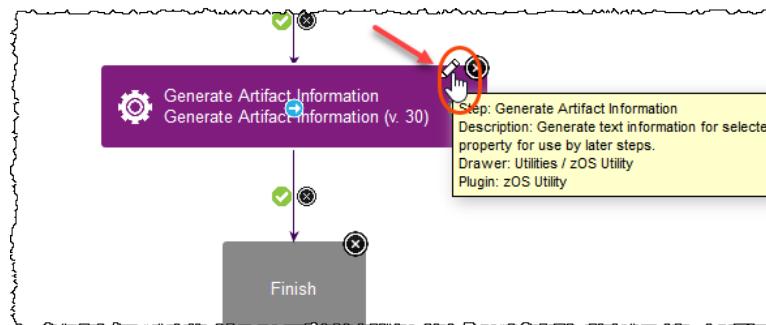
8.15 ► Scroll down on left pallet and drag and drop **Generate Artifact Information** on the line before Finish. You will need to specify a list of the load modules that CICS needs to do the *Newcopy*. This step generate text information for selected version artifacts. Information is put to output property 'text' to be consumed by a later step



8.16 ► Click on **Autolayout** to have the layout organized



- 8.17 ►| Click on the pencil icon  on **Generate Artifact Information** to change the UCD properties for this step.



- 8.18 ►| ① Change from *Generate Artifact Information* to **Generate Program List** (must respect upper/lower case and spaces)

- | ② Add **CICS_LOAD** to *Deploy Type Filter*
►| ③ Scroll down to add another property

Edit Properties for Generate Artifact Information

Name*
Generate Program List 1

For Each*
PDS Member

Order By*
ASC Order

Container Name Filter

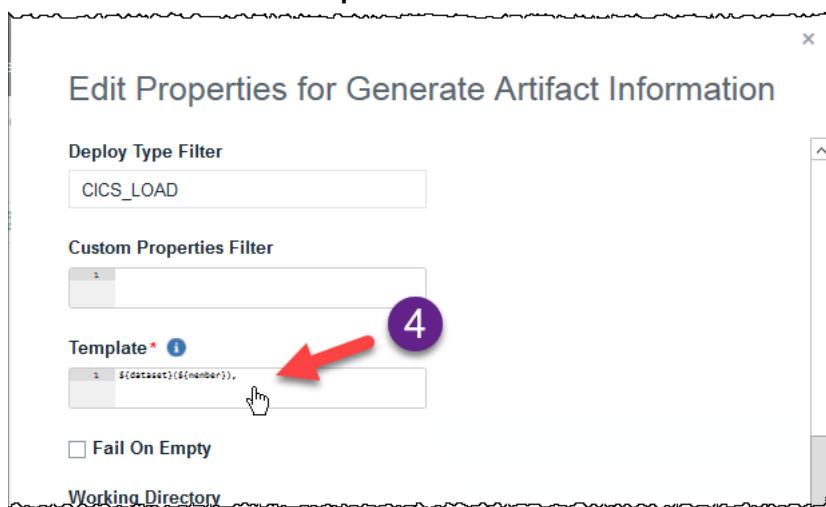
Target Data Set Name Filter

Resource Name Filter

Deploy Type Filter
CICS_LOAD 2

3 

8.19 ➡ 4 Click on **Template**



➡ Change the value from “

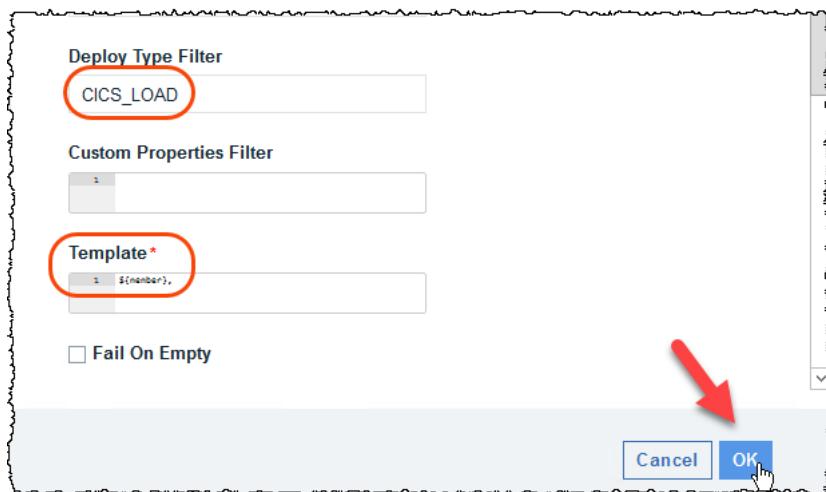
`${dataset} (${member}),`” to `${member},`

IMPORTANT ➔ Be sure that you add a , (comma) after the `${member},`

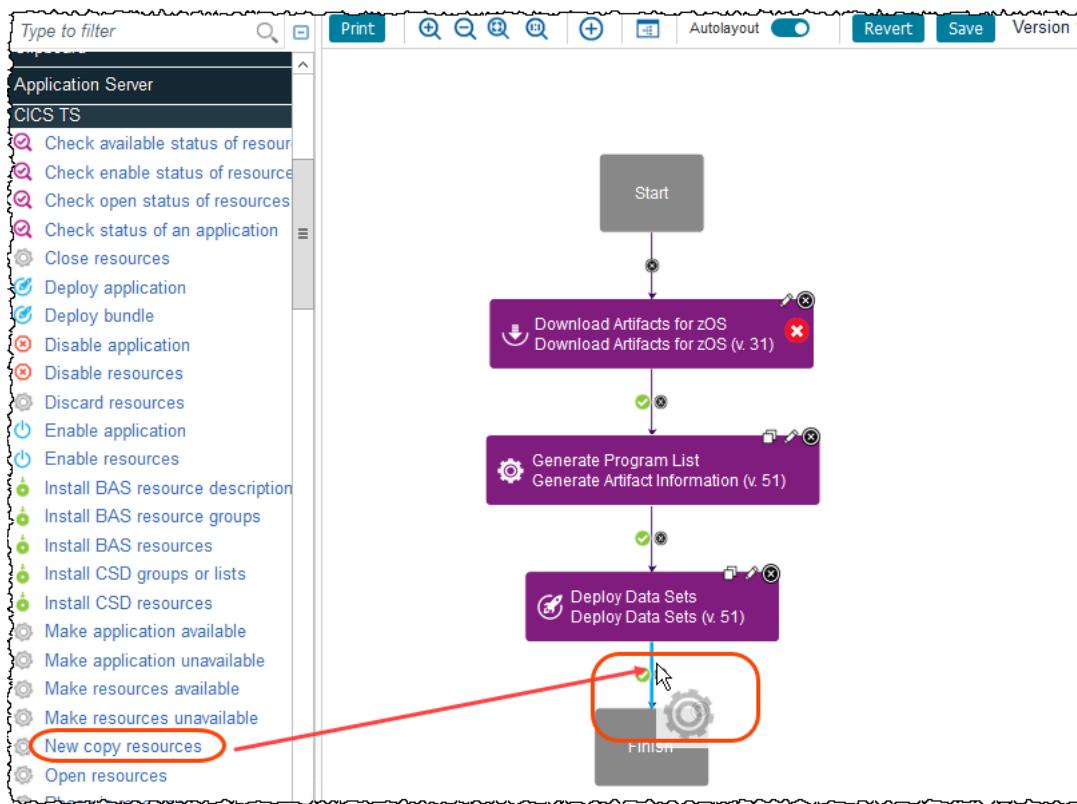


➡ Click **Save** to save the modified value

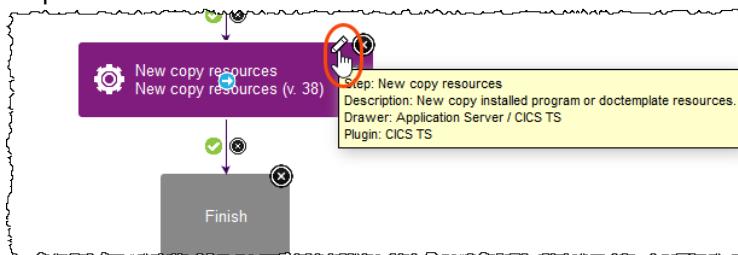
8.20 ➡ Click **OK** to close the dialog



- 8.21 ►► On the Palette (left side) expand **Application Server** (scroll up) and **CICS TS**.
 ►► Drag and drop **New copy resources** to the line that connects to the *Finish* box..

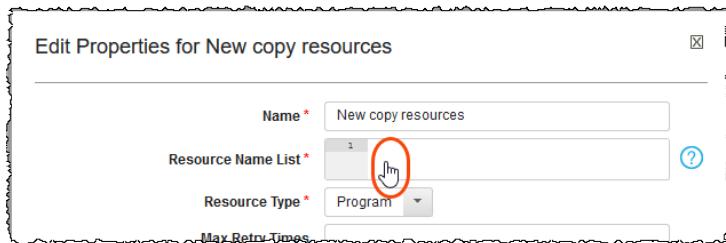


- 8.22 ►► Click on the pencil icon ► on **New copy resources** to change the UCD properties for this step.

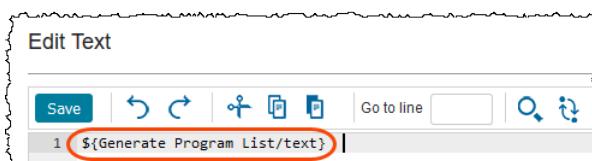


8.23 You need to specify a list of the load modules that CICS needs to do the *Newcopy*.

►| Click on Resource Name List

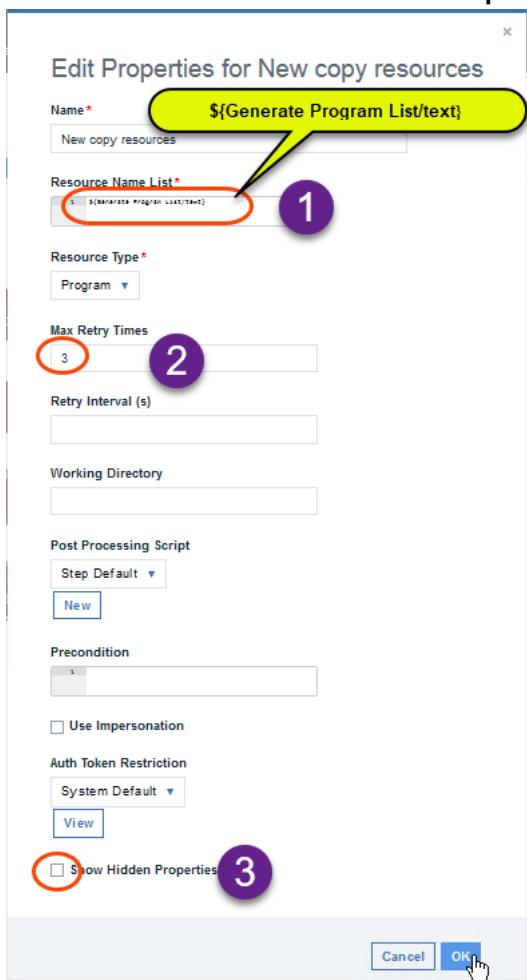


►| Type **\${Generate Program List/text}** (This is the step name that you created on 8.18) and click **Save**



►| Type 3 for Max Retry Times

►| Scroll down and click **Show Hidden properties**.



8.24 Scroll down to see those properties that need to be filled. Later we will do that at the **Test** level

Edit Properties

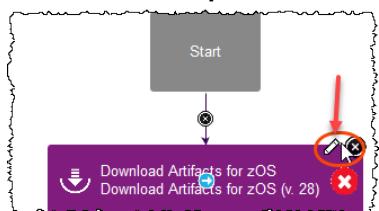
Show Hidden Properties

Host *	<input type="text" value="\${p:cics.host}"/>
Port *	<input type="text" value="\${p:cics.cmcpport}"/>
CICSPlex	<input type="text" value="\${p?:cics.cicsplex}"/>
Scope	<input type="text" value="\${p?:cics.scope}"/>
Username	<input type="text" value="\${p?:cics.username}"/>
Password	<input type="password" value="*****"/>
Enable SSL	<input type="text" value="\${p?:cics.ssl}"/>
Keystore Location	<input type="text" value="\${p?:cics.kslocation}"/>
Keystore Type	<input type="text" value="\${p?:cics.kstype}"/>
Keystore Password	<input type="password" value="*****"/>
Truststore Location	<input type="text" value="\${p?:cics.tslocation}"/>
Truststore Type	<input type="text" value="\${p?:cics.tstype}"/>
Truststore Password	<input type="password" value="*****"/>

▶ Click **OK** to save it

8.25 You still need to fix a warning on first step (see the icon).

▶ Click on the pencil icon on **Download Artifacts for zOS** to possibly change the properties.



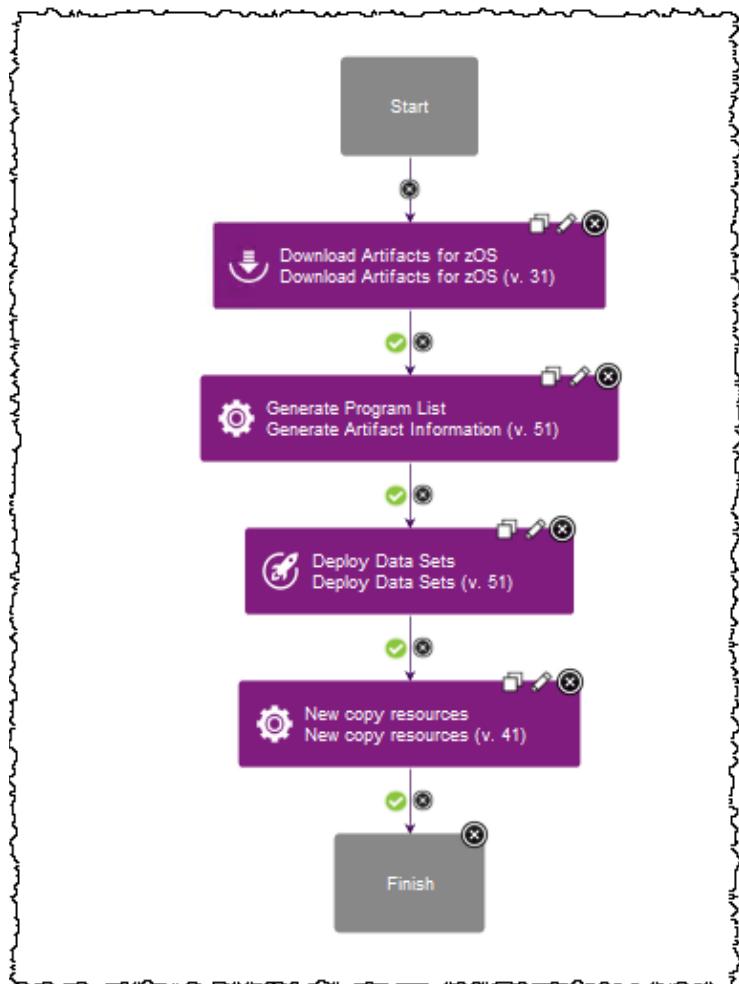
8.26 ▶ No changes are required here and click **OK**

Edit Properties for Download Artifacts for zOS

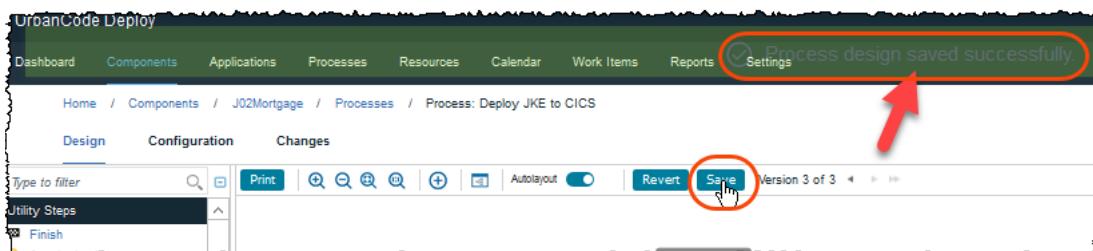
Name *	<input type="text" value="Download Artifacts for zOS"/>
Directory Offset *	<input type="text" value="."/>
Working Directory	<input type="text"/>
Post Processing Script	<input type="button" value="Step Default"/> <input type="button" value="New"/>
Precondition	<input type="text" value="1"/>
Use Impersonation	<input type="checkbox"/>
Show Hidden Properties	<input type="checkbox"/>

OK **Cancel**

8.27 The final result should look similar to this:



8.28 ► Finally, click the **Save** button located in the upper left portion of the process editor to save the process:



A message will indicate that the process is saved.

Task 9 - Create an application process

Application processes direct underlying component processes and orchestrate multi-component deployments. An application process, like a component process, consists of steps that are configured with the process editor.

In this task, you create an application process that installs the UCD application defined .

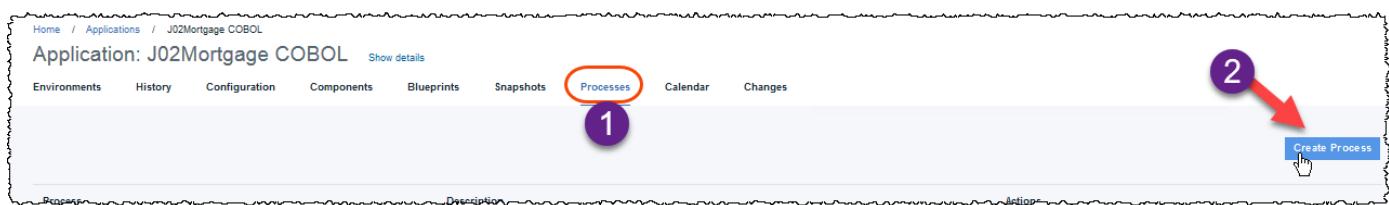
- 9.1 ► Go to Applications page, select your application (**J02 Mortgage - COBOL**),

Name	Template	Description
A HC zOS COBOL CICS		Deploy the HC application to CICS - No DB2 Binding
Account Management CICS and Worklight (LINUX)		Used in PDTTOOLS demo
J02 Mortgage zOS and Worklight		J02 CICS + JKE Mobile
J02Mortgage COBOL		
JKE Emulator		

- 9.2 ► Click Components tab and click Add Component:

- 9.3 ► Select your UCD component created previously (**J02Mortgage**) and click Save:

9.4 ► Click the Processes tab and click **Create Process**:



9.5 ► Give a name to your process, like **Deploy J02 to CICS** and click **Save**:

Create an Application Process

Name*
Deploy J02 to CICS (1)

Description

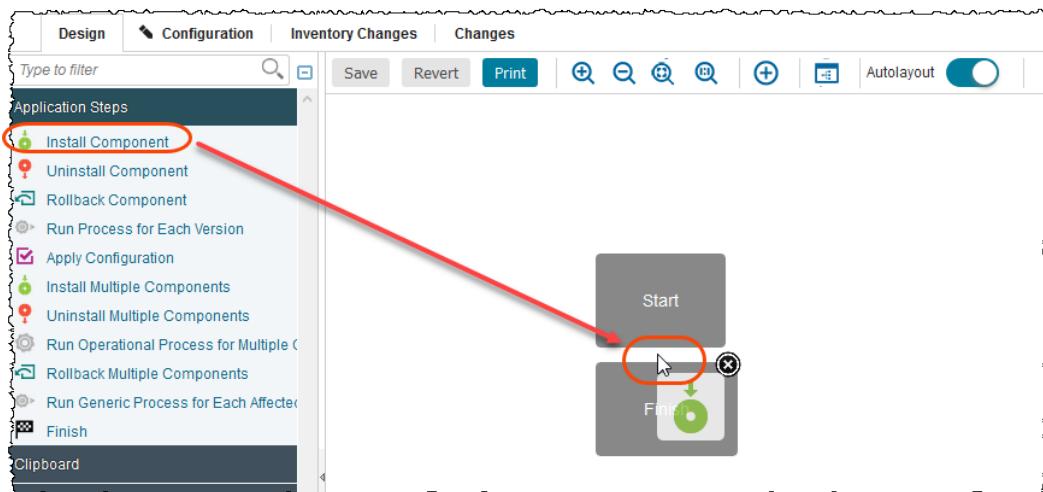
Inventory Management*
Automatic

Offline Agent Handling*
Check Before Execution

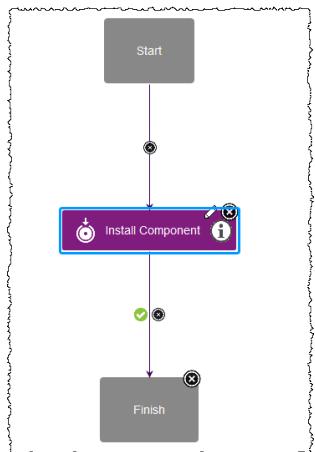
Required Role (2)

Cancel Save

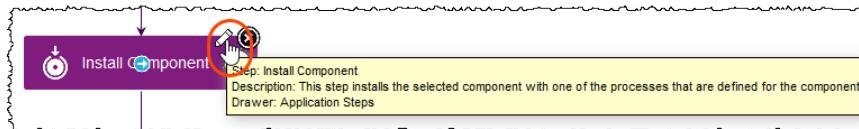
9.6 ► Find **Install Component...** step in the Design Palette, and drag it onto the space between *Start* and *Finish* boxes.



9.7 The result will be as below:

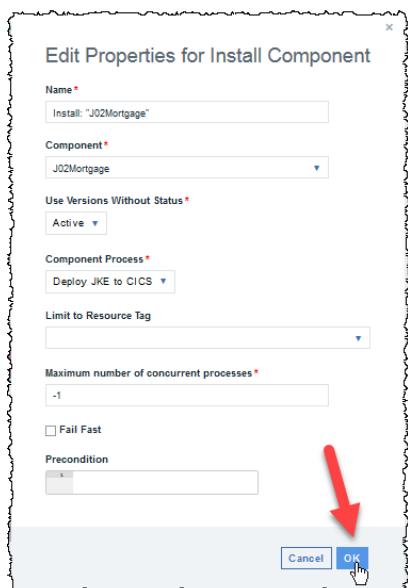


9.8 ► Click on the pencil icon 🖍 on **Install Component** to possibly change the properties

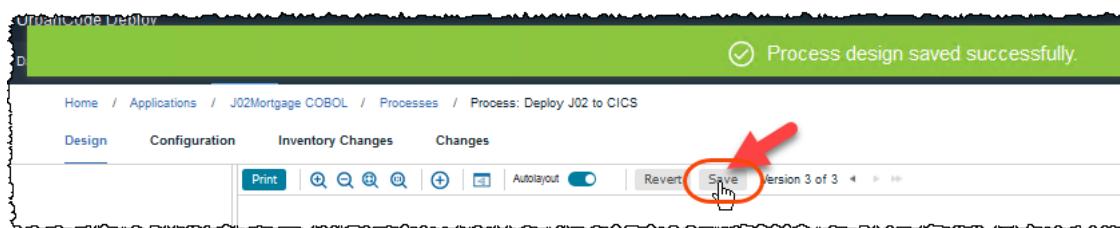


9.9 Notice that the properties of this step are pre filled for you.

► Click OK:



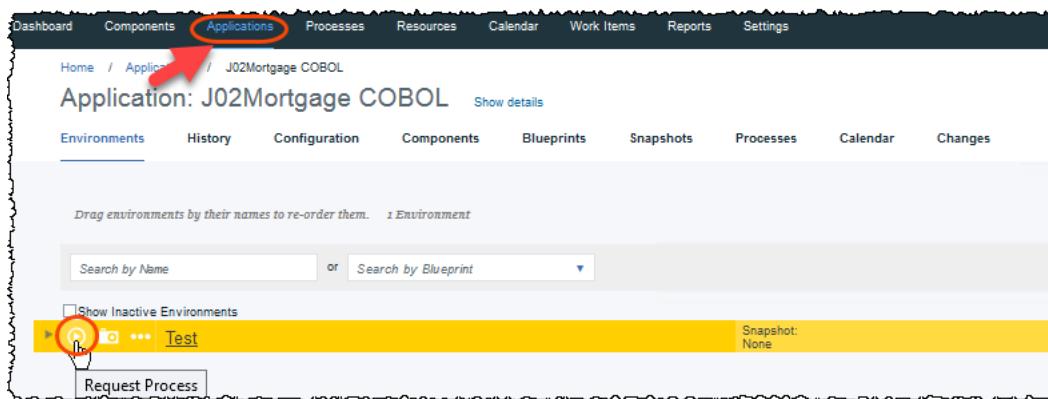
9.10 ► Click Save



Task 10 – Environment properties configuration

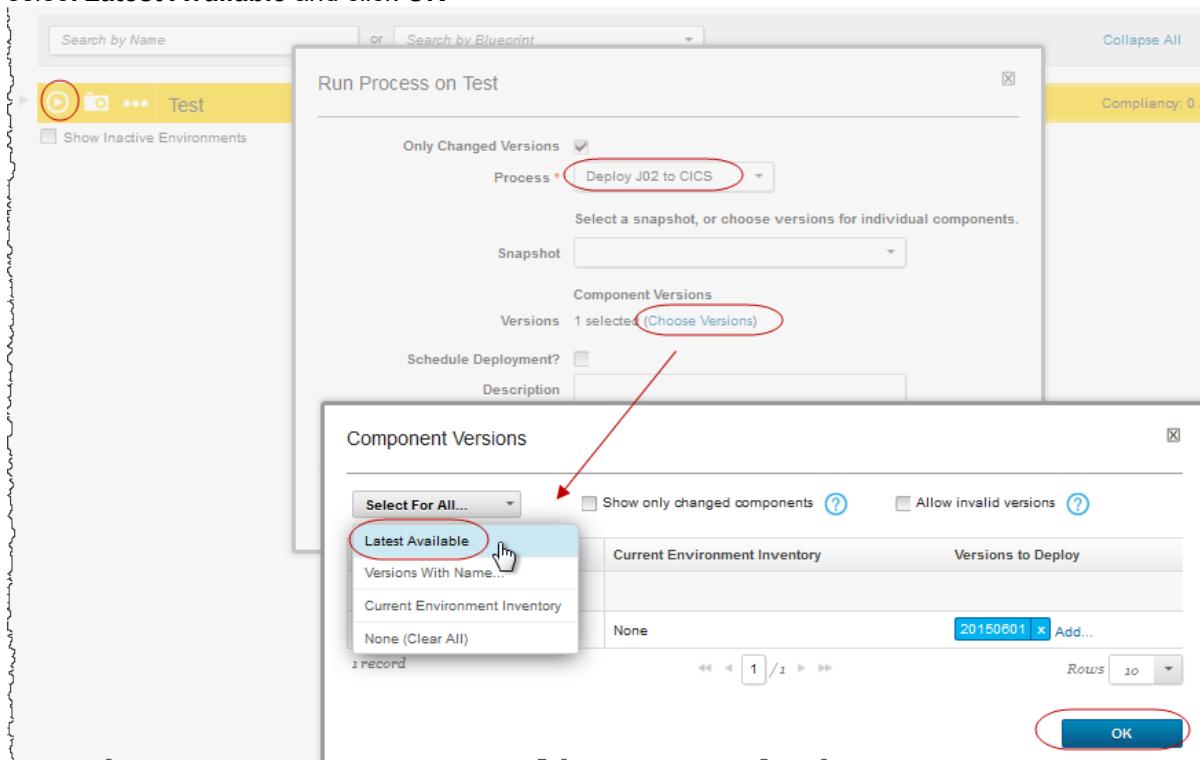
On the task 7 you created the **Test** environment. For this lab we have just one environment, but usually we have few environments. Example the **Test** environment could reflect the Test LPAR and **Prod** environment that would reflect other LPAR. Each environment may have different values used on the deploy. Example the CICS test environment uses port 30090, the prod environment uses port 30091, etc.. Those values in our lab were not yet defined and you will see errors when trying to deploy the application for **Test** environment..

- 10.1 ► Go to Application > J02 Mortgage COBOL and click the icon  on **Test** environment



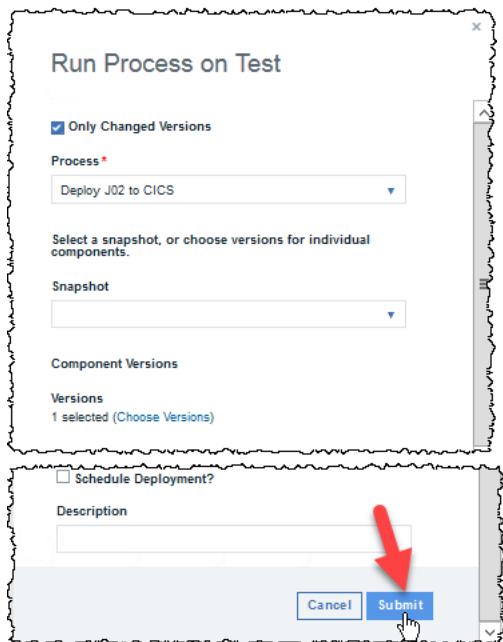
The screenshot shows the Application Details page for 'J02Mortgage COBOL'. The 'Environments' tab is active. At the bottom of the page, there is a yellow bar containing the 'Test' environment name and a 'Request Process' button. A red circle highlights the 'Applications' tab in the top navigation bar.

- 10.2 ► Select the Process Deploy J02 to CICS , click Choose Versions, click Select For All.. select Latest Available and click OK



The screenshot shows the 'Run Process on Test' dialog. The 'Process' dropdown is set to 'Deploy J02 to CICS'. Under 'Component Versions', there is a link '(Choose Versions)' which is highlighted with a red circle. A red arrow points from this link to the 'Component Versions' dialog, which is also highlighted with a red circle. In the 'Component Versions' dialog, the 'Latest Available' option is selected. The 'OK' button at the bottom right of the dialog is also highlighted with a red circle.

10.3 ► When the dialog below is shown click **Submit**



The Deploy process start, but it will fail as you see soon..

10.4 ► Click **Expand all** on the far right

Step	Progress	Start Time	Duration	Status
► 1. Install: "J02Mortgage"	0 / 1	4:57:55 PM	0:00:49	Running
Total Execution	0 / 1	4:57:55 PM	0:00:49	Running

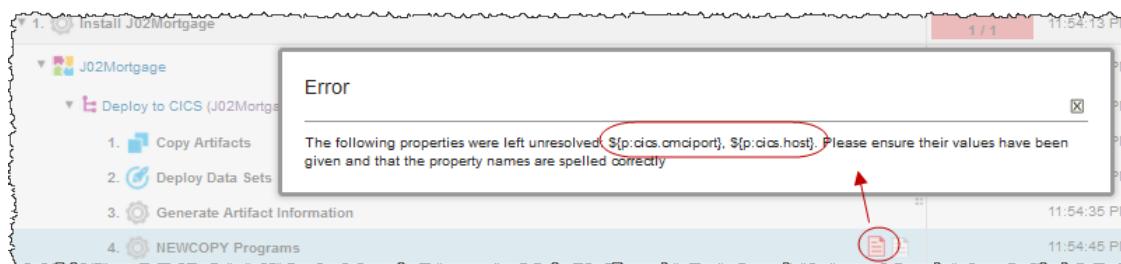
10.5 ► Click the icon ► to expand all steps being executed

Step	Progress	Start Time	Duration	Status
▼ 1. Install: "J02Mortgage"	0 / 1	5:00:49 PM	0:00:06	Running
▼ J02Mortgage	0 / 1	5:00:49 PM	0:00:06	Running
► Deploy JKE to CICS (J02Mortgage 20180709)	0 / 1	5:00:49 PM	0:00:06	Running
1. Download Artifacts for zOS	0 / 1	5:00:50 PM	0:00:08	Running
2. Deploy Data Sets				Not Started
3. Generate Program List				Not Started
4. New copy resources				Not Started
Total Execution	0 / 1	5:00:49 PM	0:00:06	Running

10.6 ► Click on Error Log icon

Step		Progress	Start Time	Duration	Status
v 1.	Install: "J02Mortgage"	1 / 1	1:11:20 PM	0:03:12	● Failed
	J02Mortgage	1 / 1	1:11:20 PM	0:03:12	● Failed
	Deploy JKE to CICS (J02Mortgage_20180709)				
1.	Download Artifacts for zOS		1:11:20 PM	0:03:12	● Failed
2.	Generate Program List		1:12:35 PM	0:00:35	● Success
3.	Deploy Data Sets		1:13:11 PM	0:01:21	● Success
4.	New copy resources		1:14:31 PM	0:00:00	● Failed
Total Exec	(CICS TS v. 4.1.20181207-0633)	Error Log	1 / 1	1:11:20 PM	0:03:12

10.7 ► See the properties missing and close the dialog



10.8 We could add properties at various levels.. In our example let's make the variables at the Environment level. The missing properties are:

cics.cmciport and **cics.host**.

10.9 To add the properties..

► Click Applications > J02 Mortgage COBOL > Test

10.10 ► Select Configuration > Environment properties and click Add Property

10.11 ► Add the property **cics.cmciport** with value **1490** and click **Save**

Add Property

Name*
cics.cmciport

Description

Secure

Value
1490

Cancel Save

10.12 ► Add the property **cics.host** with value **10.1.1.2** and click **Save**

Add Property

Name*
cics.host

Description

Secure

Value
10.1.1.2

Batch Edit Add Property

1 of 1 pages

Cancel Save

10.13 ► Using the **Add Property** button

add the property **cics.userid** with the value **empot02** (your userid)

10.14 ► Also add the property **cics.password** with the value **empot02**.

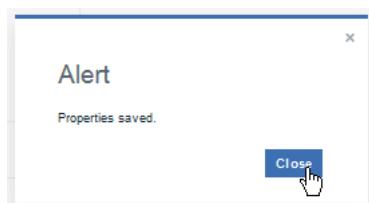
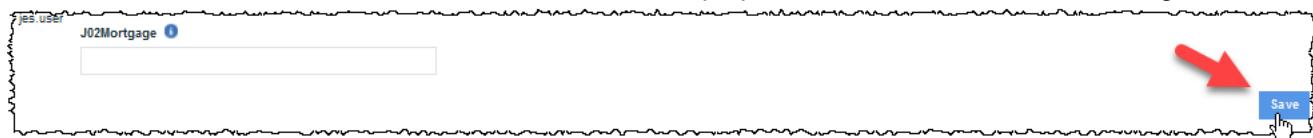
In the real world you will not add the password here and will use other secure way to authenticate., also

In the Edit Property dialog, you can check the secure box and this will hide (secure) the password..

When complete:

Name	Value	Description	Actions
cics.cmciport	1490		Edit Delete
cics.host	10.1.1.2		Edit Delete
cics.password	empot02		Edit Delete
empot02	empot02		Edit Delete

- 10.15 ► Scroll down and click **Save** to save the defined properties and **Close** to close the dialog.



PART 3 – Deploy the application to z/OS CICS

On this part you will deploy the Application to the z/OS CICS.

To deploy the components in the application, you must run the application process on the specified environment. We created only one environment named **Test**, you will deploy this environment.

Task 11 – Run an application process

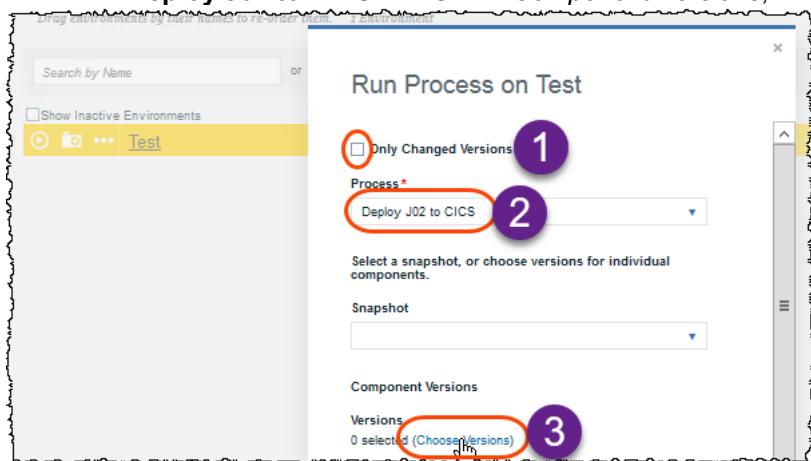
You are now ready to test our deployment process.

- 11.1 ► Click the Applications tab and then click **J02 Mortgage COBOL**

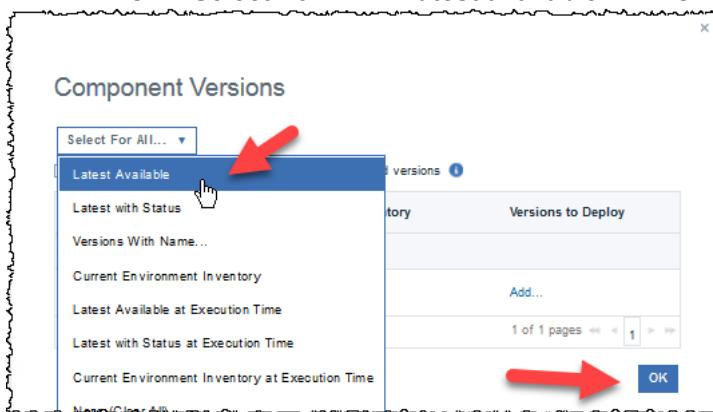
Name	Template	Description	Created
A HC zOS COBOL CICS		Deploy the HC application to CICS - No DB2 Binding	12/14/2017, 4:51 PM
Account Management CICS and Worklight (LINUX)		Used in PDTTOOLS demo	8/18/2015, 11:02 PM
J02 Mortgage zOS and Worklight		J02 CICS + JKE Mobile	12/16/2014, 1:38 PM
J02Mortgage COBOL			5/8/2020, 8:37 AM
JKE Emulator			12/15/2014, 8:17 PM

- 11.2 ► Under Test environment click the Request Process

- 11.3 ► In the **Run Process** window, un-select **Only Changed Versions**, in the **Process**, select the **Deploy J02 to CICS** and Under **Component Versions**, click **Choose Versions**

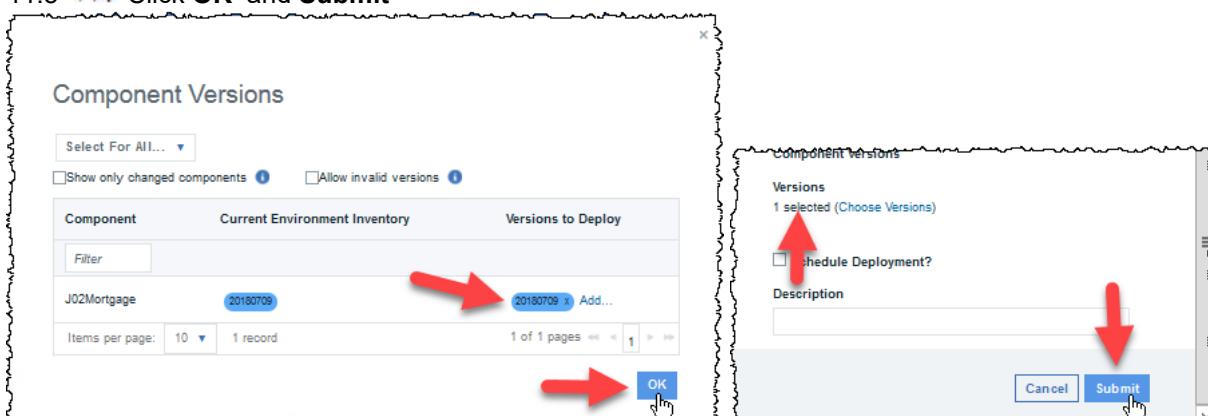


- 11.4 ► Click **Select For All > Latest available** click **OK**



Make sure that the version that you created on PART 1 of the lab is selected for **J02Mortgage**. If you do not select a version, that component is not deployed.

- 11.5 ► Click **OK** and **Submit**



The web page shows you the progress of the application process request. From this page, you can watch as the processes run. The following figure shows the application process partially completed. The application component process is finished and the other two component processes are running.

11.6 ► Click **Expand All** and expand **Deploy JKE to CICS** to see all steps and WAIT the deploy...

Application Process Request: J02 Mortgage COBOL [\(show details\)](#)

Log	Properties	Manifest	Configuration Changes	Inventory Changes																																													
Execution <div style="display: flex; justify-content: space-between;"> Pause Cancel Download All Logs </div> <table border="1"> <thead> <tr> <th>Step</th> <th>Progress</th> <th>Start Time</th> <th>Duration</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>▼ 1. Install: "J02Mortgage"</td> <td>0 / 1</td> <td>5:32:19 PM</td> <td>0:00:48</td> <td>Running</td> </tr> <tr> <td>▼ J02Mortgage</td> <td>0 / 1</td> <td>5:32:19 PM</td> <td>0:00:48</td> <td>Running</td> </tr> <tr> <td> ▼ Deploy JKE to CICS (J02Mortgage 20180709)</td> <td>0 / 1</td> <td>5:32:19 PM</td> <td>0:00:48</td> <td>Running</td> </tr> <tr> <td> 1. Download Artifacts for zOS</td> <td>0 / 1</td> <td>5:32:20 PM</td> <td>0:00:35</td> <td>Success</td> </tr> <tr> <td> 2. Deploy Data Sets</td> <td>0 / 1</td> <td>5:32:56 PM</td> <td>0:00:12</td> <td>Running</td> </tr> <tr> <td> 3. Generate Program List</td> <td>0 / 1</td> <td></td> <td></td> <td>Not Started</td> </tr> <tr> <td> 4. New copy resources</td> <td>0 / 1</td> <td></td> <td></td> <td>Not Started</td> </tr> <tr> <td>Total Execution</td> <td>0 / 1</td> <td>5:32:19 PM</td> <td>0:00:48</td> <td>Running</td> </tr> </tbody> </table>					Step	Progress	Start Time	Duration	Status	▼ 1. Install: "J02Mortgage"	0 / 1	5:32:19 PM	0:00:48	Running	▼ J02Mortgage	0 / 1	5:32:19 PM	0:00:48	Running	▼ Deploy JKE to CICS (J02Mortgage 20180709)	0 / 1	5:32:19 PM	0:00:48	Running	1. Download Artifacts for zOS	0 / 1	5:32:20 PM	0:00:35	Success	2. Deploy Data Sets	0 / 1	5:32:56 PM	0:00:12	Running	3. Generate Program List	0 / 1			Not Started	4. New copy resources	0 / 1			Not Started	Total Execution	0 / 1	5:32:19 PM	0:00:48	Running
Step	Progress	Start Time	Duration	Status																																													
▼ 1. Install: "J02Mortgage"	0 / 1	5:32:19 PM	0:00:48	Running																																													
▼ J02Mortgage	0 / 1	5:32:19 PM	0:00:48	Running																																													
▼ Deploy JKE to CICS (J02Mortgage 20180709)	0 / 1	5:32:19 PM	0:00:48	Running																																													
1. Download Artifacts for zOS	0 / 1	5:32:20 PM	0:00:35	Success																																													
2. Deploy Data Sets	0 / 1	5:32:56 PM	0:00:12	Running																																													
3. Generate Program List	0 / 1			Not Started																																													
4. New copy resources	0 / 1			Not Started																																													
Total Execution	0 / 1	5:32:19 PM	0:00:48	Running																																													

If the process runs successfully, the request shows that each component process is finished, as in the following figure:

Execution

Start	Progress	Status	Duration	End
6:08:25 PM	1 / 1	Success	0:02:23	6:10:49 PM
<div style="display: flex; justify-content: space-between;"> Repeat Request Download All Logs Expand All Collapse All </div>				
Step	Progress	Start Time	Duration	Status
▼ 1. Install: "J02Mortgage"	1 / 1	6:08:25 PM	0:02:23	Success
▼ J02Mortgage	1 / 1	6:08:25 PM	0:02:23	Success
▼ Deploy JKE to CICS (J02Mortgage 20180709)	1 / 1	6:08:25 PM	0:02:23	Success
1. Download Artifacts for zOS	1 / 1	6:08:25 PM	0:00:40	Success
2. Deploy Data Sets	1 / 1	6:09:06 PM	0:00:45	Success
3. Generate Program List	1 / 1	6:09:52 PM	0:00:27	Success
4. New copy resources	1 / 1	6:10:19 PM	0:00:29	Success
Total Execution	1 / 1	6:08:25 PM	0:02:23	Success

11.7 ► Click on **Output Log** to see the CICS Programs NEWCOPY executed.

Output Log

37 2020/05/08 20:51:10.758 GMT BUZCP0006I Connected to "10.1.1.2:1490". CICS TS version: 050300.
38 2020/05/08 20:51:17.893 GMT BUZCP0037I Perform NEWCOPY Operation.
39 2020/05/08 20:51:18.904 GMT BUZCP0024I NEWCOPY PROGRAM "J02CMORT" succeeded.
40 2020/05/08 20:51:19.202 GMT BUZCP0024I NEWCOPY PROGRAM "J02MORT" succeeded.
41 2020/05/08 20:51:19.714 GMT BUZCP0028I Summary: 2 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.
42

11.8 ► Close the web browser

Task 12 – Verifying the Deploy results at z/OS CICS

The z/OS components are unique for each student, so you can see the code that you deployed to CICS.

12.1 Use the 3270 emulator

Go to the windows desktop and **double click the 3270 emulation icon**



12.2 Verify the CICS application. L cicsts53

Type **I cicsts53** (I is L lower case)

```

z/OS V2R2 PUT1606 / RSU1607                               IP Address = 10.1.1.1
                                                               VTAM Terminal = SC0TCP01

Application Developer System
      // 0000000  SSSSS
      // 00    00 SS
zzzzzz // 00    00 SS
      zz // 00    00 SSSS
      zz // 00    00   SS
zzzzzz // 0000000  SSSS

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 cicsts53_
Mn H                                         24/011

```

12.3 Authenticate using your id **empot02** and password **empot02**

```

Signon to CICS                                APPLID CICSTS53
WELCOME TO CICS TS 5.3

Type your userid and password, then press ENTER:
  Userid . . . empot02  Groupid . . .
  Password . . . _
  Language . . . _

  New Password . . .

DFHCE3520 Please type your userid.
F3=Exit
Mn A                                         11/033

```

12.4 ► Type the transaction **J02P**.



12.5 ► Type **00** on Length of Loan in Years. Note that the error message.

```
Session A - [24 x 80]
File Edit View Communication Actions Window Help
Host: zos.dev Port: 23 LU Name:
-
JKE MORTGAGE CALCULATOR

Amount of Loan: 1000
Length of Loan in Years: 00
Interest Rate: 5

Press F3 to quit or Enter to calculate loan
Press PF9 to see companies that can match or beat this rate

Monthly Payment: 7.91

LOAN TERM MUST BE BETWEEN 1 AND 40 YEARS

MA 01/001
Connected to remote server/host zos.dev using ls
```

Congratulations! You completed the lab.

LAB 6B – (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.

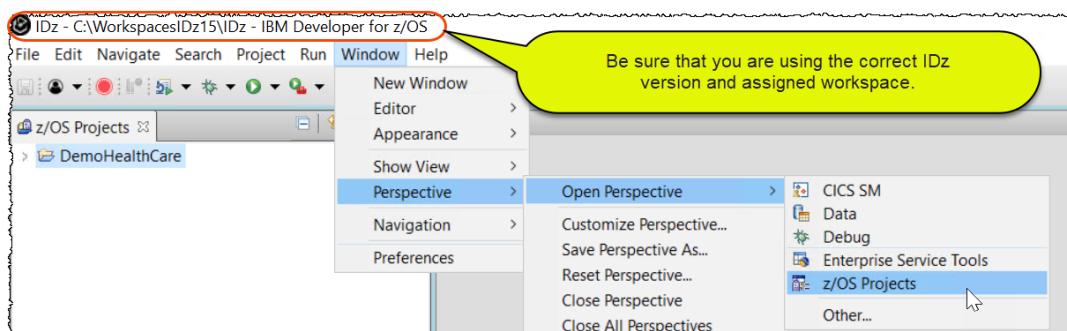
1.1.1 Start *IBM Developer for z Systems version 15* if it is not already started

- 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**

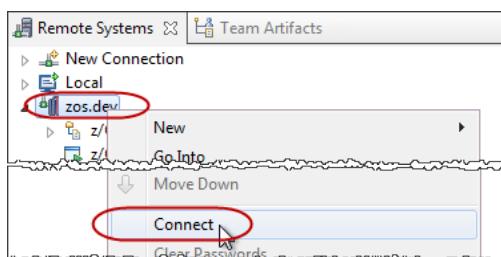


1.1.3 You will use userid **empot01**.

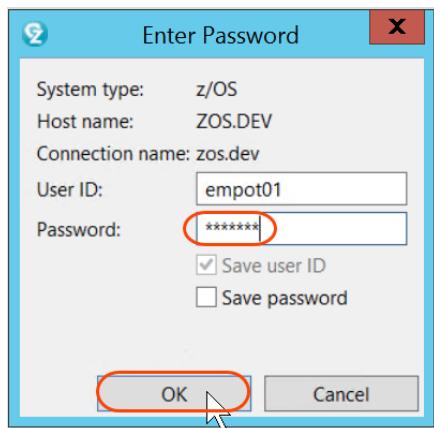
If you are connected as empot01, jump to step 1.1.5 Otherwise.

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

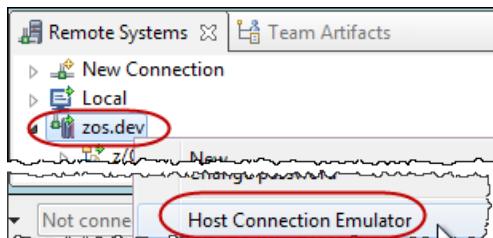
(You also may need to disconnect first)



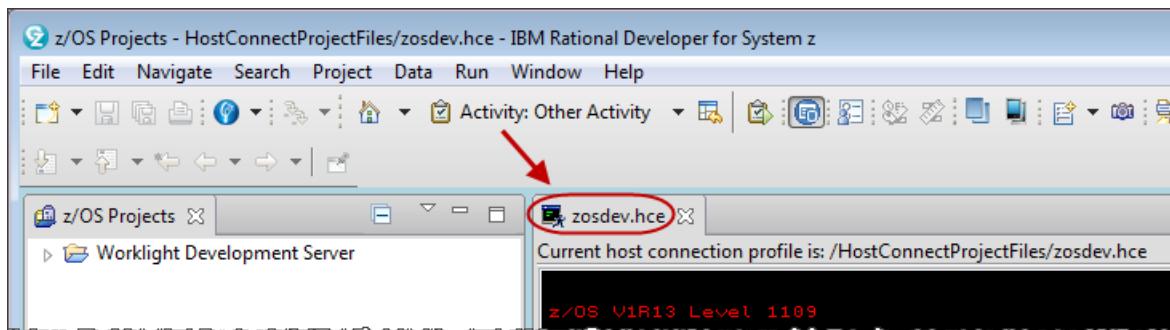
- 1.1.4 ► Type **empot01**as userid and **empot01** as password.
 The userid and password can be any case; don't worry about having it in UPPER case.
 Click **OK** to connect to z/OS.



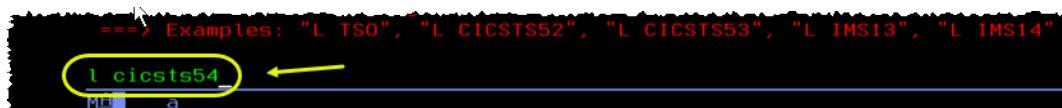
- 1.1.5 ► Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator**.



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



- 1.1.7 ► Type **I cicsts54**. (where "I" 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**.



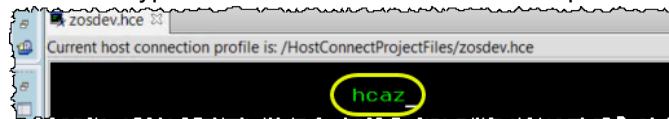
1.1.9 The sign-on message is displayed



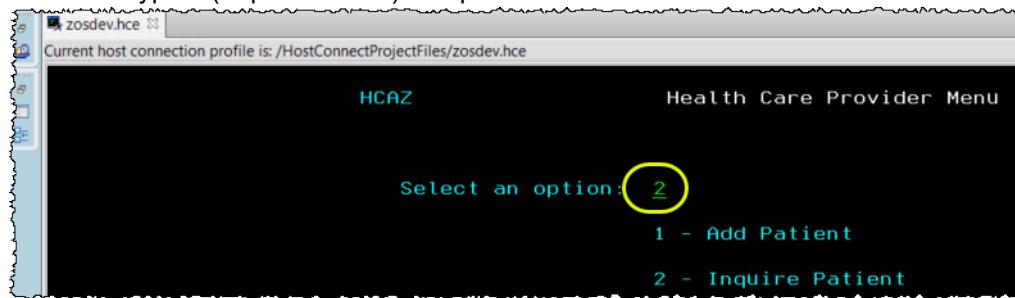
1.2 Run CICS transaction HCAZ

You should now be in the z/OS CICS region named C/CSTS54. This is the CICS instance where you will make the recording of your interaction.

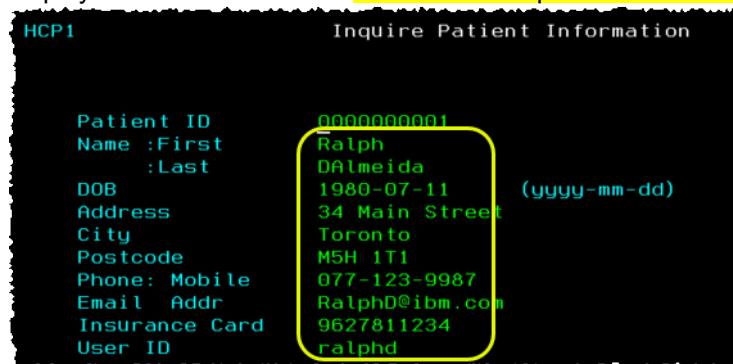
1.2.1 ► Type the CICS transaction **hcaz** and press the **Enter** key.



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



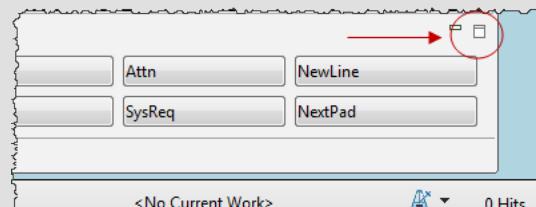
1.2.3 ► Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**



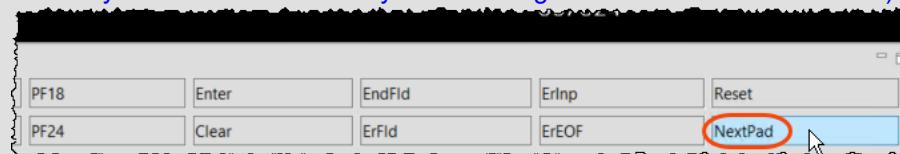
IF you need to use functions like Clear or Reset

If you need to use the **clear** function use the key **Esc**.

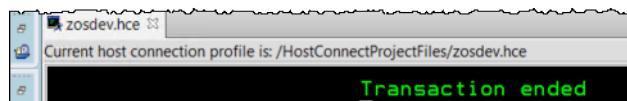
Also you may look in the right lower corner, select this icon  . This will display possible keys, including the clear button.



You may also use the Reset key after clicking NextPad

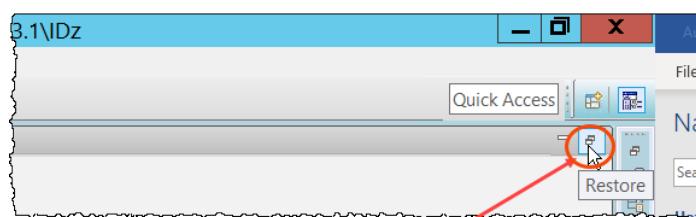


1.2.4  Press **F3** to end the application.



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.

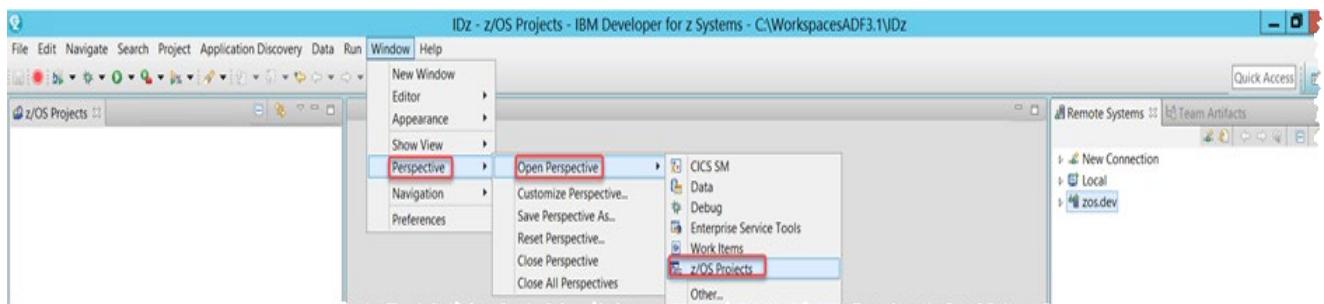
Section 2 – Import a z/OS project

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

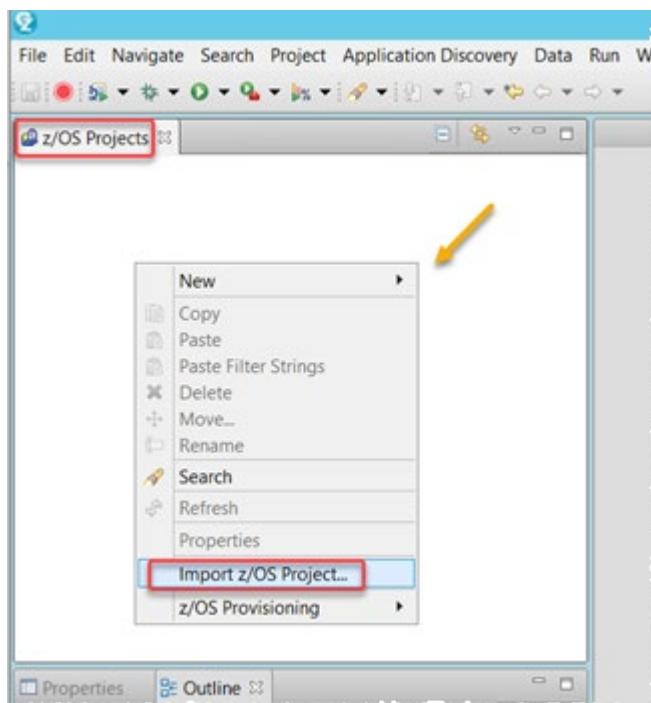
2.1 Importing the LAB6B z/OS Project

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).

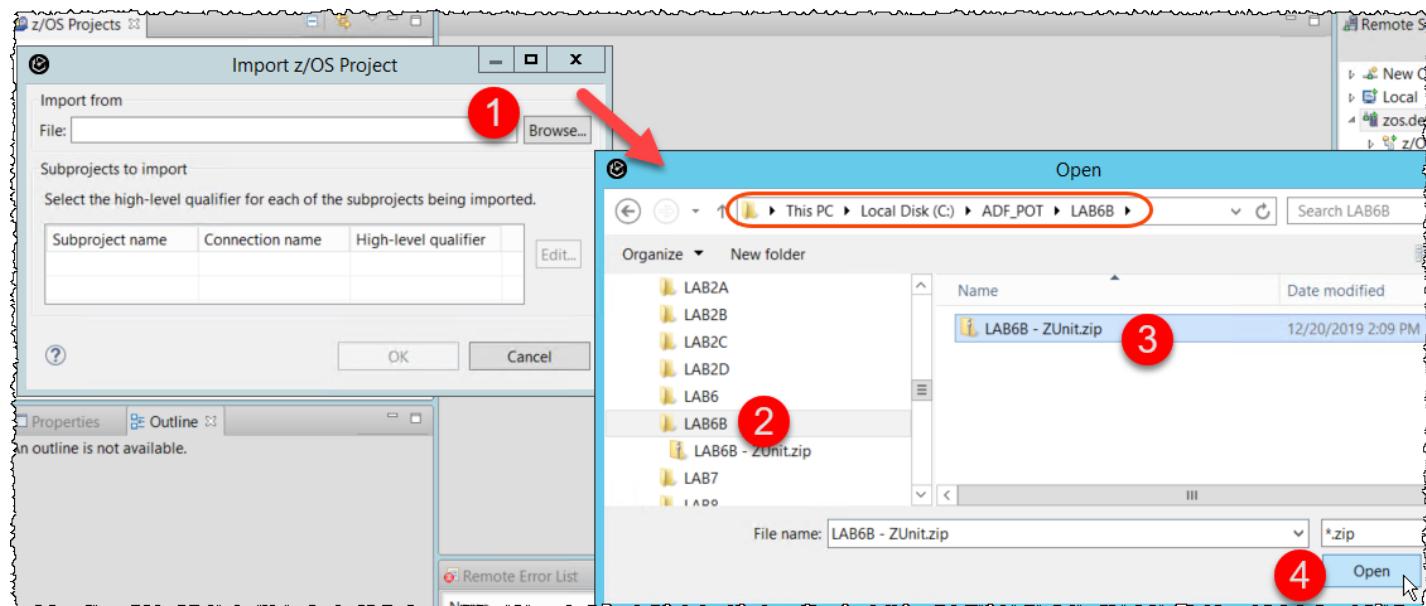
- 2.1.1 ► If not already in the *z/OS Projects* perspective, open the ***z/OS Projects*** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



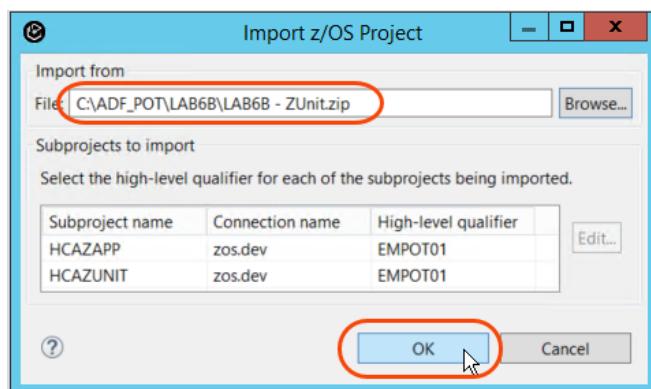
- 2.1.2 ► Using the ***z/OS Projects*** view on top left, position the cursor anywhere in the view. Right mouse click and select the **Import z/OS Project** action.



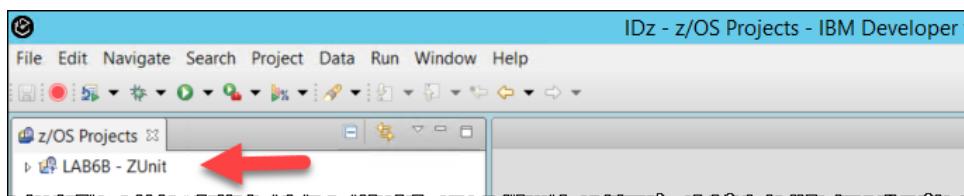
2.1.3 ► In the **Import z/OS Project** dialog, click the **Browse** button, navigate to **C:\ADF_POT\LAB6B** and select **LAB6B - ZUnit.zip**. Click **Open**.



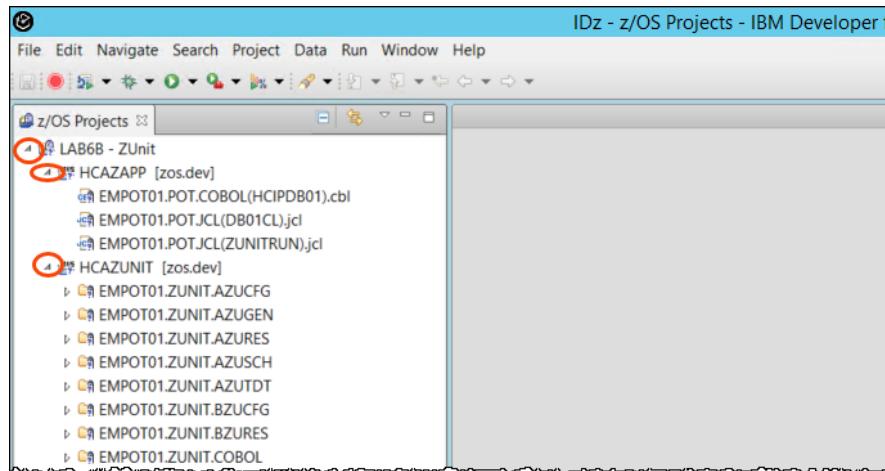
2.1.4 ► In the dialog that opens, click **OK**.



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



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



Section 3 – Record data interaction using the CICS application.

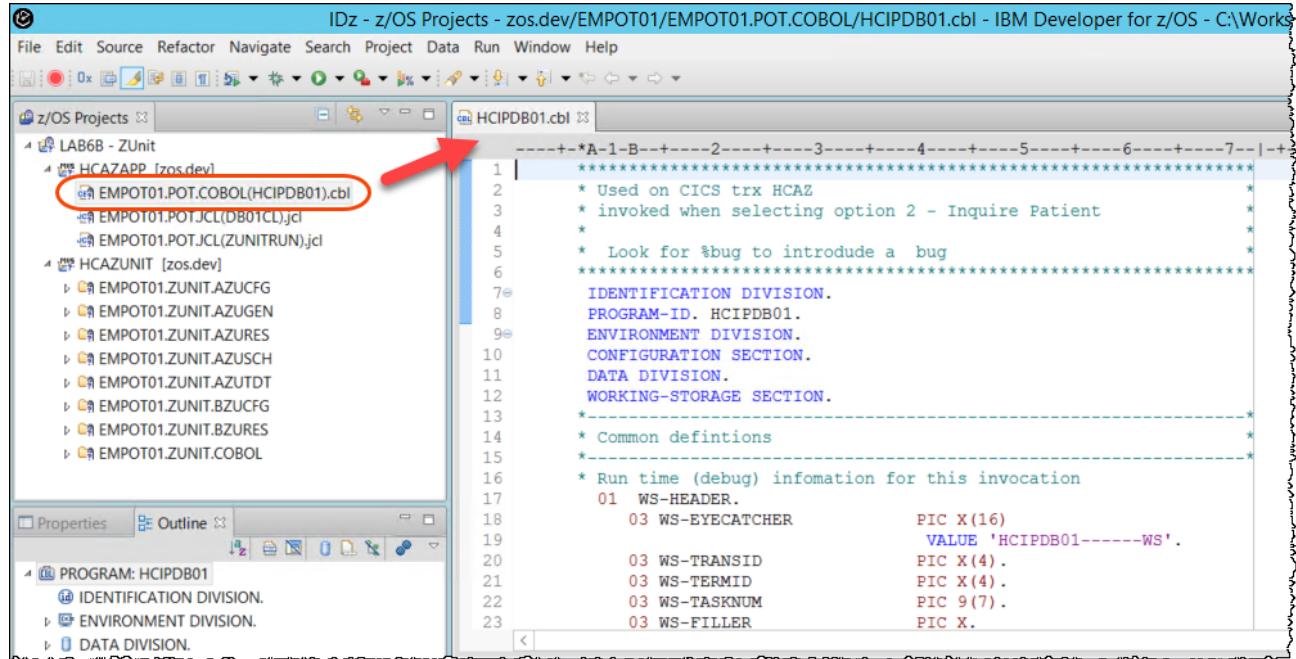
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

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.



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..

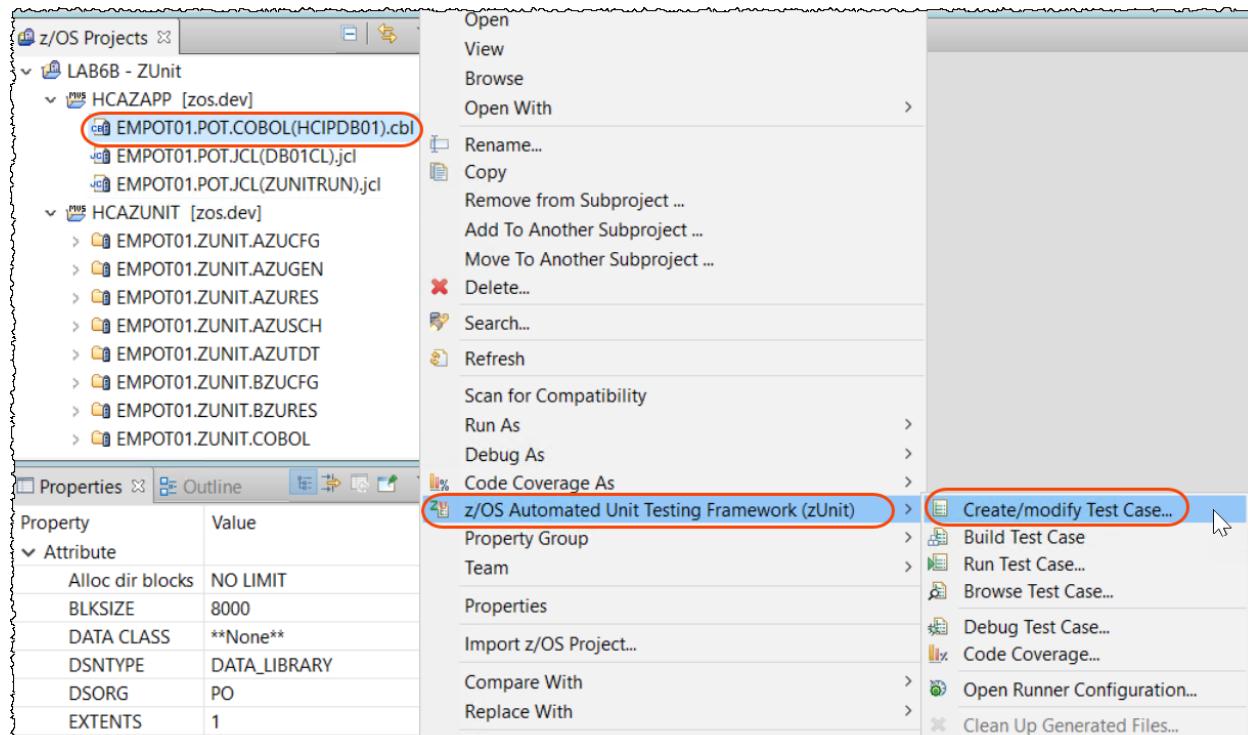
```

109      * END PROGRAM and return to caller
110      *
111      MAINLINE-END.
112      EXEC CICS RETURN END-EXEC.
113      MAINLINE-EXIT.
114      EXIT.
115
116      GET-PATIENT-INFO.
117      EXEC SQL
118          SELECT FIRSTNAME,
119              LASTNAME,
120              DATEOFBIRTH,
121              insCardNumber,
122              ADDRESS,
123              CITY,
124              POSTCODE,
125              PHONEMOBILE,
126              EMAILADDRESS,
127              USERNAME
128          INTO :CA-FIRST-NAME,
129              :CA-LAST-NAME,
130              :CA-DOB,
131              :CA-INS-CARD-NUM,

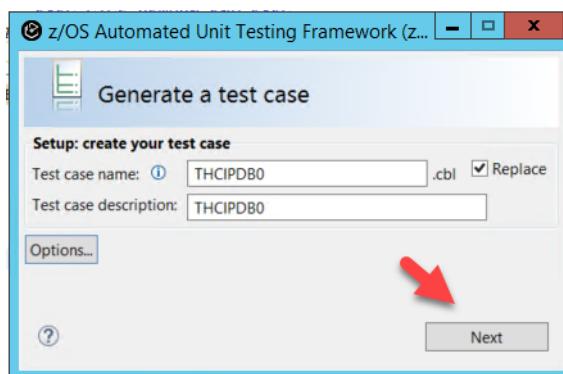
```

3.2 Recording the COBOL program that sends the message

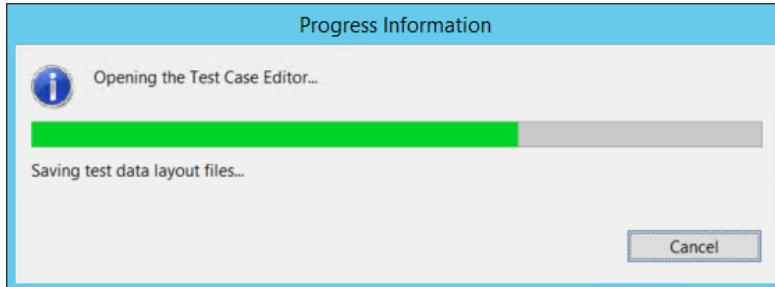
3.2.1 ► To start the recording, right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)**-> **Create/modify Test Case..**



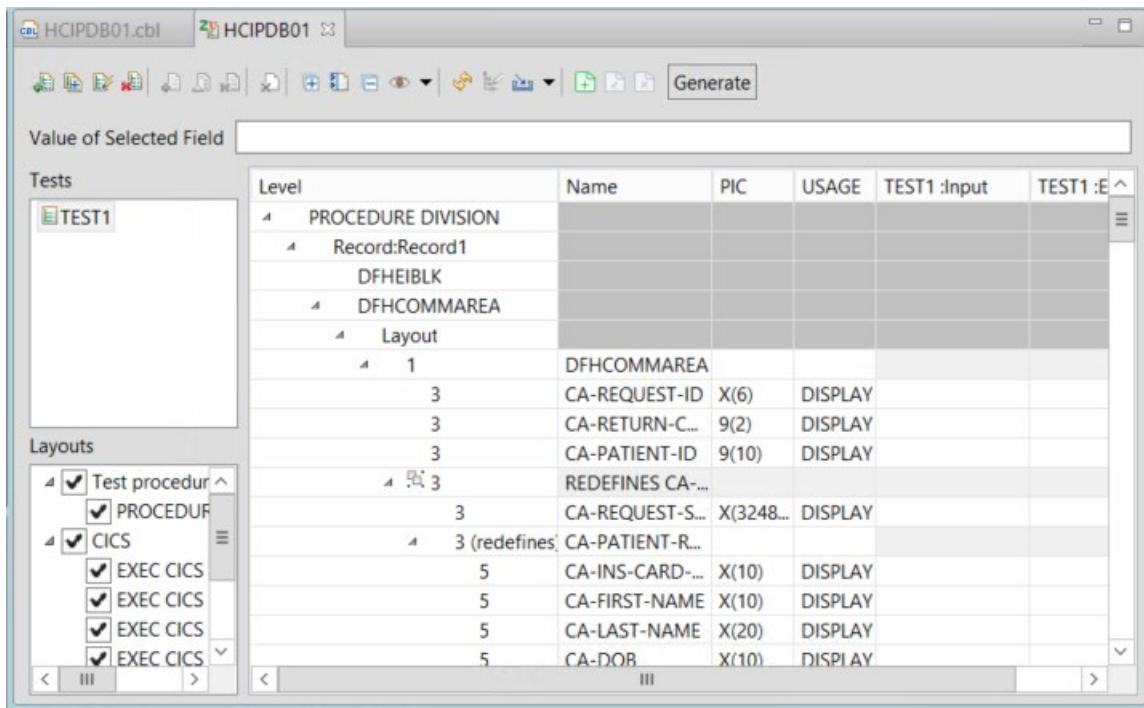
3.2.2 ► This opens a dialog where you can name your test case. Accept the auto-generated name and click **Next**.



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

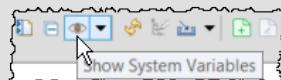


3.2.4 This will open the **Test Case Editor**, as shown below. TEST1 may or may not be on your screen. If TEST1 is there you will delete on next step.



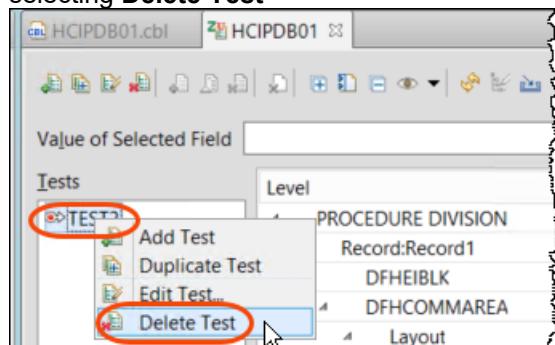
Understanding the test case editor

- 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



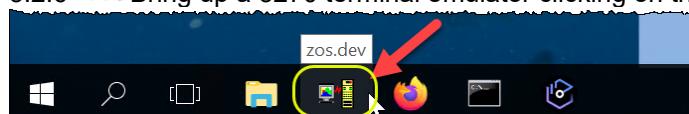
The variables that are returned by the program are the output from the program logic that a developer should be checking

- 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**



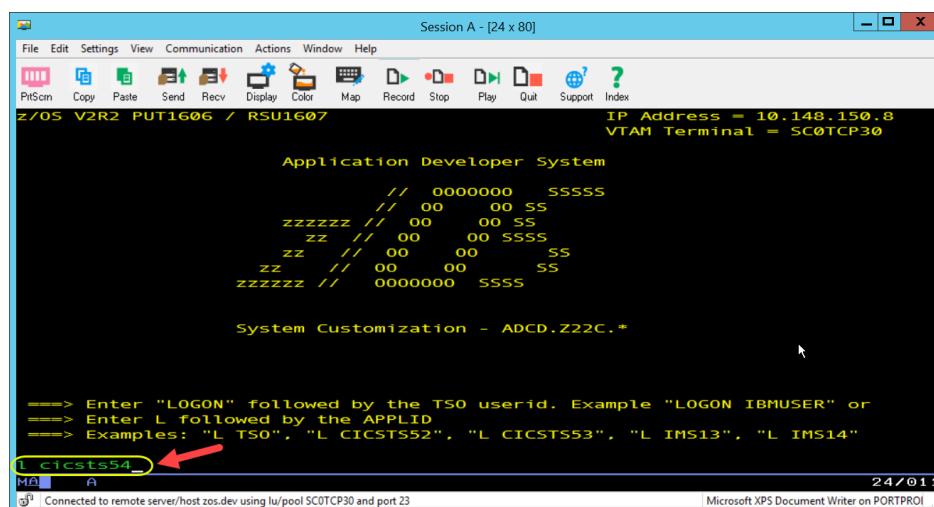
Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

- 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 **I cicsts54**. (where I is L lower case) and press **Enter key** or the **right Ctrl key**.

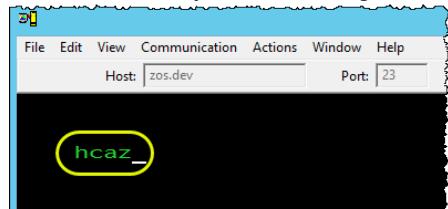


- 3.2.8 ► Sign on using **empot01** as the userid and **empot01** as the password.

Type your userid and password, then press ENTER:

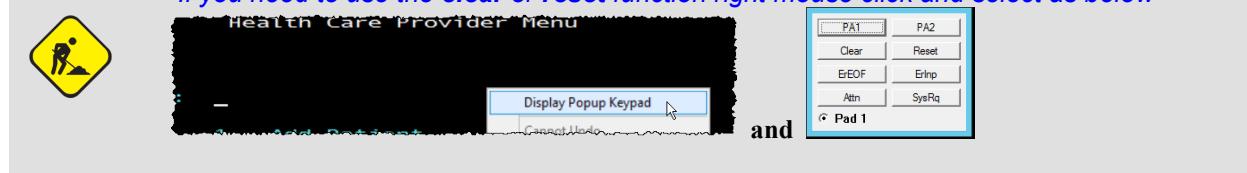
Userid . . .	empot01	Groupid . . .
Password . . .	_____	
Language . . .	_____	

- 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

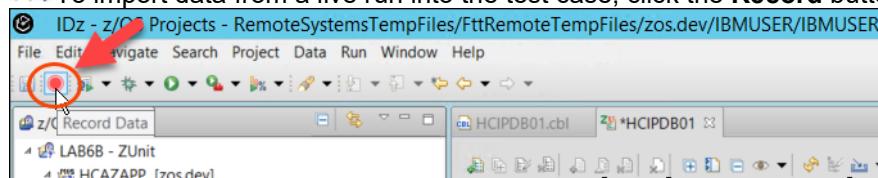
If you need to use the clear or reset function right mouse click and select as below



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

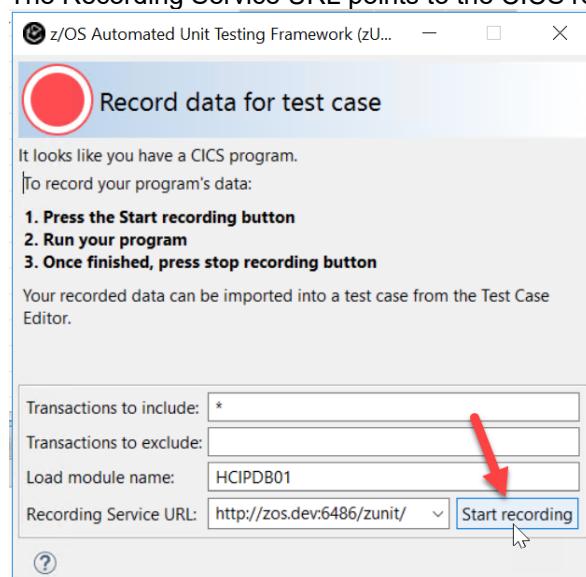
- 3.2.10 ► Minimize the terminal emulator and go back to IDz dialog.

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

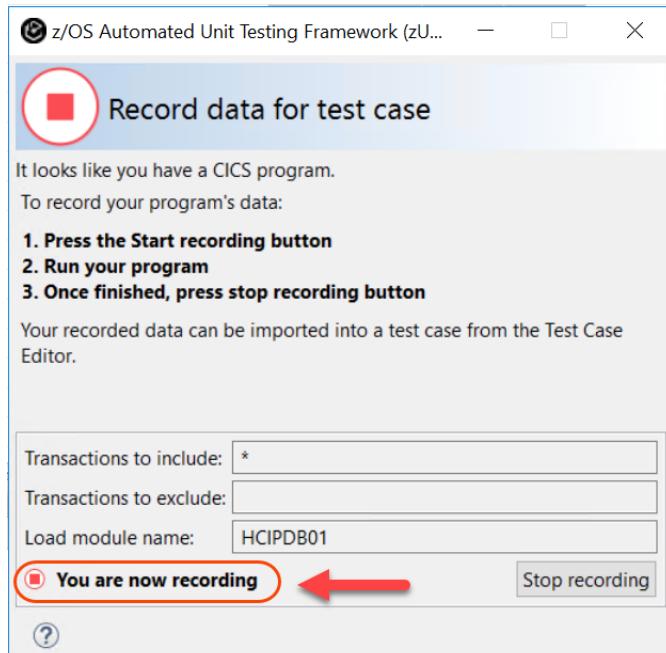


- 3.2.11 ► In the dialog that comes up, click on **Start recording**.

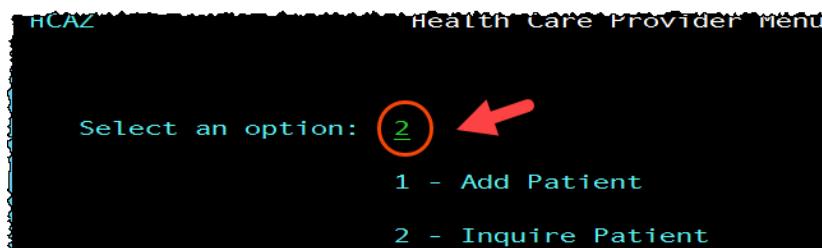
The Recording Service URL points to the CICS region where the live run is recorded.



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



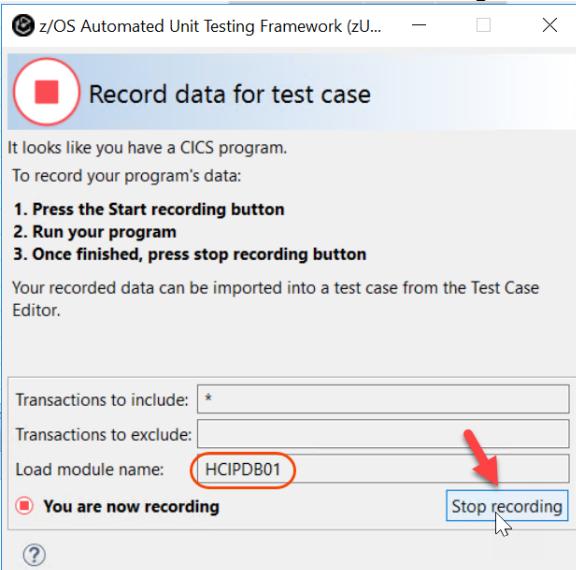
3.2.13 ► Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



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

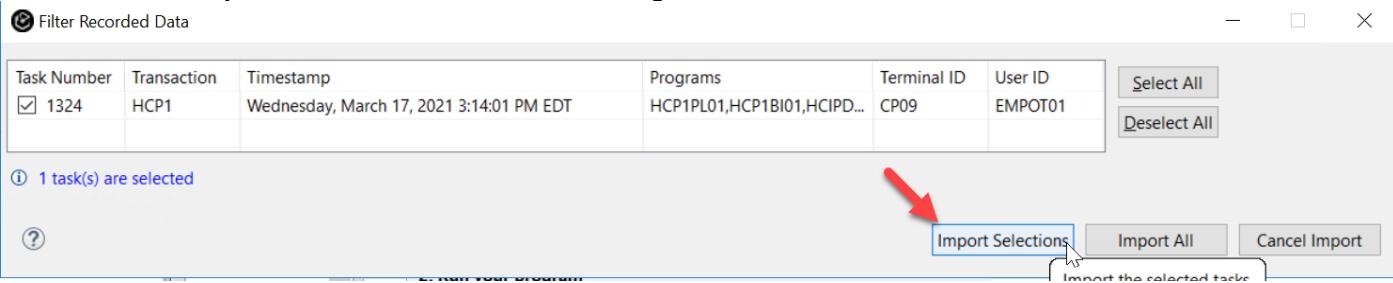


3.2.15 ►| Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



The dialog Filter Recorded Data is displayed.

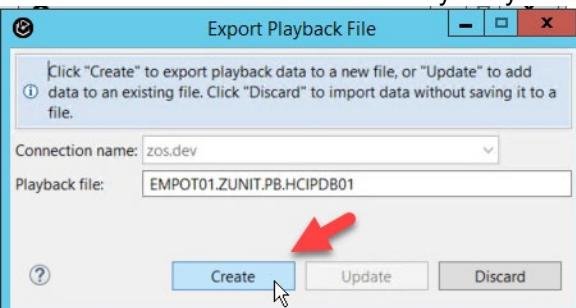
3.2.16 ►| Click **Import Selections** to dismiss the dialog.



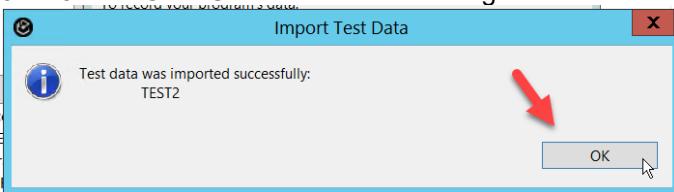
3.2.17 ►| Change the *Playback file* to **EMPOT01.ZUNIT.PB.HCIPDB01** (instead of **EMPOT05**)

►| 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.

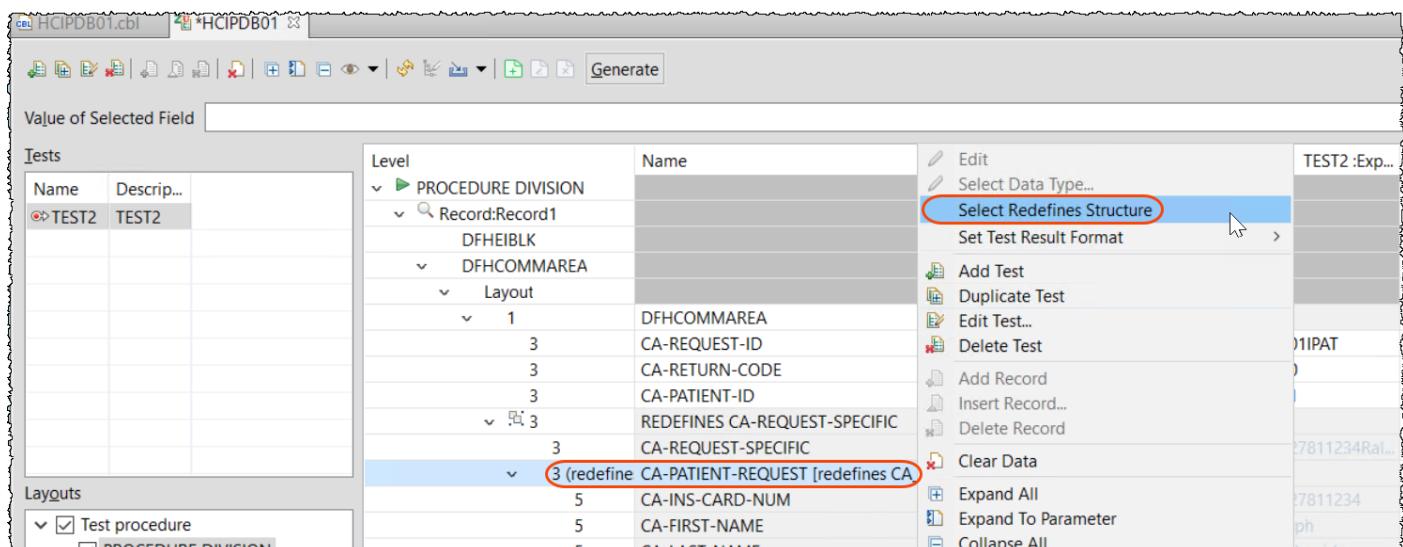


3.2.20 You now see a new test created in the editor and populated with the values from the live run. Also you may notice that this program has many redefines.

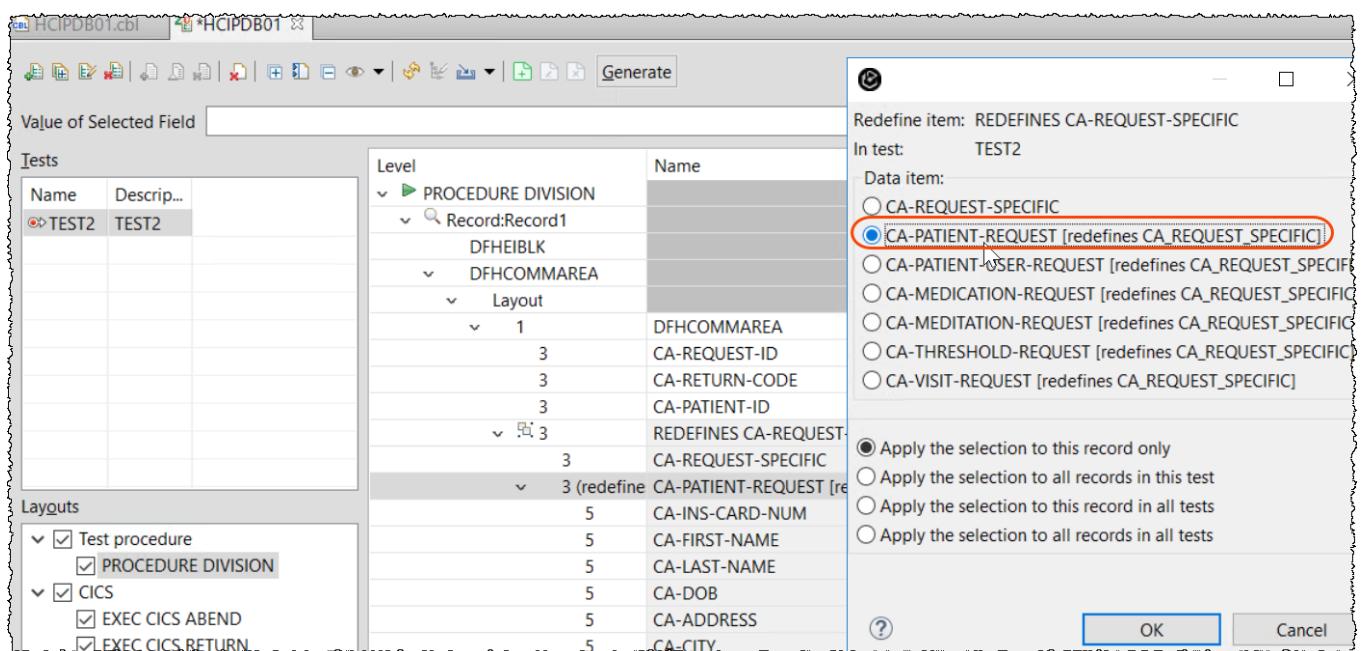
You need to identify which is the area being used on this program execution. In our example is the “CA-PATIENT-REQUEST” area being redefined.

► 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.

Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
PROCEDURE DIVISION					
Record:Record1	DFHEIBLK				
DFHCOMMAREA					
Layout					
1	DFHCOMMAREA				
3	CA-REQUEST-ID	X(6)	DISPLAY	01PAT	<input type="button" value="01PAT"/>
3	CA-RETURN-CODE	9(2)	DISPLAY	0	<input type="button" value="0"/>
3	CA-PATIENT-ID	9(10)	DISPLAY	1	<input type="button" value="1"/>
3	REDEFINES CA-REQUEST-SPECIFIC	X(3248...)	DISPLAY		
3 (redefine)	CA-PATIENT-REQUEST [redefines CA_QUE...				
5	CA-INS-CARD-NUM	X(10)	DISPLAY		
5	CA-FIRST-NAME	X(10)	DISPLAY		
5	CA-LAST-NAME	X(20)	DISPLAY		
5	CA-DOB	X(10)	DISPLAY		
5	CA-ADDRESS	X(20)	DISPLAY		
5	CA-CITY	X(20)	DISPLAY		
5	CA-POSTCODE	X(10)	DISPLAY		
5	CA-PHONE-MOBILE	X(20)	DISPLAY		
5	CA-EMAIL-ADDRESS	X(50)	DISPLAY		
5	CA-USERID	X(10)	DISPLAY		

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

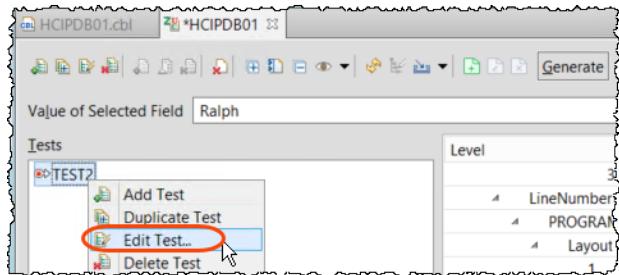
3.2.24 You now see a new test created in the editor and populated with the values from the live run.

► 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 ”

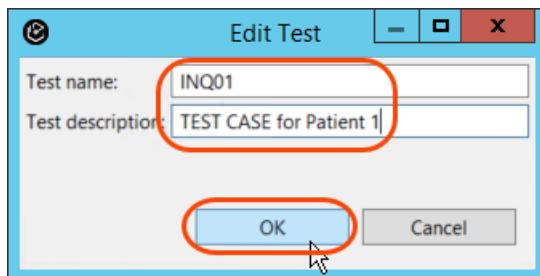
Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
1	HCAZERRS	X(8)	DISPLAY		
COMMAREA					
Layout					
1	CA-ERROR-MSG				
3	FILLER	X(9)	DISPLAY		
3	CA-DATA	X(90)	DISPLAY		
EXEC SQL SELECT INTO [PATI					
Record:Record1					
LineNumber=117					
INTO					
5	CA-FIRST-NAME	X(10)	DISPLAY	Ralph	<input type="button" value="Ralph"/>
5	CA-LAST-NAME	X(20)	DISPLAY	DALmeida	<input type="button" value="DALmeida"/>
5	CA-DOB	X(10)	DISPLAY	1980-07-11	<input type="button" value="1980-07-11"/>
5	CA-INS-CARD-...	X(10)	DISPLAY	9627811234	<input type="button" value="9627811234"/>
5	CA-ADDRESS	X(20)	DISPLAY	34 Main Street ...	<input type="button" value="34 Main Street ..."/>
5	CA-CITY	X(20)	DISPLAY	Toronto	<input type="button" value="Toronto"/>
5	CA-POSTCODE	X(10)	DISPLAY	M5H 1T1	<input type="button" value="M5H 1T1"/>
5	CA-PHONE-M...	X(20)	DISPLAY	077-123-9987 ...	<input type="button" value="077-123-9987 ..."/>
5	CA-EMAIL-AD...	X(50)	DISPLAY	RalphD@ibm.c...	<input type="button" value="RalphD@ibm.c..."/>
5	CA-USERID	X(10)	DISPLAY	ralphd	<input type="button" value="ralphd"/>
WHERE					
3	DB2-PATIENT-ID	S9(9)	BINARY	1	
SQLCA					

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.

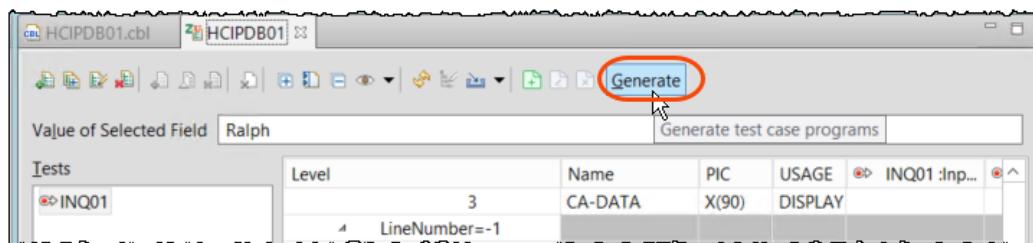


Section 4. Generate, build and run the unit test.

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

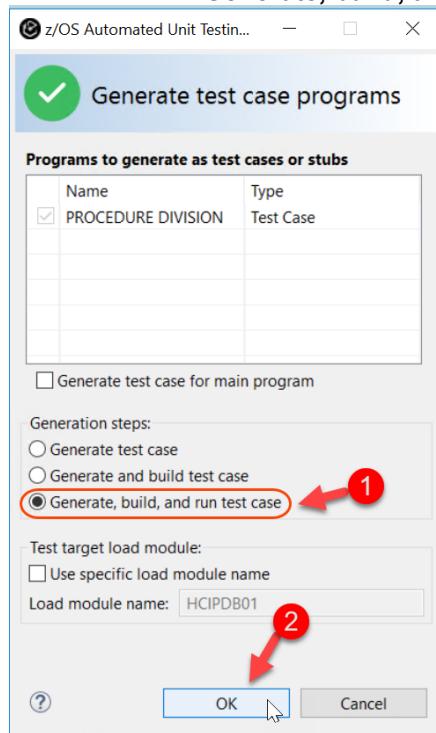
4.1 Generating, building and running the test case program.

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.



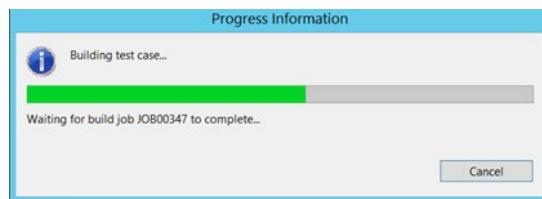
On the **Generate test case programs** dialog,

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

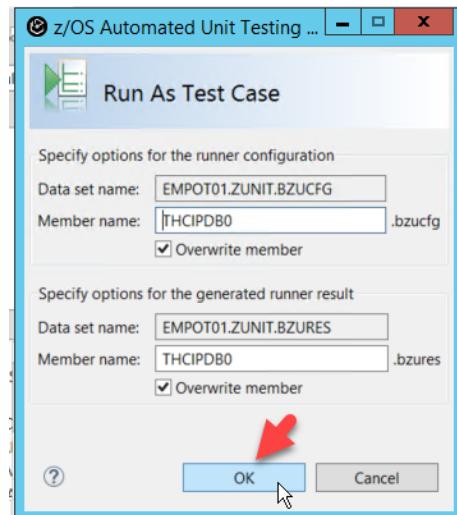


A **Progress information** dialog on the building will open.

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.



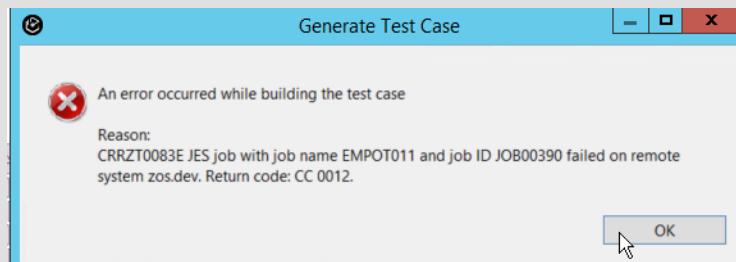
4.1.3 ► Once the building is complete, a **Run As Test Case** dialog will open, click **OK** to continue.



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*

IDz_review - C:\Workspaces\IDz15\IDz_review - IBM Developer for z/OS
File Edit Navigate Search Project Run Window Help

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

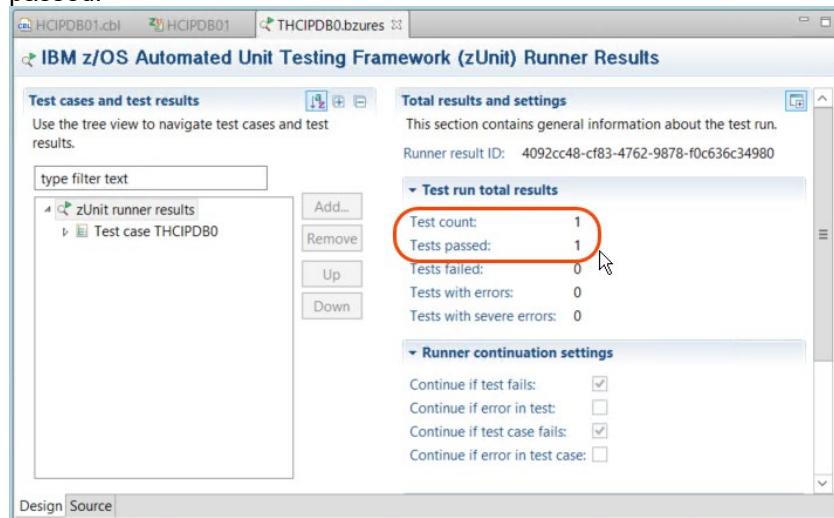
#2. Had a “strange Test fail” ?

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.

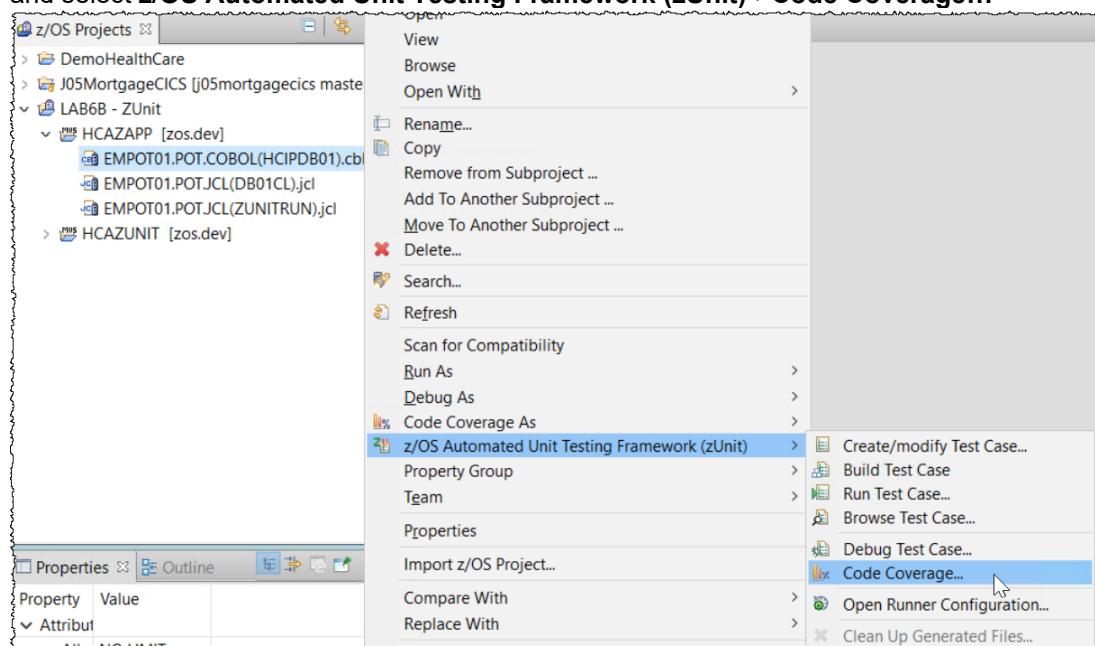
- 4.1.6 ►| Use **Ctrl + Shift + F4** to close all opened editors.

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.

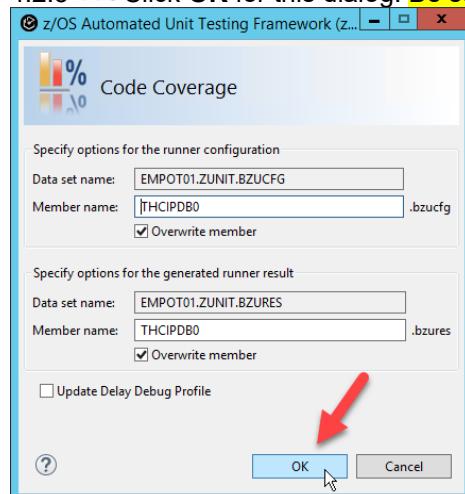
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

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



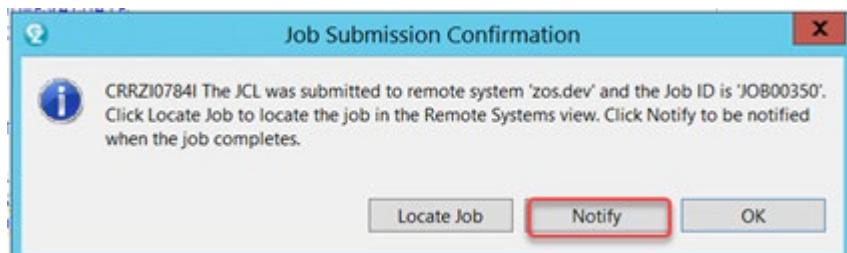
- | If you get a message that files are updated click **Yes** to continue

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

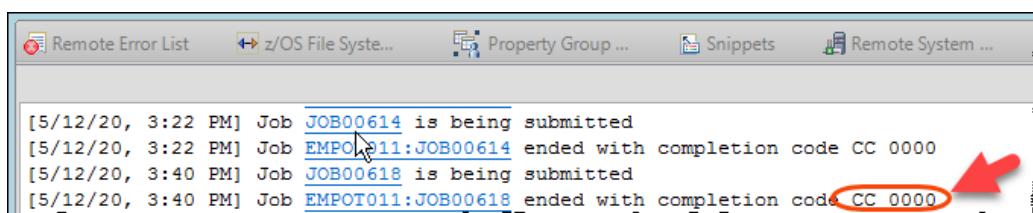


4.2.4 A Job Is submitted for batch execution..

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

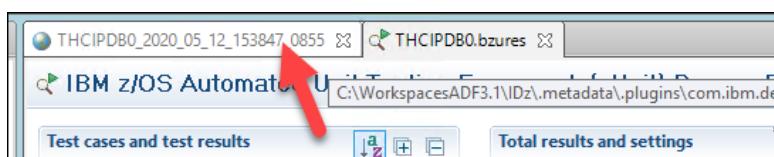


- Make sure the job completes with completion code of **0**.



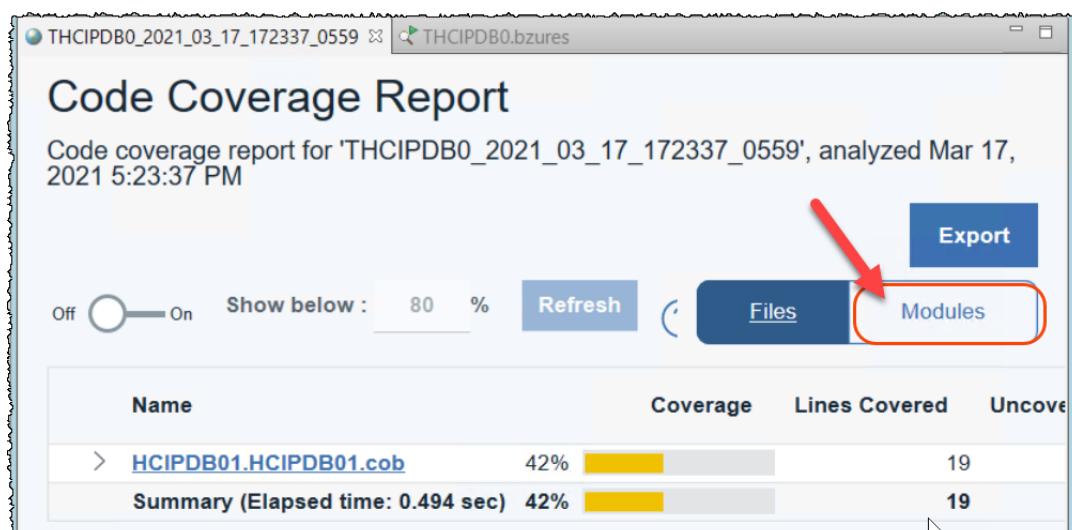
4.2.5 Once again the zUnit Results are displayed and the Test passed.

- 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**:

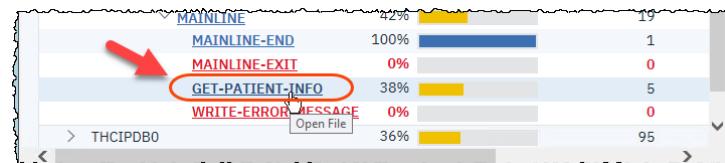


4.2.7 ► Expand the nodes as below to see the COBOL paragraphs.

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.

This screenshot shows a COBOL source code editor with the file `HCIPDB01.HCIPDB01.cob` open. The code is displayed in a scrollable window. A yellow speech bubble labeled "lines covered" points to a green vertical bar on the left margin, indicating coverage for lines 247 through 255. A red circle highlights line 259, which is part of a conditional block. A red arrow points to line 262, which is part of another conditional block. A red box highlights the text "Lines 259-266 not covered." The code itself includes various COBOL statements like `MOVE`, `EVALUATE`, and `PERFORM`.

```

247      :CA-ADDRESS,
248      :CA-CITY,
249      :CA-POSTCODE,
250      :CA-PHONE-MOBILE,
251      :CA-EMAIL-ADDRESS,
252      :CA-USERID
253      FROM PATIENT
254      WHERE PATIENTID = :DB2-PATIENT-ID
255      END-EXEC.
256      Evaluate SQLCODE
257      When 0
258          MOVE '00' TO CA-RETURN-CODE
259          Lines 259-266 not covered.
260          MOVE '01' TO CA-RETURN-CODE
261      When -913
262          MOVE '01' TO CA-RETURN-CODE
263      When Other
264          MOVE '90' TO CA-RETURN-CODE
265          PERFORM WRITE-ERROR-MESSAGE
266          EXEC CICS RETURN END-EXEC
267      END-Evaluate.
268      * #bug -- the line below will introduce a BUG

```

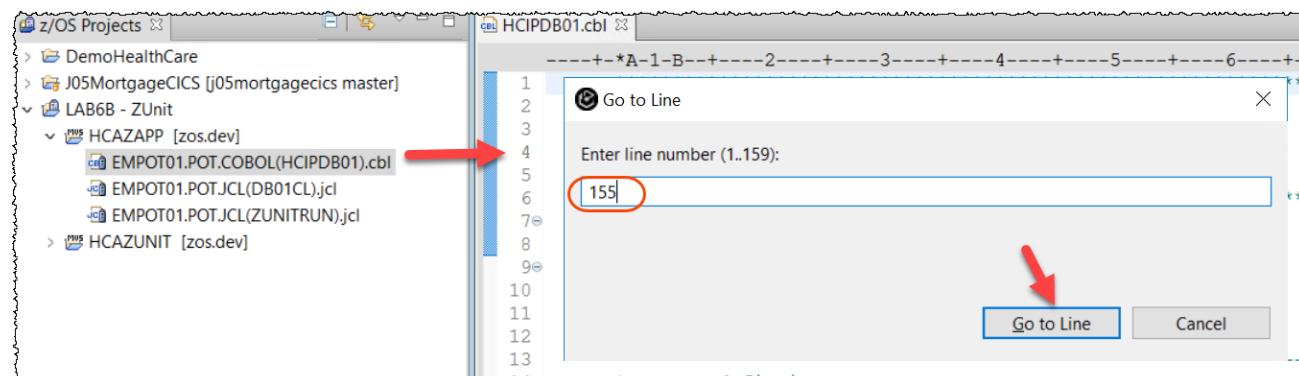
4.2.10 From this report we can see that the test cases created for this program are not sufficient .. The developer could go back to the test cases editor (step 3.2.19 above) and manually add test cases that cover other paragraphs. Due the lack of time we will not cover that here, but if you have extra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

```

-----+*A-1-B---+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
138      FROM PATIENT
139      WHERE PATIENTID = :DB2-PATIENT-ID
140      END-EXEC.
141      Evaluate SQLCODE
142      When 0
143          MOVE '00' TO CA-RETURN-CODE
144      When 100
145          MOVE '01' TO CA-RETURN-CODE
146      When -913
147          MOVE '01' TO CA-RETURN-CODE
148      When Other
149          MOVE '90' TO CA-RETURN-CODE
150          PERFORM WRITE-ERROR-MESSAGE
151          EXEC CICS RETURN END-EXEC
152      END-Evaluate.
153      * %bug -- the line below will introduce a BUG
154      *
155      MOVE "BAD NAME" to CA-FIRST-NAME
156      *
157      EXIT.
158      *

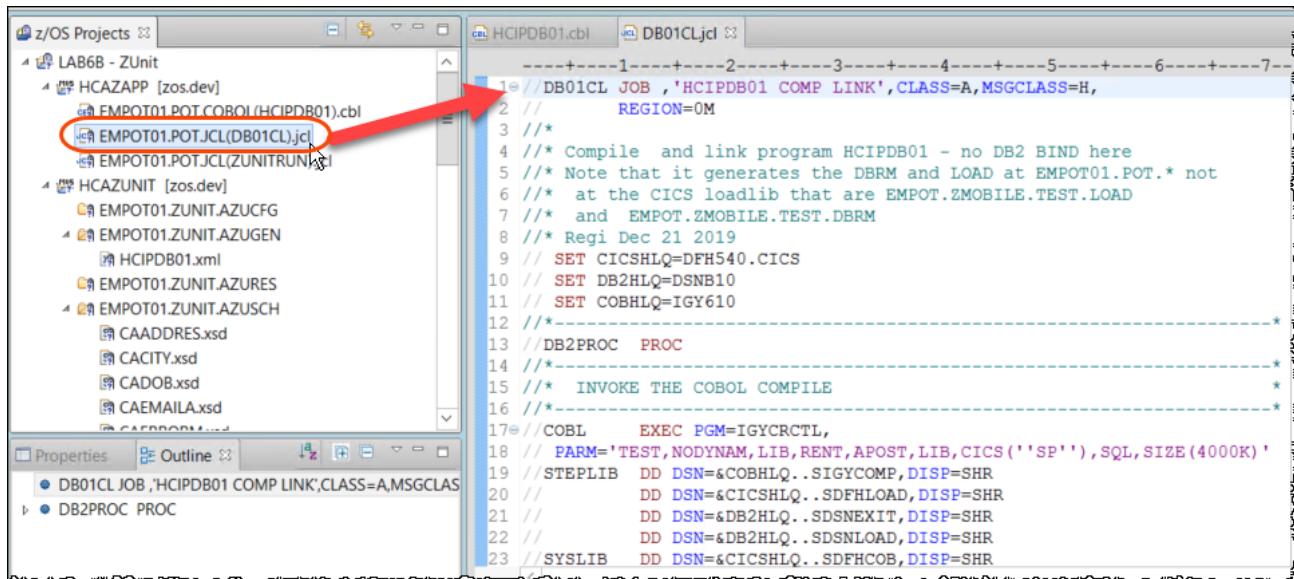
```

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.



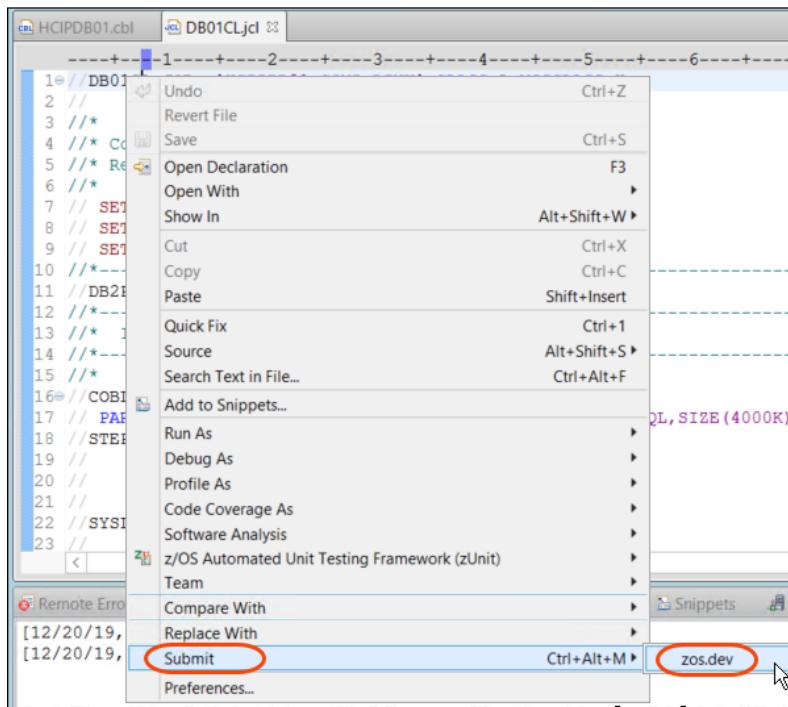
The screenshot shows the IBM Rational Developer for z/OS interface. On the left, the 'z/OS Projects' view displays several project nodes, including LAB6B - ZUNIT, HCAZAPP [zos.dev], and HCAZUNIT [zos.dev]. A red arrow points from the 'z/OS Projects' view to the 'EMPOT01.POT.JCL (DB01CL).jcl' file in the code editor window on the right. The code editor window displays the JCL code for the DB01CL job, which includes compilation and linking steps for the HCIPDB01 program.

```

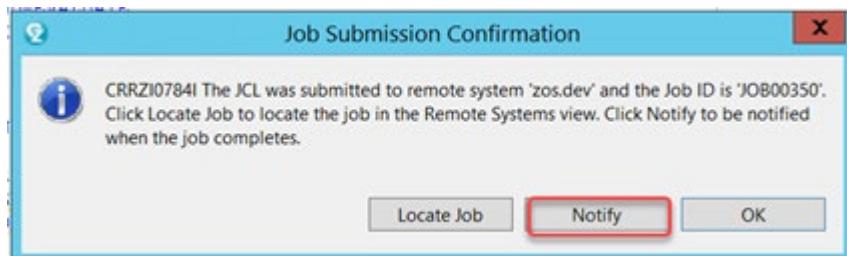
1 //DB01CL JOB , 'HCIPDB01 COMP LINK', CLASS=A, MSGCLASS=H,
2 // REGION=0M
3 /**
4 //** Compile and link program HCIPDB01 - no DB2 BIND here
5 //** Note that it generates the DBRM and LOAD at EMPOT01.POT.* not
6 //** at the CICS loadlib that are EMPOT.ZMOBILE.TEST.LOAD
7 //** and EMPOT.ZMOBILE.TEST.DBRM
8 //** Regi Dec 21 2019
9 // SET CICSHLQ=DFH540.CICS
10 // SET DB2HLQ=DSNB10
11 // SET COBHLQ=IGY610
12 /**
13 //DB2PROC PROC
14 /**
15 /**
16 /**
17 //COBL EXEC PGM=IGYCRCTL,
18 // PARM='TEST,NODYNAM,LIB,RENT,APOST,LIB,CICS(''SP''),SQL,SIZE(4000K)'
19 //STEPLIB DD DSN=&COBHLQ..SIGYCOMP,DISP=SHR
20 // DD DSN=&CICSHLQ..SDFHLOAD,DISP=SHR
21 // DD DSN=&DB2HLQ..SDSNEXIT,DISP=SHR
22 // DD DSN=&DB2HLQ..SDSNLOAD,DISP=SHR
23 //SYSLIB DD DSN=&CICSHLQ..SDFHCOB,DISP=SHR

```

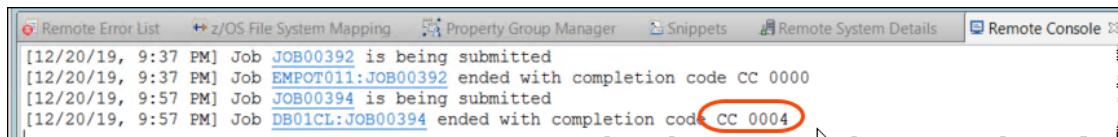
- 5.2.2 ► Right click and select **submit > zos.dev** to submit this job for execution



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

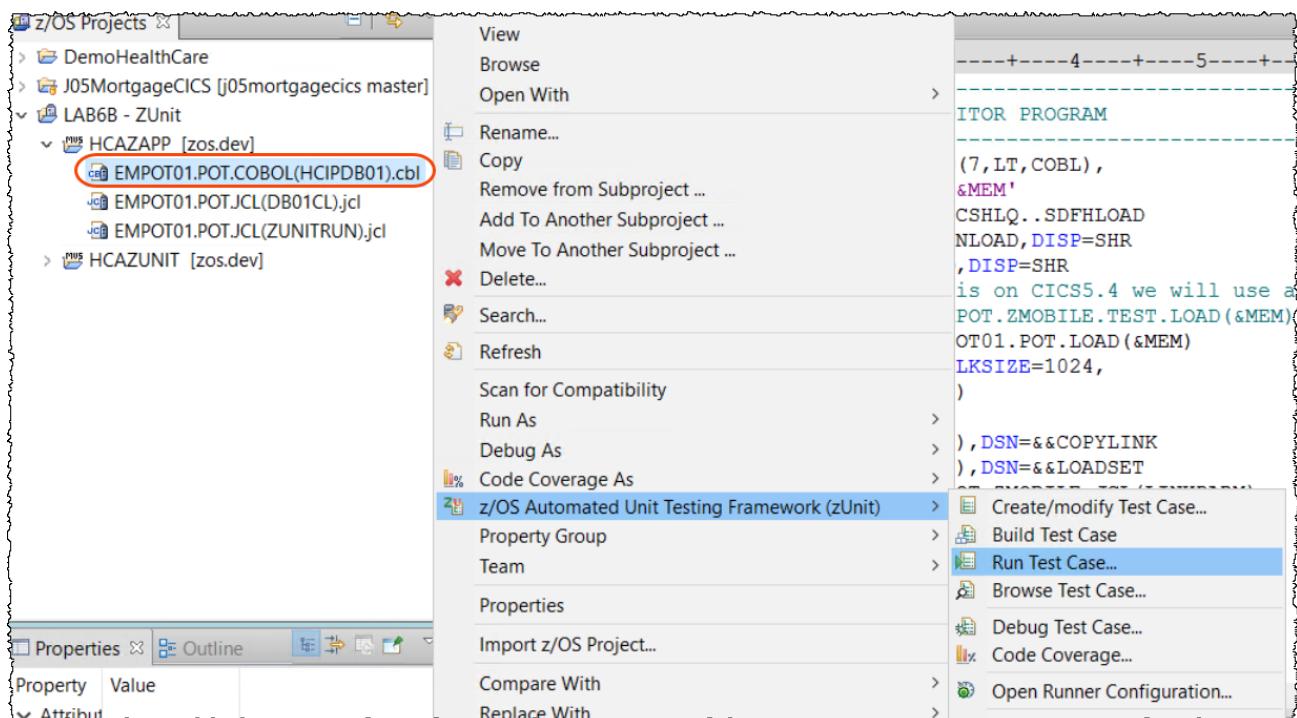


Make sure the job completes with completion code of 4.



5.3 Rerunning the test case

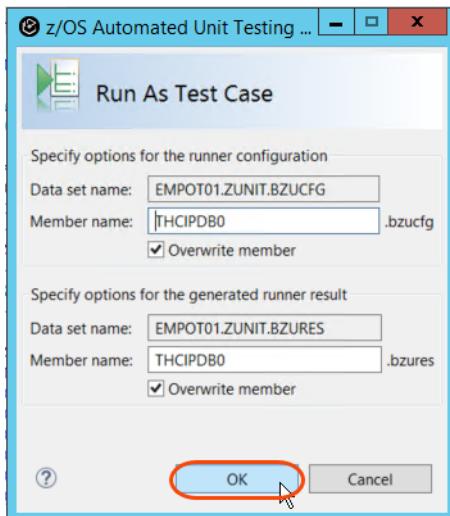
5.3.1 ➡ On the z/OS Projects view, select **EMPOT01.POT.COBOL(HCIPDB01).cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..** action.



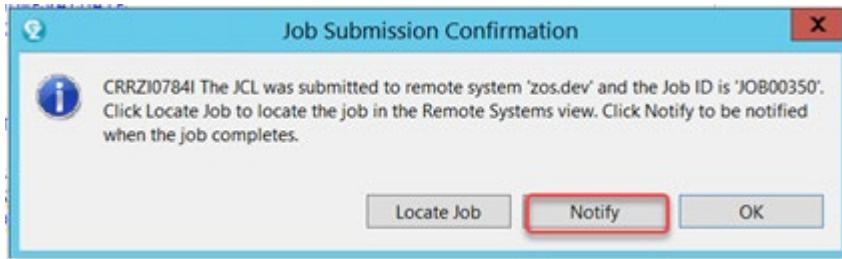
5.3.2 ➡ Click **Yes** to the following dialog to continue running the test case.



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



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

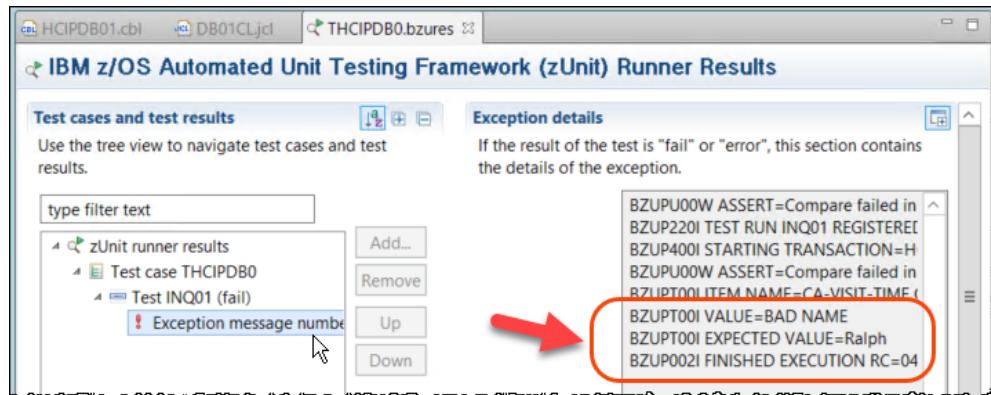


5.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

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

Test case ID	e30e513c-140b-4176-b4f1-0b497de49f44
Module name	THCIPDB0
Test case name	THCIPDB0
Result	fail
Test count	1
Tests passed	0
Tests failed	1
Tests with errors	0
Tests with severe errors	0

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



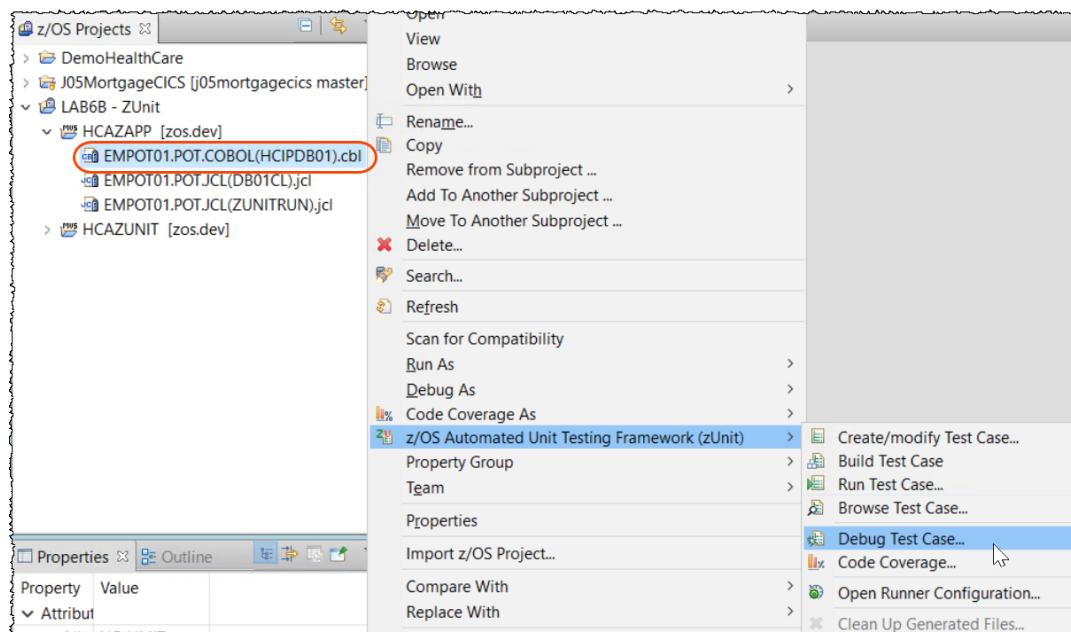
5.4 Running zUnit test case with debugging

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.

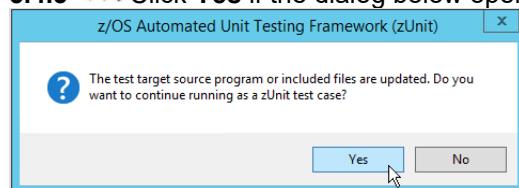
Notice that you are debugging a batch job without need to have CICS and DB2 active, what is very handy.

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

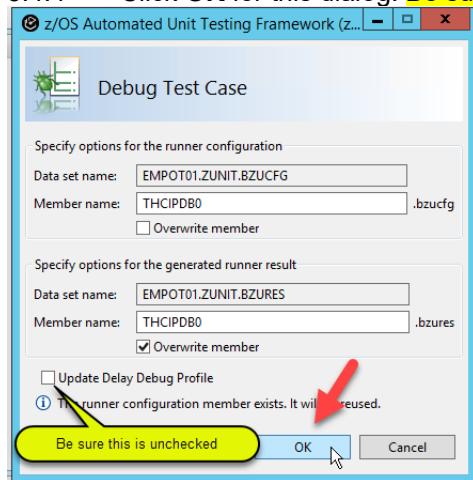
5.4.2 ►| Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > Debug Test Case...**



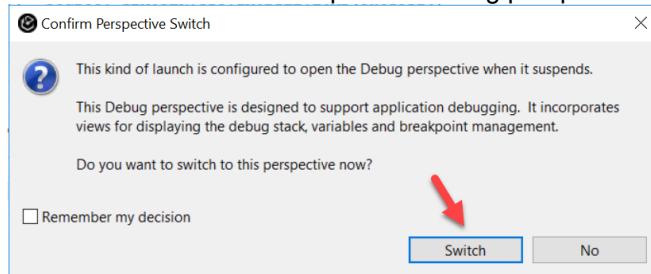
5.4.3 ►| Click **Yes** if the dialog below opens



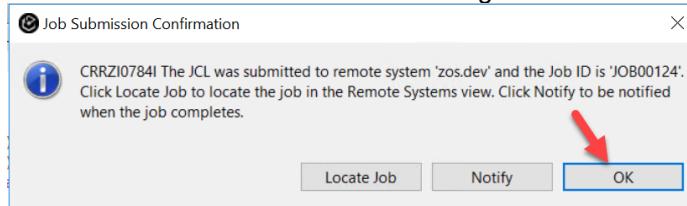
5.4.4 ► Click OK for this dialog. Be sure that Update Delay Debug Profile is un-checked.



5.4.5 ► Click Switch to open the debug perspective.

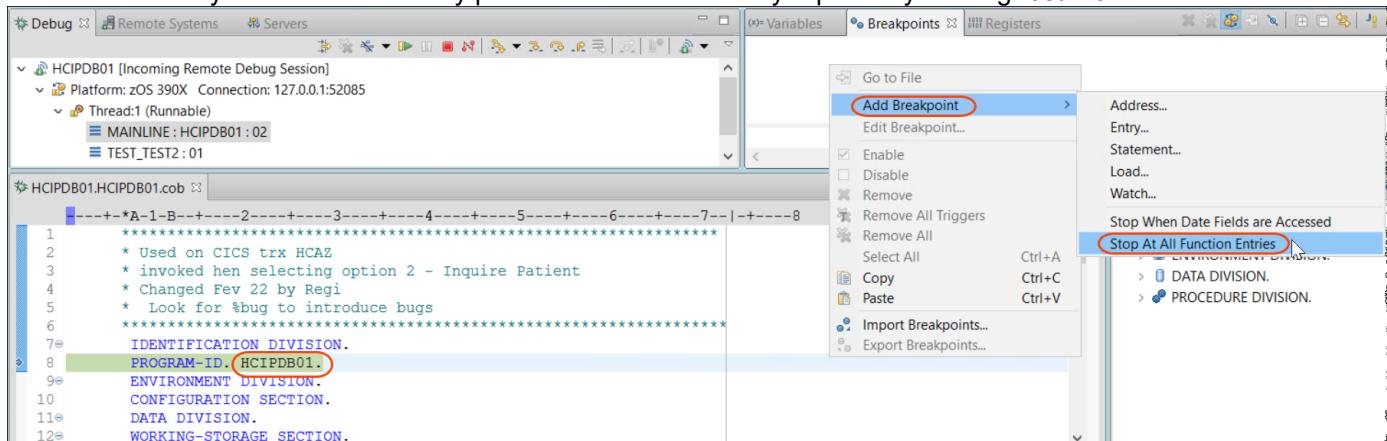


5.4.6 ► Click OK to dismiss this dialog

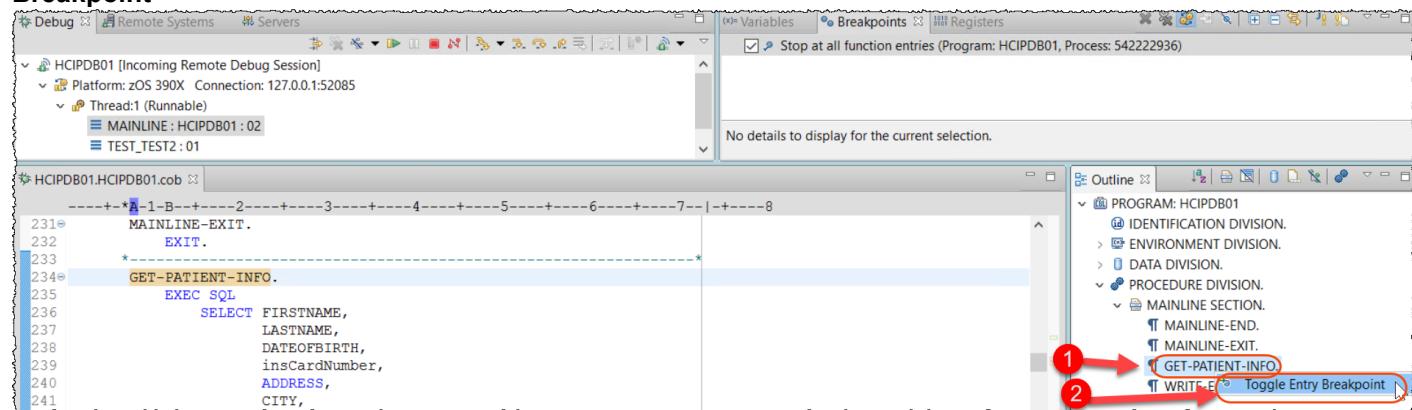


5.4.7 Notice that the first program being debugged is the program that you introduced the bug. (**Hcipdb01**)
The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

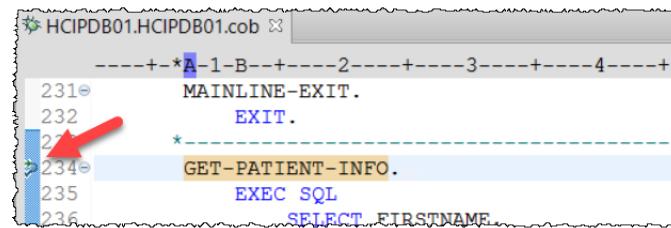
► Using the Breakpoints view, right click and select **Add Breakpoints > Stop At All Functions Entries**. This would allow you to reach the entry point of the source files by repeatedly clicking resume.



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



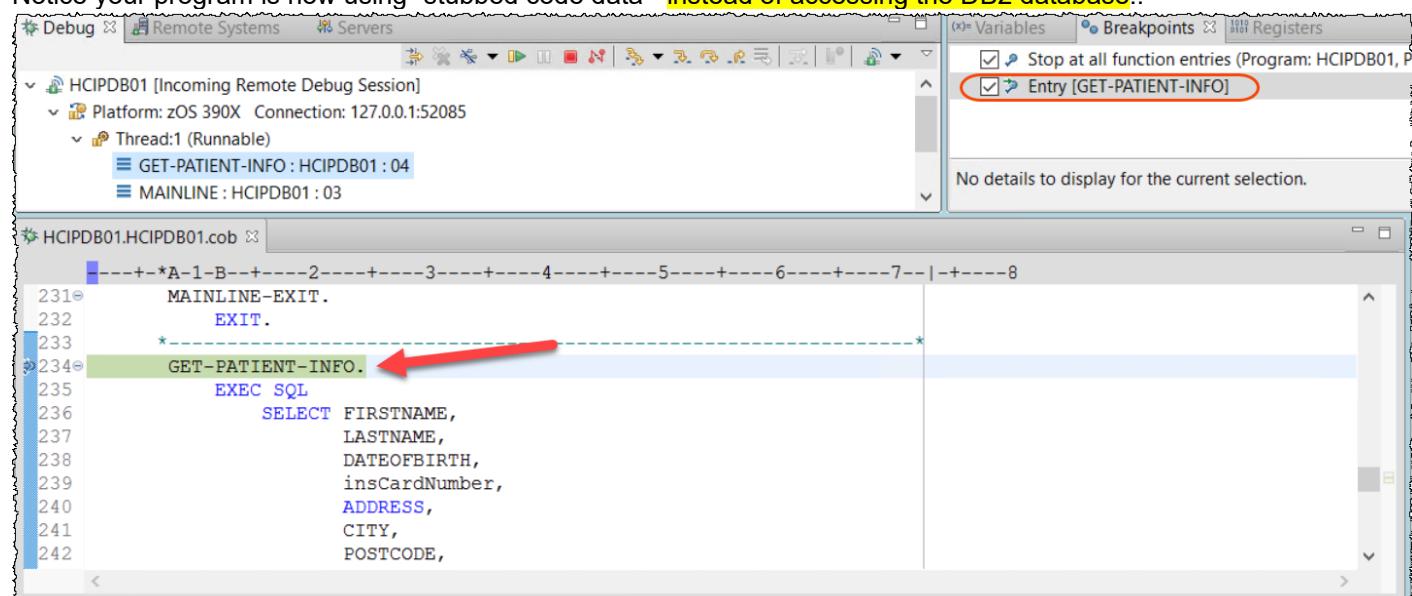
5.4.9 A breakpoint is created on the **EXEC SQL SELECT statement. When the program runs it will stop there.**



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 its contents

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

```

-----+*A-1-B-----2-----3-----4-----5-----6-----7-----8
      FROM PATIENT
      WHERE PATIENTID = :DB2-PATIENT-ID
      END-EXEC.
Evaluate SQLCODE
When 0
  MOVE '00' TO CA-RET
When 100
  MOVE '01' TO CA-RET
When -913
  MOVE '01' TO CA-RET
When Other
  MOVE '90' TO CA-RET

```

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**".

```

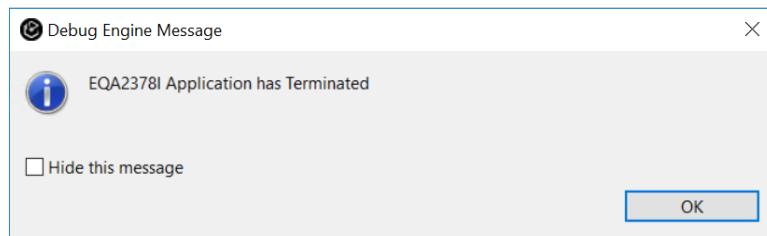
-----+*A-1-B-----2-----3-----4-----5-----6-----7-----8
      END-Evaluate.
      * %bug -- the line below will introduce a BUG
      *
      MOVE "BAD NAME" to CA-FIRST-NAME
      *
      EXIT.
      *
      *COPY HCERRSPD.
      *
      *=====
      * Procedure to write error m
      *   message will include Dat
      *   Medication Id and SQLCOD

```

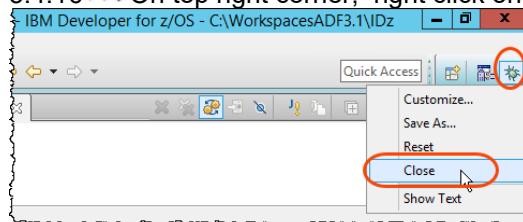
5.4.14 ► Click on icon or press **F8** to resume.. Or if you prefer keep clicking on *Step Into*.



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



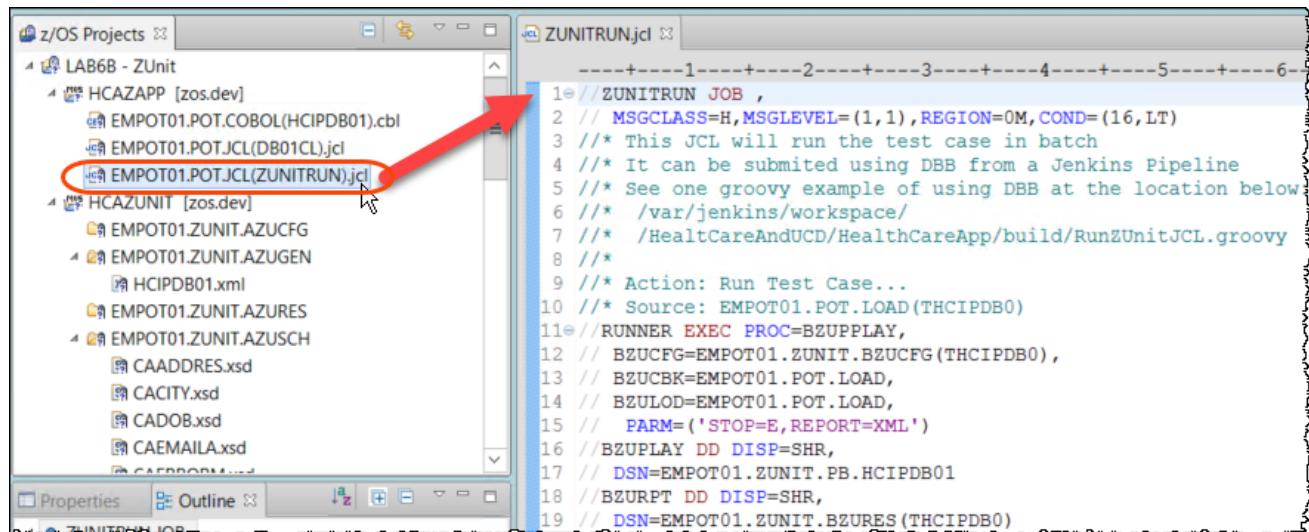
Section 6. Run the unit test from a batch JCL.

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..

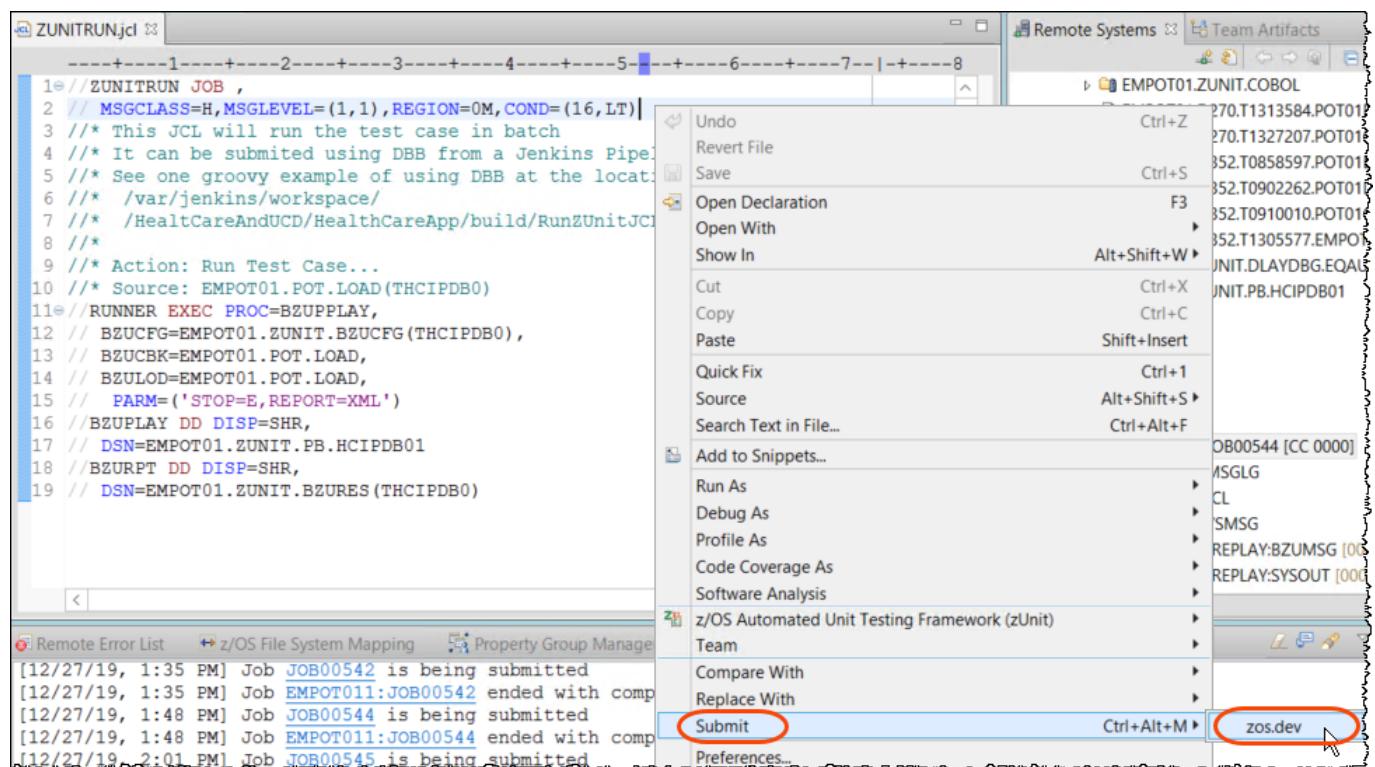
6.1 Running the unit test from a batch JCL

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

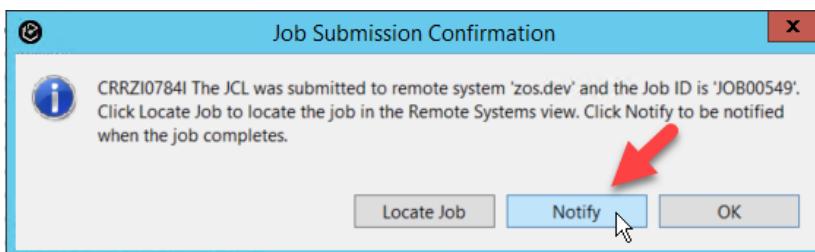
6.1.2 ► On the IDz z/OS Projects view, double click **EMPOT01.POT.JCL(ZUNITRUN).jcl** to open it. This JCL will run the test case that you created using a batch job.



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

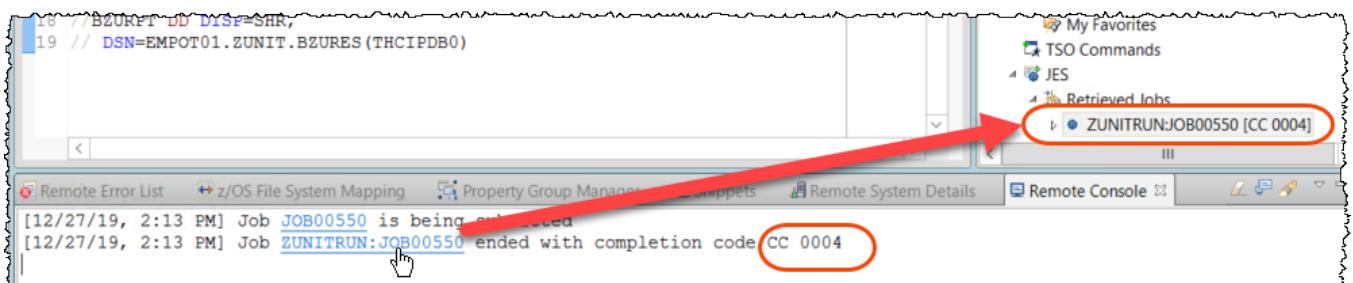


6.1.4 ► Click **Notify** on the Job Submission Confirmation dialog.



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**.

The screenshot shows the Rational Developer for z/OS interface. On the right, the 'Remote Systems' view displays a tree structure under 'zos.dev'. The 'Retrieved Jobs' node is expanded, showing 'ZUNITRUN:JOB00183 [CC 0004]'. This job has several output files listed, with 'RUNNER:REPLAY:SYSOUT' circled in red. A red arrow points from this circled item to the main workspace on the left, which contains a code editor window titled 'RunZUnitJCLgroovy' and a terminal window showing the test results.

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**

The screenshot shows the Rational Developer for z/OS interface. On the right, the 'Remote Systems' view displays a tree structure under 'zos.dev'. The 'z/OS UNIX Files' node is expanded, showing a directory named 'groovy_samples'. Inside this directory, there is a file named 'RunZUnitJCL.groovy', also circled in red. A red arrow points from the 'groovy_samples' node to the code editor window on the left, which contains the Groovy script 'RunZUnitJCLgroovy'. Another red arrow points from the 'RunZUnitJCL.groovy' file within the tree to the same file in the code editor.

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

Congratulations! You have completed the Lab 6B.

LAB 6C – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 program using Local assets (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 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 to 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.

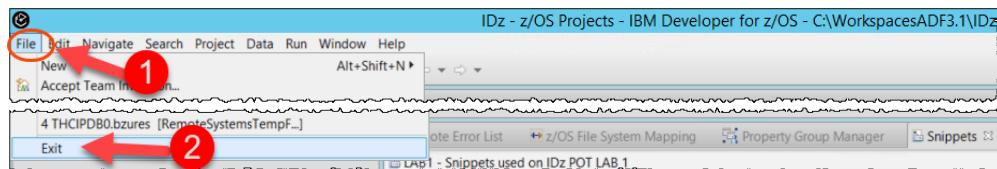
Section 1. Get familiar with the application using the 3270 terminal

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.

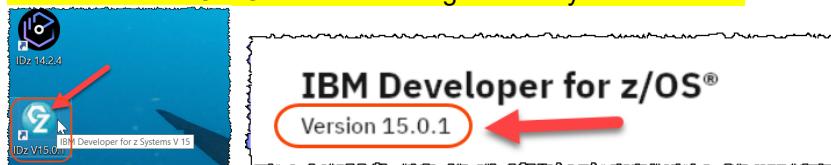
- 1.1.0 This Lab will use **IDz Version 15**, and if it is already running jump to step 1.1.2
- If IDz version 14 used on LAB 1 is running, you must close it using **File** and **Exit**.



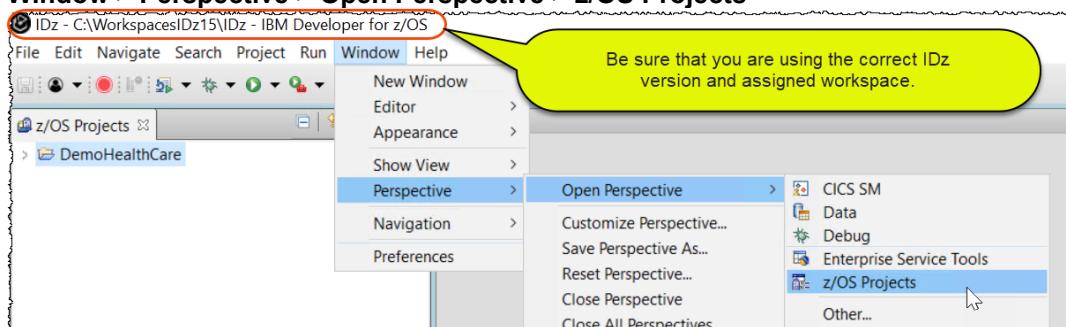
1.1.1 Start IBM Developer for z Systems version 15

- Using the desktop double click on **IDz V15.01** 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.



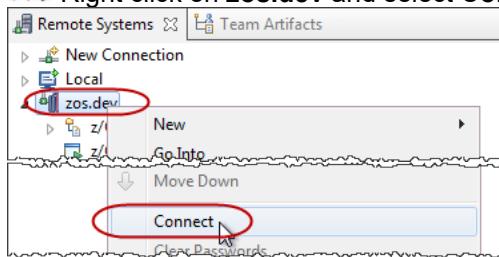
1.1.2 ► Open the z/OS Projects perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



1.1.3 On this lab you will use userid **ibmuser**.

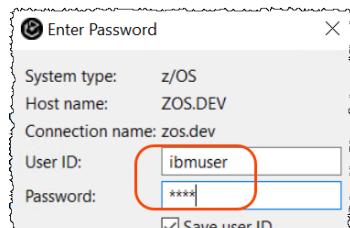
If you are already connected as ibmuser, jump to step 1.1.5 Otherwise disconnect from z/OS and reconnect

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



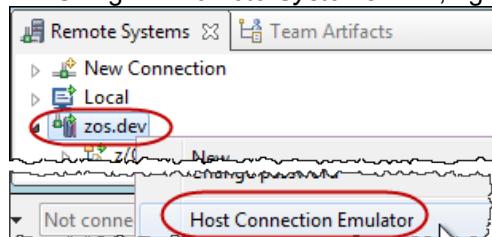
1.1.4 ► Type **ibmuser** as userid and **sys1** as password.

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

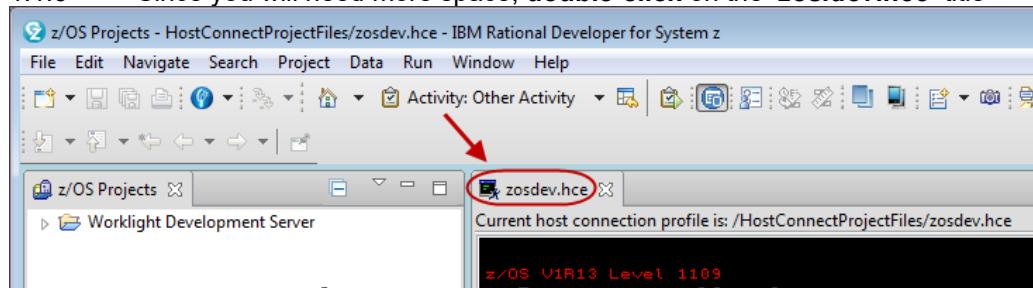


1.1.5 Wait until connection is complete (on the bottom and left until the green bar disappears)

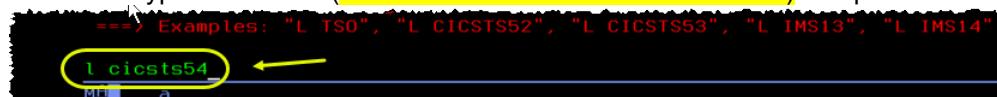
► Using the **Remote Systems** view, right click on **zos.dev** and select **Host Connection Emulator**.



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



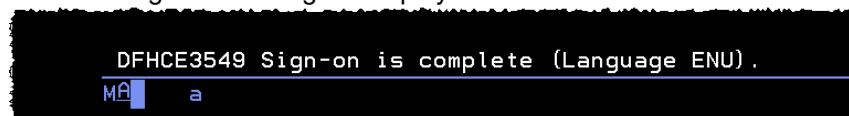
1.1.7 ► Type **I cicsts54**. (where "I" is the lower case of letter "L") and press **Enter key**.



1.1.8 ► Logon using your z/OS user id **ibmuser** and password **sys1** and press **Enter**.



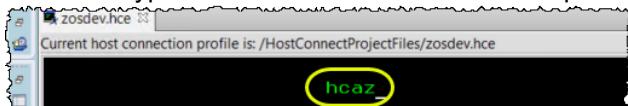
1.1.9 The sign-on message is displayed



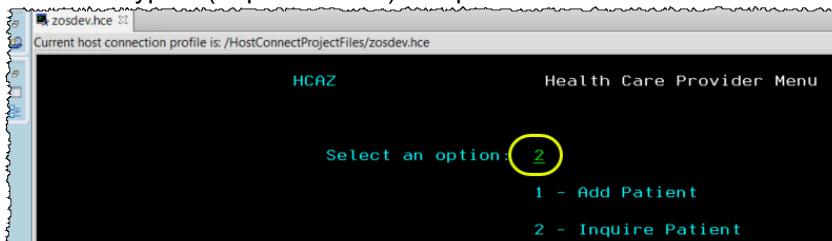
1.2 Run CICS transaction HCAZ

You should now be in the z/OS CICS region named *C/CSTS54*. 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**.



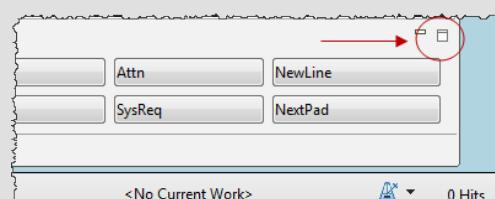
- 1.2.3 ► Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**



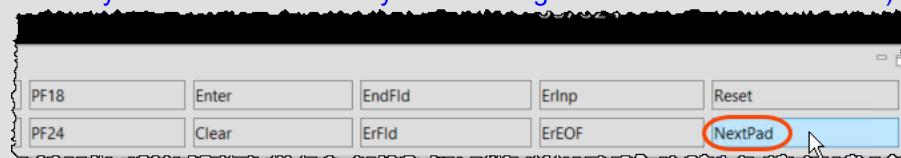
IF you need to use functions like Clear or Reset

If you need to use the *clear* function use the key **Esc**.

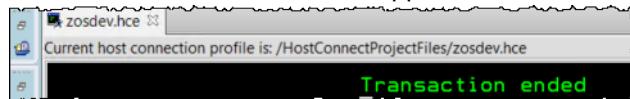
Also you may look in the right lower corner, select this icon . This will display possible keys, including the clear button.



You may also use the Reset key after clicking NextPad

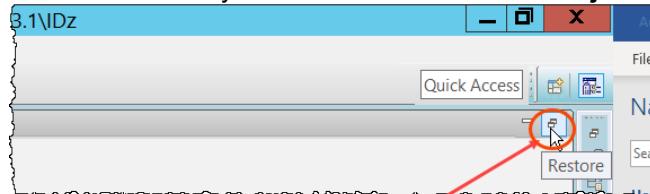


1.2.4 ►| Press **F3** to end the application.



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 Health Care application. The objective here was to show the code that you will update.

Section 2 – Record data interaction using the CICS application.

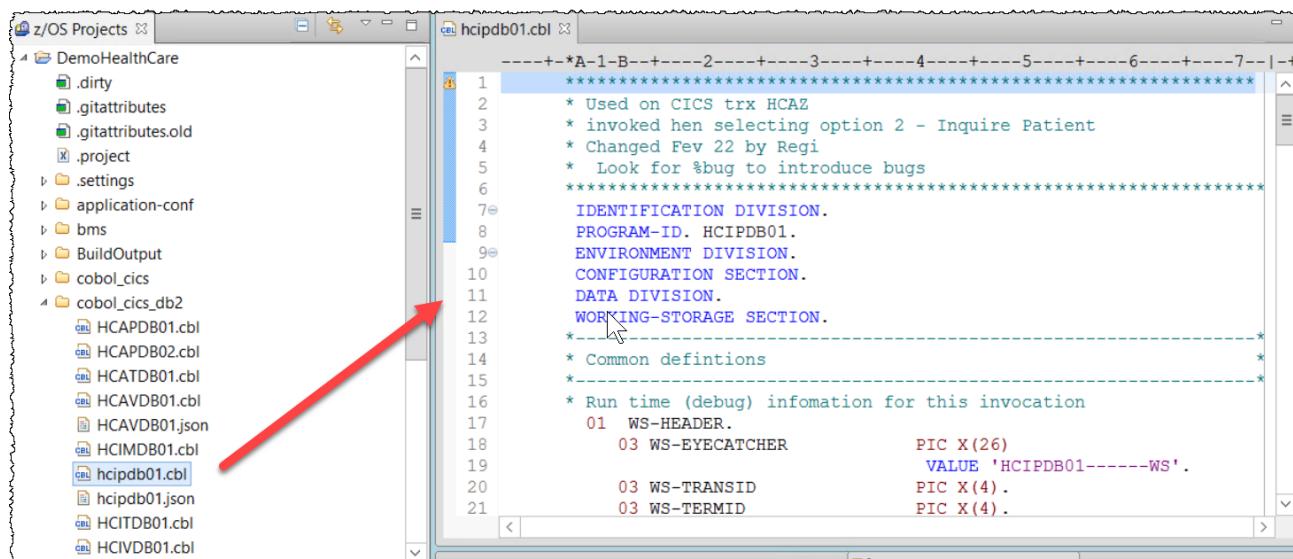
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

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, expand **DemoHealthCare**
double click on **hcipdb01cbl** under **DemoHealthCare/cobol cics db2**

This is the program that you will update later on. The name is lower case to make it easier to find it 😊 .



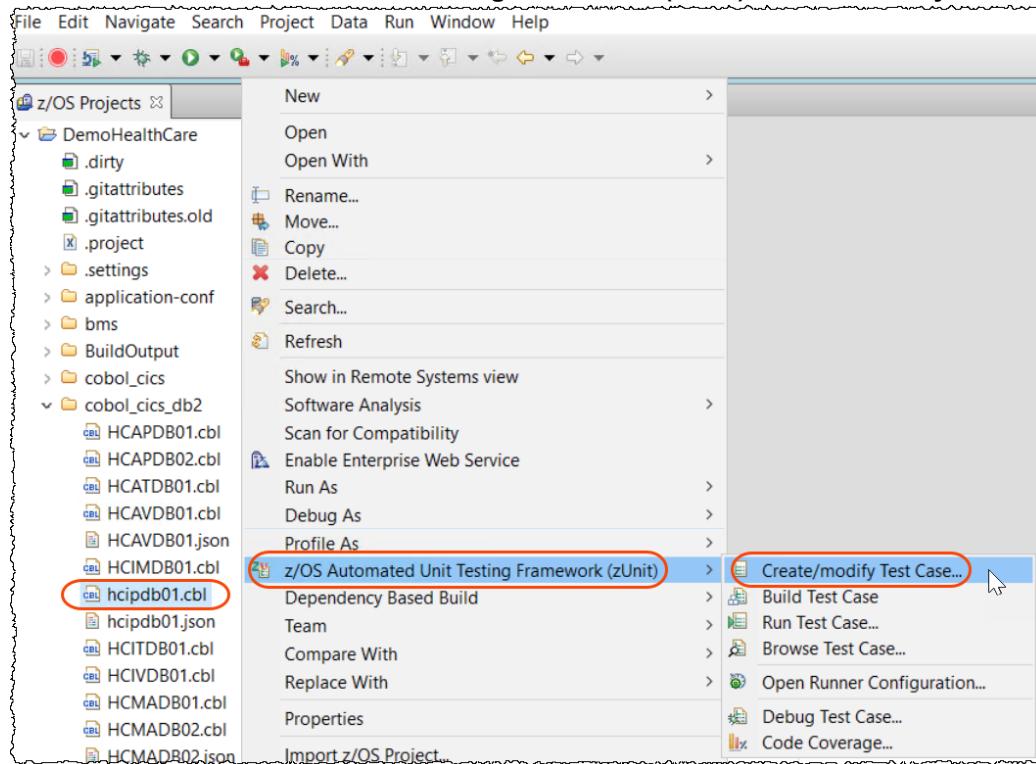
2.1.2 ► Using the Outline view on left expand **PROCEDURE DIVISION** and click on **GET-PATIENT-INFO**. This is where the patient data is read from the DB2 table.
Later on you will introduce a bug in this program..

The screenshot shows the Rational Application Developer interface with the following details:

- Project Explorer (z/OS Projects):** Shows the project "DemoHealthCare" with files like .dirty, .gitattributes, .gitattributes.old, .project, .settings, application-conf, bms, BuildOutput, cobol_cics, cobol_cics_db2, HCAPDB01.cbl, HCAPDB02.cbl, HCATDB01.cbl, HCADVDB01.cbl, and HCADVDB01.
- Properties View:** Displays properties for the selected program "PROGRAM: HCIPDB01". The "PROCEDURE DIVISION" section is expanded, showing:
 - IDENTIFICATION DIVISION.
 - ENVIRONMENT DIVISION.
 - DATA DIVISION.
 - PROCEDURE DIVISION.
 - MAINLINE SECTION.
 - MAINLINE-END.
 - MAINLINE-EXIT.
 - GET-PATIENT-INFO.
- Code Editor (hcipdb01.cbl):** Shows the COBOL code with a red box highlighting the "GET-PATIENT-INFO" section. A red arrow points from the "Properties" view to this section in the code editor.
- Bottom Bar:** Includes tabs for "Remote Error List", "z/OS File System Mapping", and "Property Group M".
- Table (Property Group M):** Shows a list of property groups with columns for Name and Description.

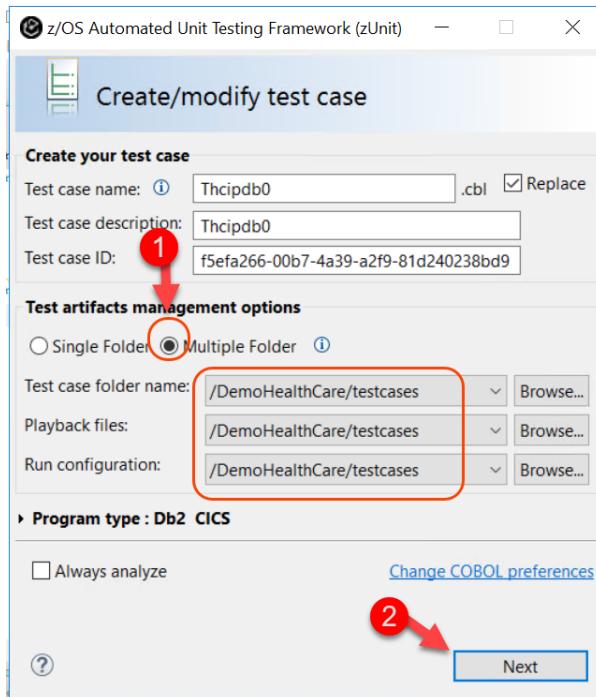
2.2 Recording the COBOL program that sends the message

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



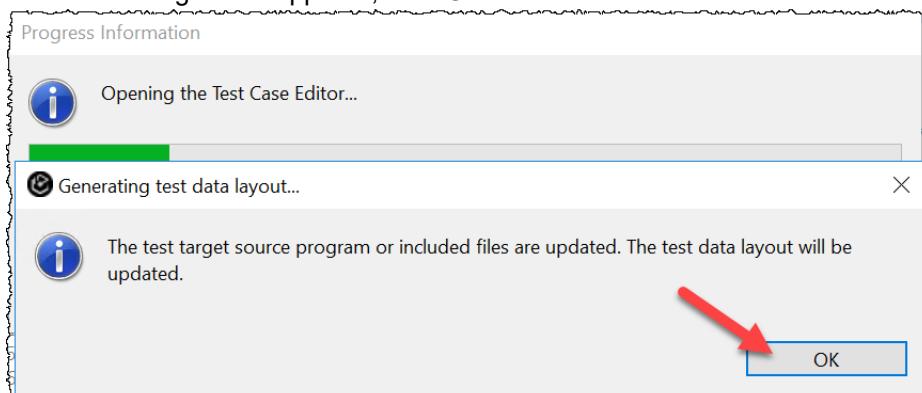
2.2.2 This opens a dialog where you can name your test case.

► Click on **Multiple Folder**, verify that the generated assets will be under **/DemoHealthCare/testcases** and click **Next**.



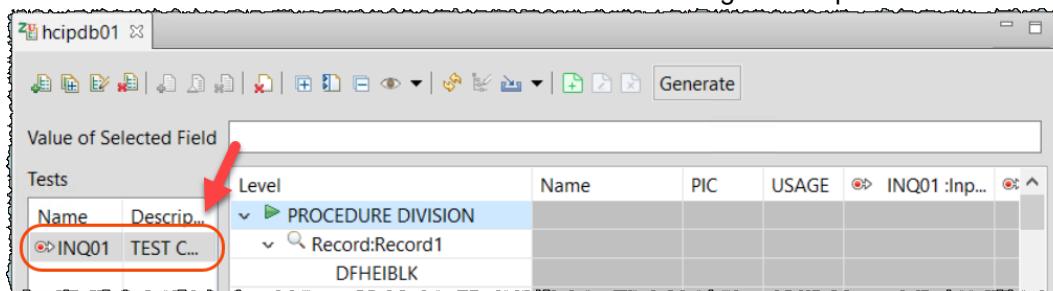
2.2.3 This operation will generate the test data layout on the local workspace.

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



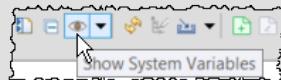
2.2.4 This will open the **Test Case Editor**, as shown below. **INQ01** may or may not be on your screen. If **INQ01** is there you will delete on next step.

The reason that this could be there is because we are reusing a workspace where this was created before.



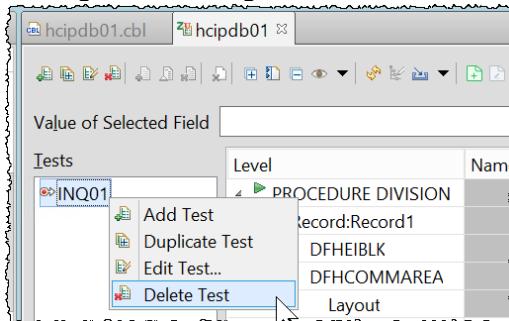
Understanding the test case editor

- 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



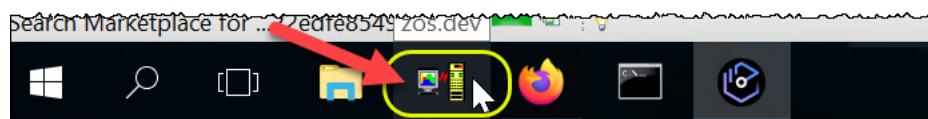
The variables that are returned by the program are the output from the program logic that a developer should be checking

- 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**



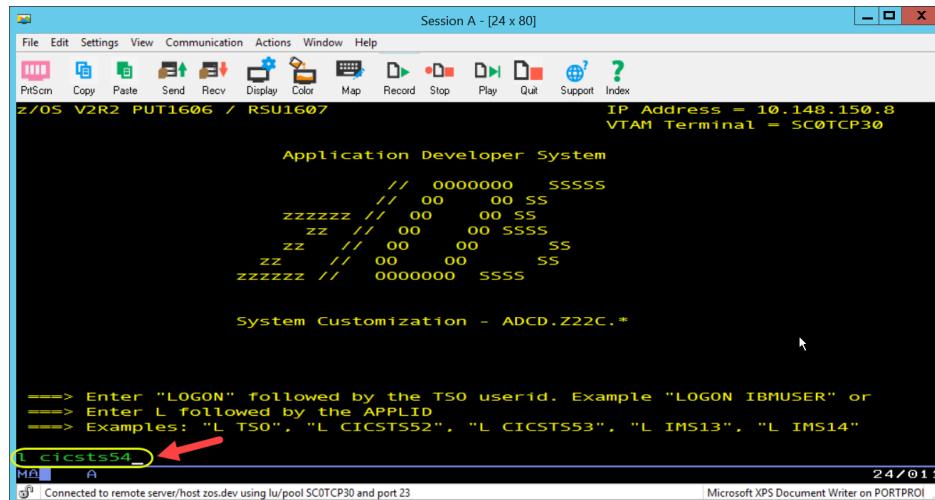
Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

- 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 **I cicsts54**. (where I is L lower case) and press **Enter key**



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

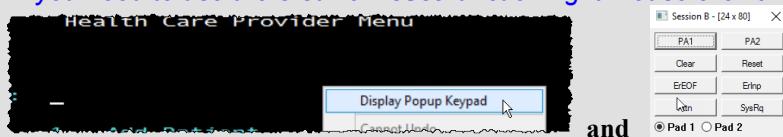


2.2.9 ►| Once you see the sign-on is complete message, enter **hcaz** and press the **Enter** key.



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

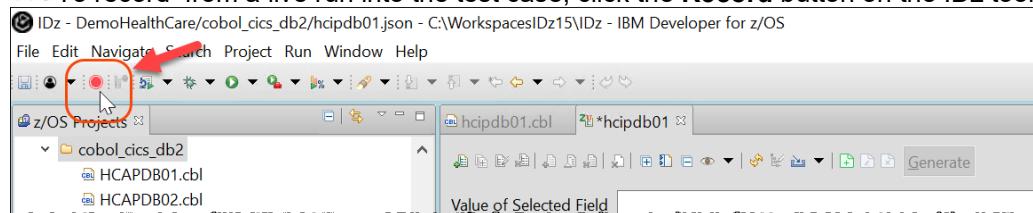


and

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

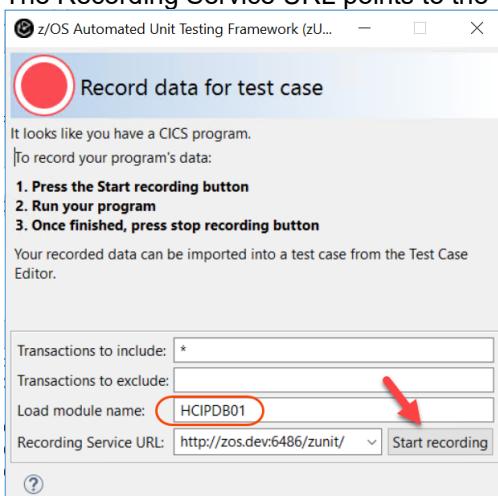
2.2.10 ►| Minimize the terminal emulator and go back to IDz dialog.

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

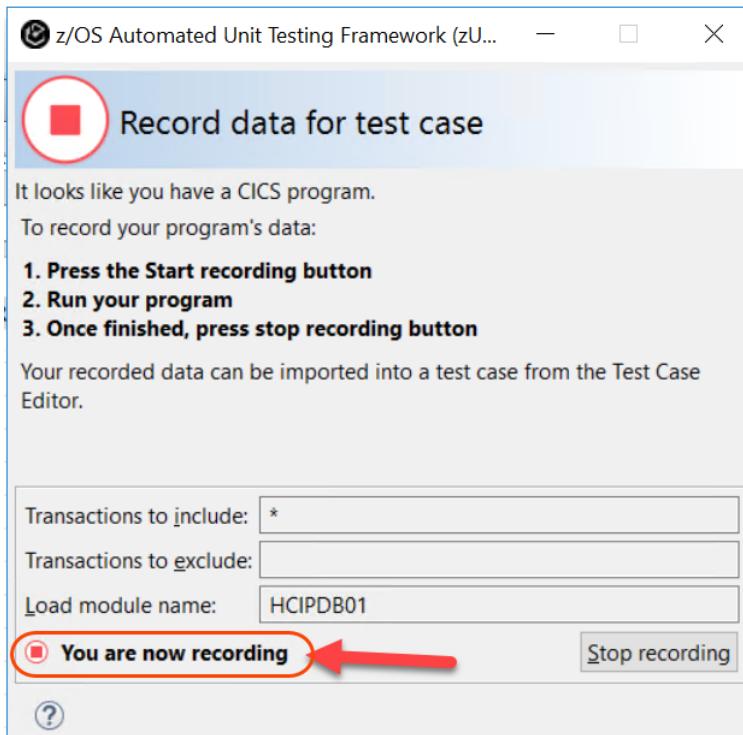


2.2.11 ►| In the dialog that comes up, click on **Start recording**.

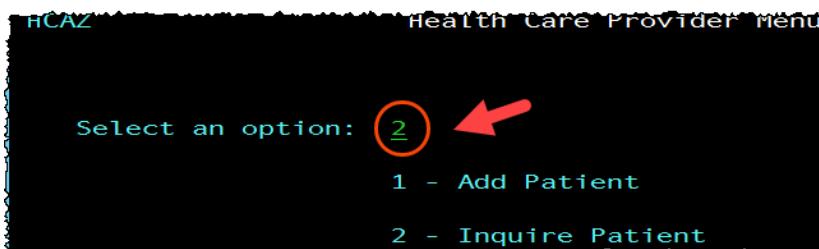
The Recording Service URL points to the CICS region where the live run is recorded.



2.2.12 When recording is turned on, the message “**You are now recording**” appears. Verify that the Load module **Hcipdb01** is the one being recorded.



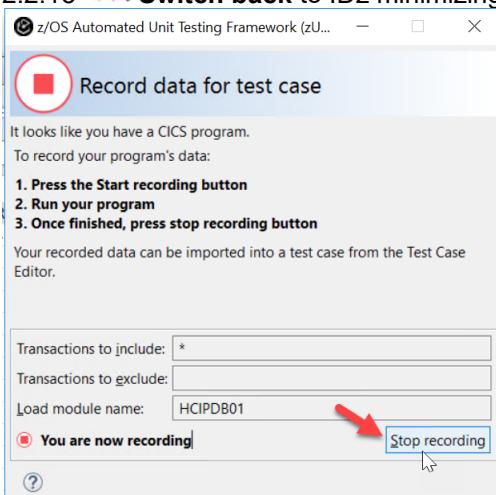
2.2.13 ► Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**. In case your transaction is not running just type **HCAZ** again.



2.2.14 ► Type **1** and press **enter**

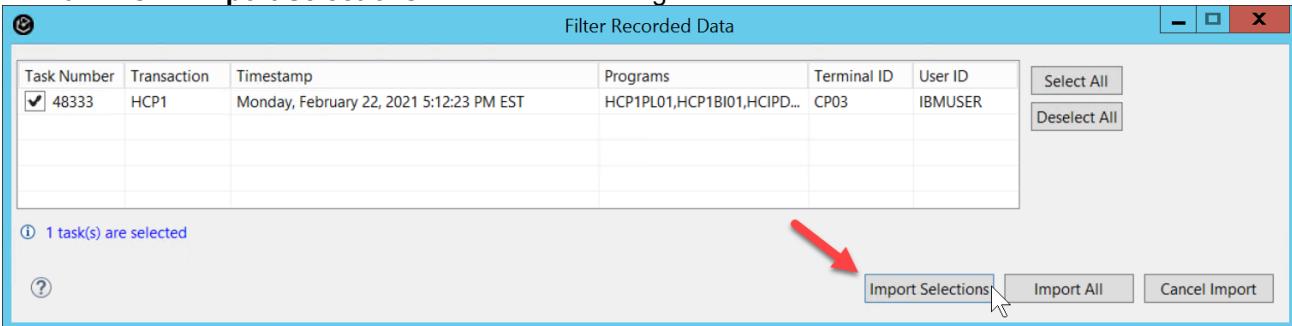


2.2.15 ►| Switch back to IDz minimizing the 3270 screen and click **Stop recording.**



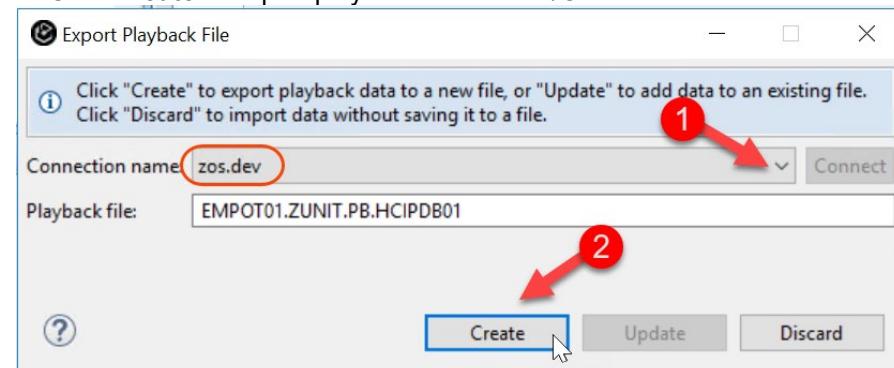
The dialog *Filter Recorded Data* is displayed.

2.2.16 ►| Click **Import Selections to dismiss the dialog.**

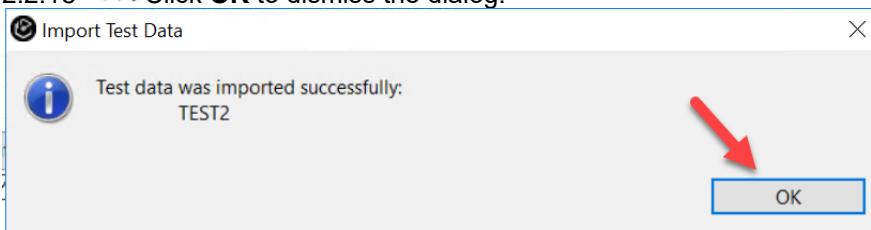


2.2.17 ►| Using drop down select the **zos.dev as *Connection name***

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



2.2.18 ►| Click **OK to dismiss the dialog.**



2.2.19 ► Double click on the **hcipdb01** title to enlarge the view.
In case by mistake you close this editor will need to go back to step 2.2.1



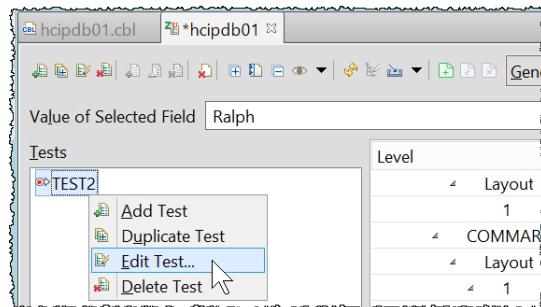
2.2.20 You now see a new test created in the editor and populated with the values from the live run.
► Scroll down the editor and notice the data displayed on the screen that was captured..

Level	Name	PIC	USAGE	TEST2:Inp...	TEST2:Exp...
5	CA-HOW-OFTEN	X(20)	DISPLAY	87	RalphD...@ibm.com
5	CA-ADDITION...	X(3235...)	DISPLAY		
4	3 (redefines CA-THRESHOL...				
5	CA-HR-THRES...	X(10)	DISPLAY	9627811234	Ralph
5	CA-BP-THRESH...	X(10)	DISPLAY		DAlmeida
5	CA-MS-THRES...	X(10)	DISPLAY		1980-07-11
5	CA-ADDITION...	X(3245...)	DISPLAY		34 Main Street ...
4	3 (redefines CA-VISIT-REQU...				
5	CA-VISIT-DATE	X(10)	DISPLAY	9627811234	Ralph
5	CA-VISIT-TIME	X(10)	DISPLAY		DAlmeida
5	CA-HEART-RATE	X(10)	DISPLAY		
5	CA-BLOOD-PR...	X(10)	DISPLAY		
5	CA-MENTAL-S...	X(10)	DISPLAY		
5	CA-ADDITION...	X(3243...)	DISPLAY		
4	EXEC CICS ABEND	line=81			
4	Record:Record1				
	DFHEIBLK				
	LineNumber=81				
4	EXEC CICS RETURN	line=97,112,151			

- 2.2.21 ► 1 Click on **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

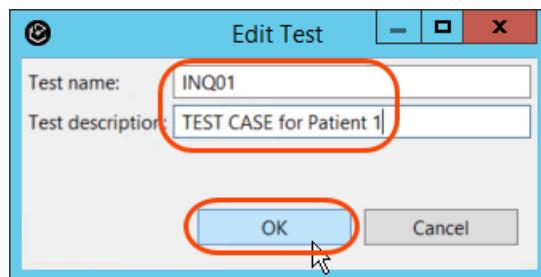
Level	Name	PIC	USAGE	TEST2:Inp...	TEST2:Exp...
1	HCAZERRS	X(8)	DISPLAY		
4	COMMAREA				
4	Layout				
4	1				
3	CA-ERROR-MSG				
3	FILLER	X(9)	DISPLAY		
3	CA-DATA	X(90)	DISPLAY		
4	EXEC SQL SELECT INTO [PATI	line=117			
4	Record:Record1				
4	LineNumber=117				
4	INTO				
5	CA-FIRST-NAME	X(10)	DISPLAY	Ralph	3
5	CA-LAST-NAME	X(20)	DISPLAY	DAlmeida	
5	CA-DOB	X(10)	DISPLAY	1980-07-11	
5	CA-INS-CARD...	X(10)	DISPLAY	9627811234	
5	CA-ADDRESS	X(20)	DISPLAY	34 Main Street ...	
5	CA-CITY	X(20)	DISPLAY	Toronto	
5	CA-POSTCODE	X(10)	DISPLAY	M5H 1T1	
5	CA-PHONE-M...	X(20)	DISPLAY	077-123-9987	
5	CA-EMAIL-AD...	X(50)	DISPLAY	RalphD@ibm.c...	
5	CA-USERID	X(10)	DISPLAY	ralphd	
4	WHERE				
3	DB2-PATIENT-ID	S9(9)	BINARY	1	
	SQLCA				

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.

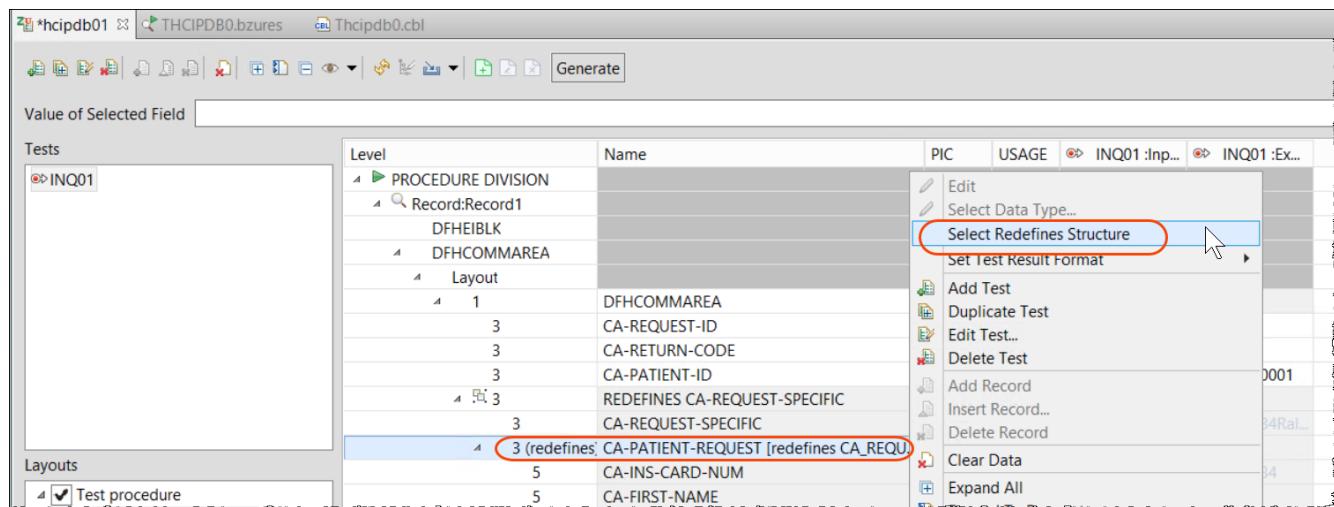
► Use a name like **INQ01** and a description like "**TEST CASE for Patient 1**". Click **OK**



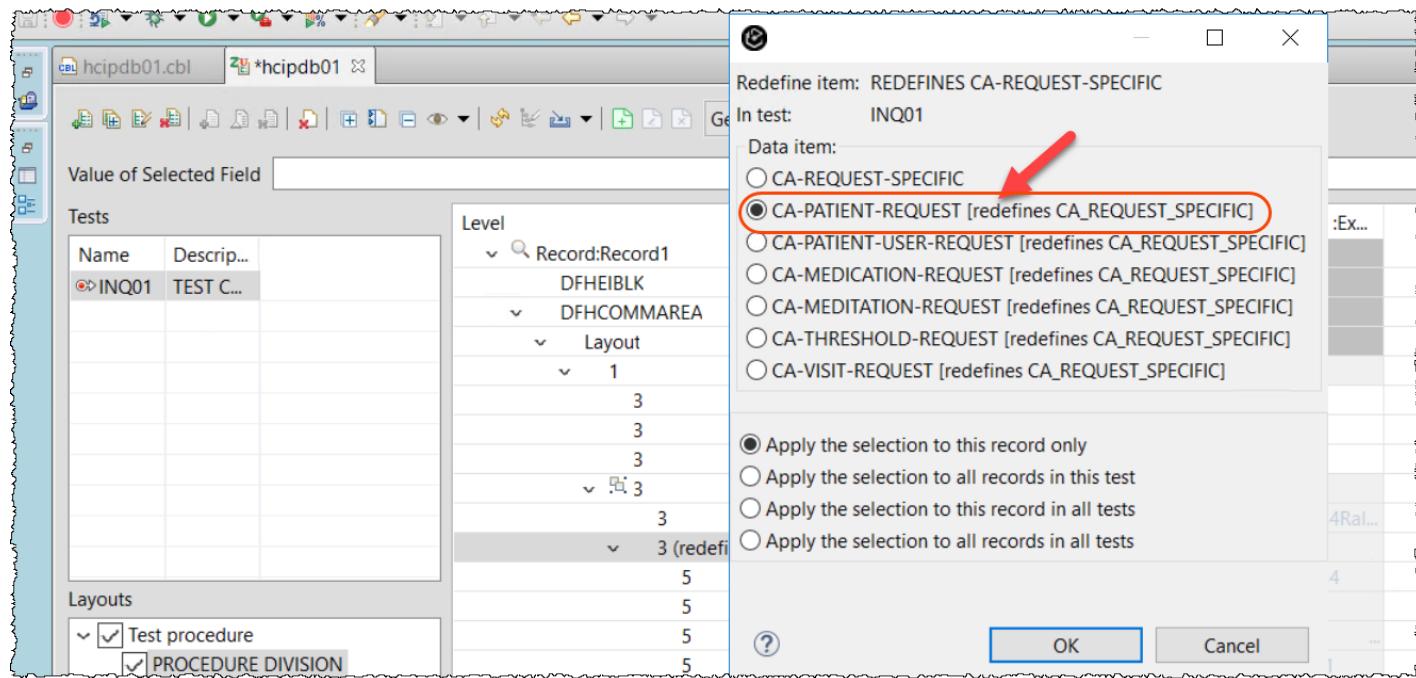
2.2.25 You may notice that this program has many redefines. You may need to identify which is the area being used on this program execution. In our example is the "inquiry patient" area being redefined.

► Scroll up until you find "**(redefines) CA-PATIENT-REQUEST**" (the first REDEFINES).

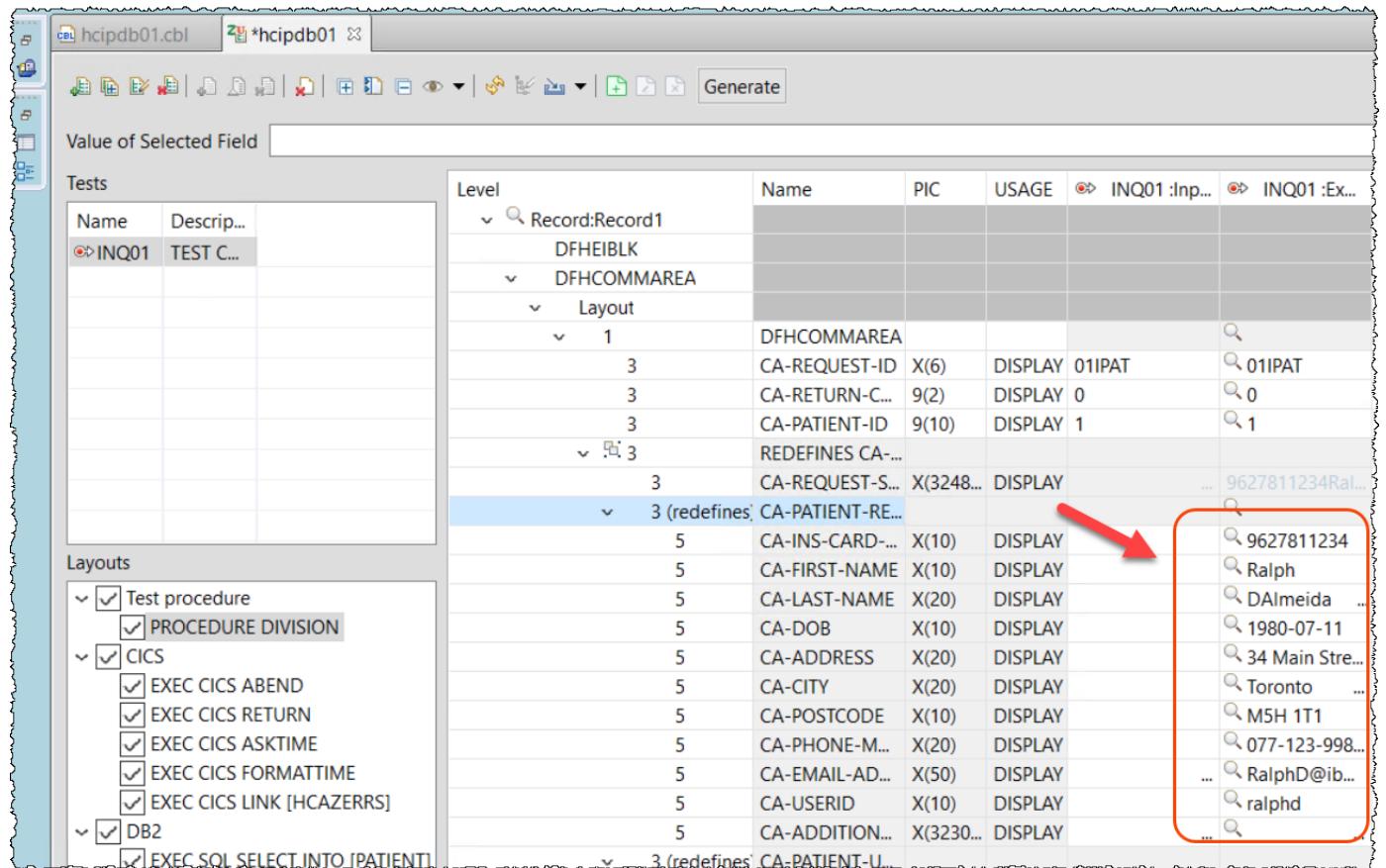
Right click on it and click **Select Redefines Structure**



2.2.26 Select CA-PATIENT-REQUEST and click OK

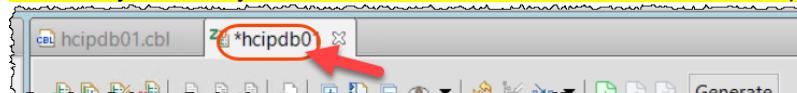


2.2.27 Notice that now the patient data recorded is displayed at the new redefines selected.



2.2.28 ►| Press **Ctrl+S** to save any changes.

2.2.29 ►| Double click again on the **hcipdb01** title to shrink the view.
In case by mistake you close this editor will can re-execute the step 2.2.1

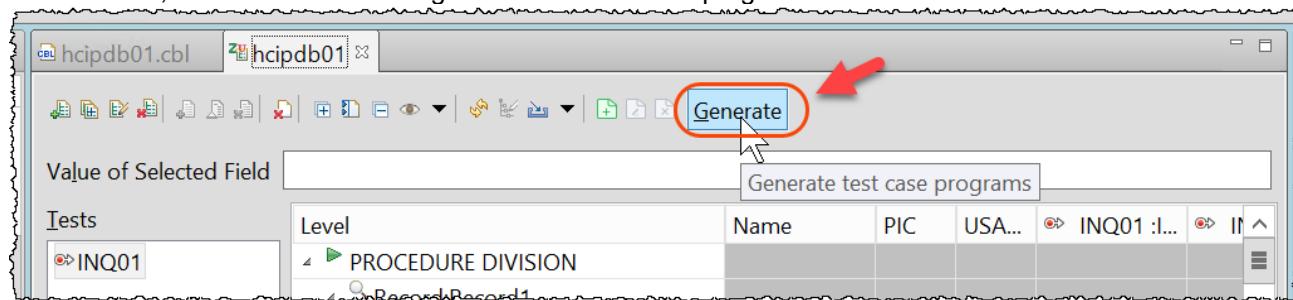


Section 3. Generate, build, and run the unit test.

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

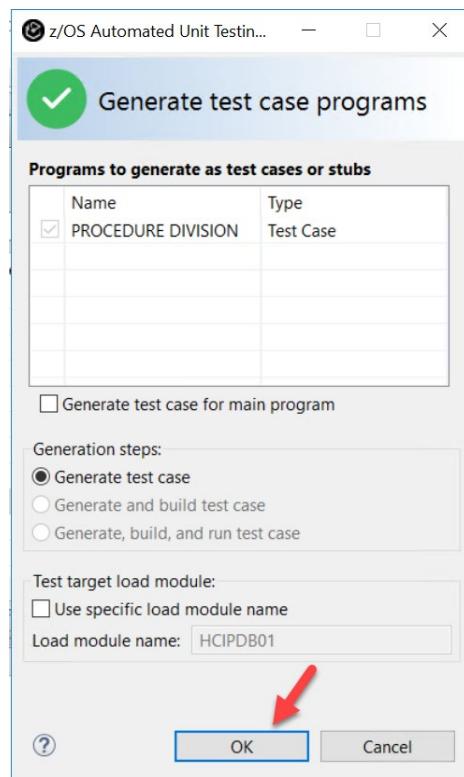
3.1 Generating the test case program.

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 ►| 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 **9 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.

```

-----+*A-1-B-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
1      PROCESS NODLL,NODYNAM,TEST(NOSEP),NOCICS,NOSQL,PGMN(LU)
*+-----+
2      *| Thcipdb0
3      *| PRODUCT: IBM DEVELOPER FOR Z/OS
4      *| COMPONENT: IBM Z/OS AUTOMATED UNIT TESTING FRAMEWORK (ZUNIT)
5      *| FOR ENTERPRISE COBOL AND PL/I
6      *| PROGRAM: ENTERPRISE COBOL ZUNIT TEST CASE FOR DYNAMIC RUNNER
7      *| DATE GENERATED: 03/12/2021 11:45
8      *| ID: f5efa266-00b7-4a39-a2f9-81d240238bd9
9      *| TEST_INQ01
10     *|   THIS PROGRAM IS FOR TEST INQ01
11
12
13
14
15* IDENTIFICATION DIVISION.
16* PROGRAM-ID. 'TEST_INQ01'.
17* DATA DIVISION.
18* WORKING-STORAGE SECTION.
19 01 PROGRAM-NAME PIC X(8) VALUE 'HCIPDB01'.
20 01 BZ-ASSERT.
21 03 MESSAGE-LEN PIC S9(4) COMP-4 VALUE 24.

```

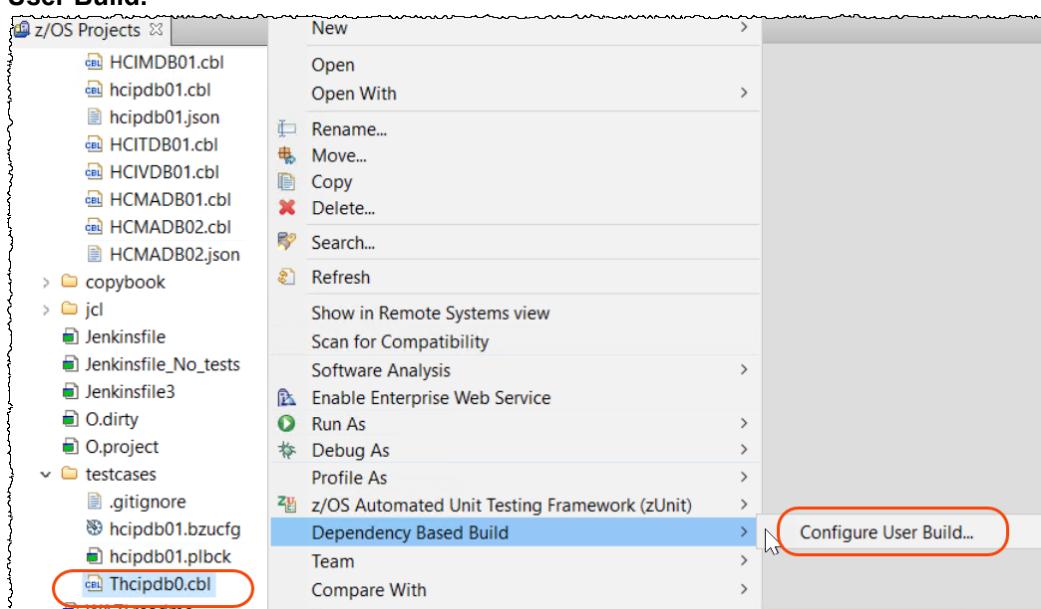
3.2 Building the generated test case programs using IBM DBB .

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.

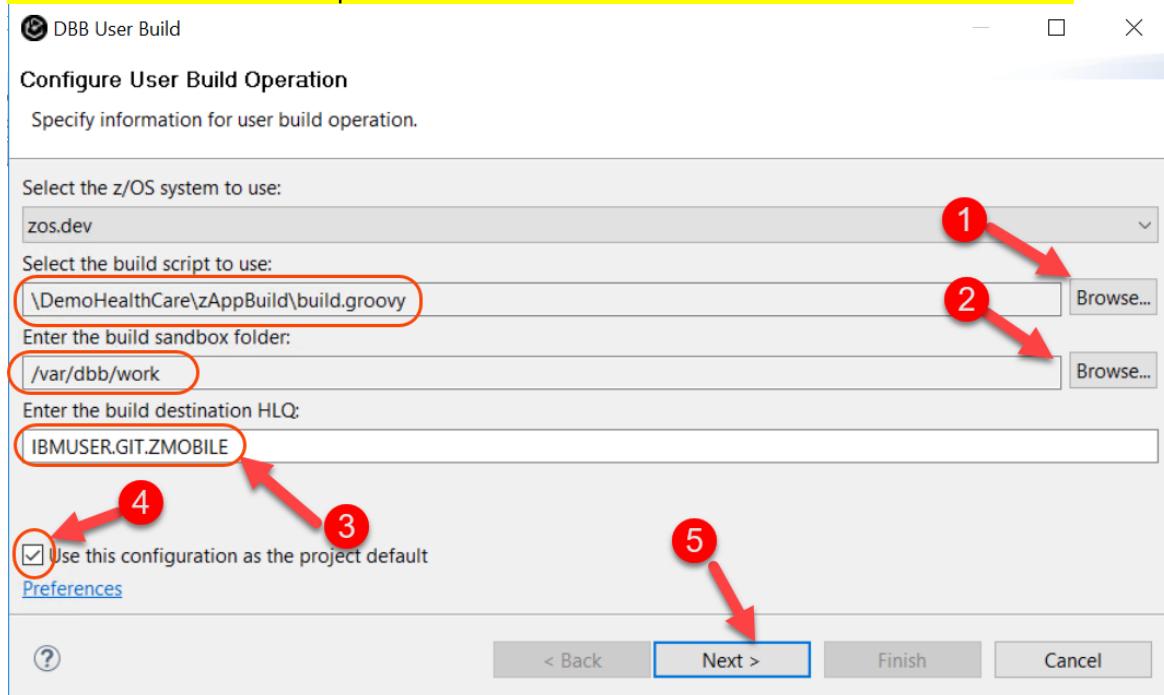
3.2.1 ► Use **Ctrl + Shift + F4** to close all opened editors.

3.2.2 ► Right click on **Thcipdb0.cbl** (under **testcases**) and select **Dependency Based Build > Configure User Build**.



3.2.3 ► Those values could be already there, otherwise use the **Browse...** buttons to specify the values as below. As build destination HLQ use **IBMUSER.GIT.ZMOBILE**, select **Use this configuration as the project default** and click **Next**

Notice that the COBOL compiler on z/OS will use datasets **IBMUSER.GIT.ZMOBILE.xxxx**

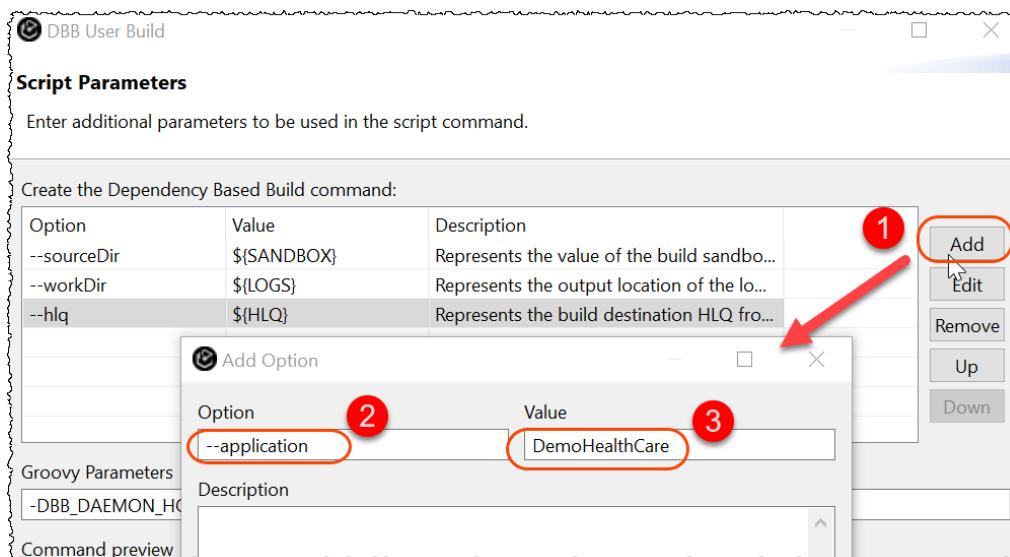


3.2.4 ► Click **Next** Twice

3.2.5 On dialog **Script Parameters**, if the “**--application**” and “**--debug**” are not there you must add
► Otherwise skip to 3.2.7

In case you need to add **--application**.

► Click **Add** to insert **--application** and **DemoHealthCare** (mixed case) and click **OK**



3.2.6 In case you need to add --debug.

► Click Add to insert **--debug** and click OK

Verify that the options below were inserted. Notice that there is an “**--**” before each option.

Create the Dependency Based Build command:

Option	Value	Description
--sourceDir	\$(SANDBOX)	Represents the value of the build sandbox...
--workDir	\$(LOGS)	Represents the output location of the lo...
--hlq	\$(HLQ)	Represents the build destination HLQ fro...
--application	DemoHealthCare	

Add Option

Option: **--debug**

Value: [empty]

3.2.7 The field **Groovy Parameters** should be cleared if there is something already specified there.

► Be sure that the Groovy parameters field is **empty**

► Click Next

DBB User Build

Script Parameters

Enter additional parameters to be used in the script command.

Create the Dependency Based Build command:

Option	Value	Description
--sourceDir	\$(SANDBOX)	Represents the value of the build sandbox from the first screen...
--workDir	\$(LOGS)	Represents the output location of the log folder from the first ...
--hlq	\$(HLQ)	Represents the build destination HLQ from the first screen of t...
--application	DemoHealthCare	
--debug		

Groovy Parameters: [empty]

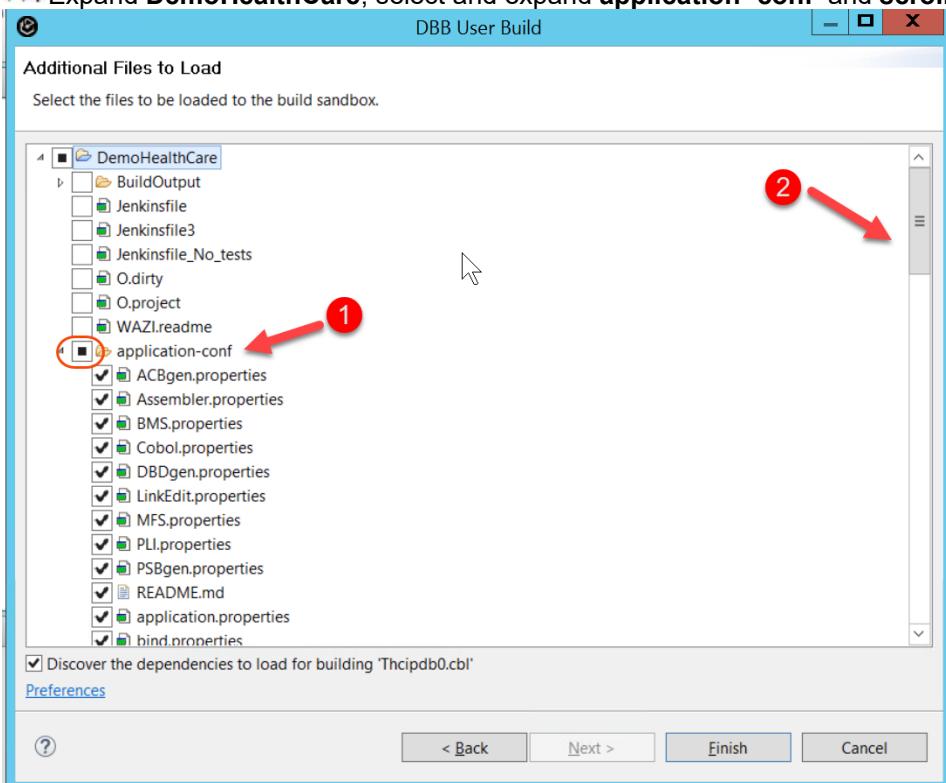
Command preview:

```
$DBB_HOME/bin/groovyz /var/dbb/work/DemoHealthCare/zAppBuild/build.groovy --sourceDir /var/dbb/work --workDir /var/dbb/logs --hlq $(HLQ) --application DemoHealthCare --debug
```

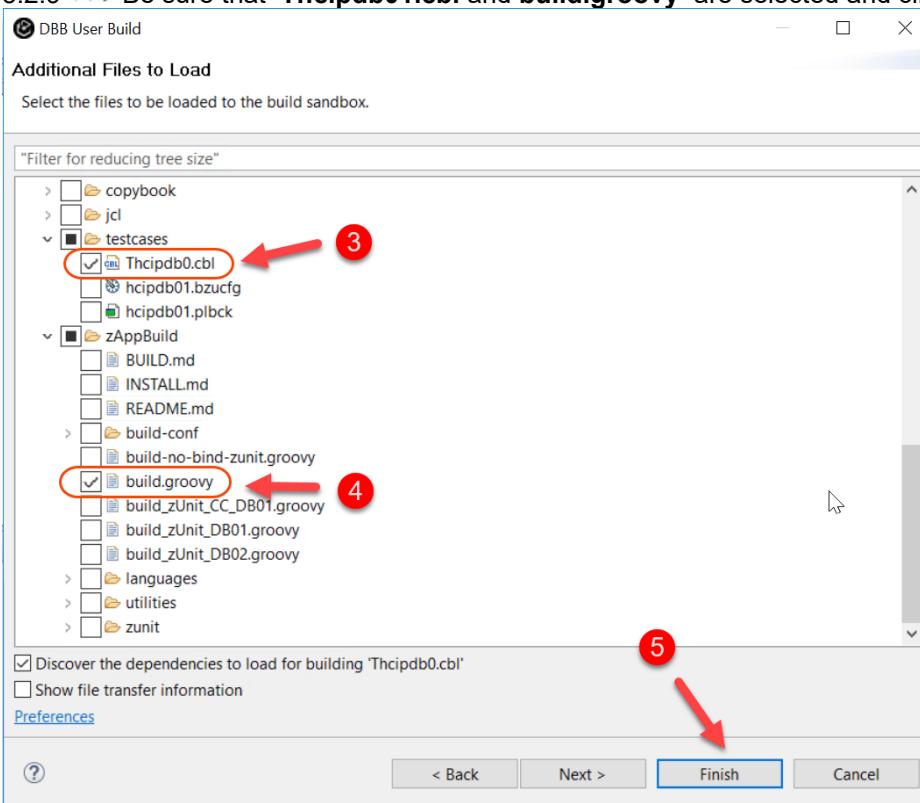
< Back Next > **Finish** Cancel

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

► Expand DemoHealthCare, select and expand application -conf and scroll down

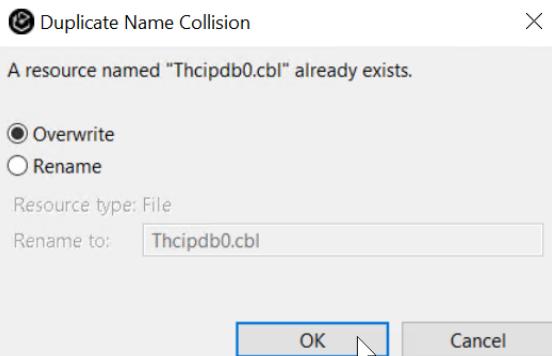


3.2.9 ► Be sure that Thcipdb0.cbl and build.groovy are selected and click Finish

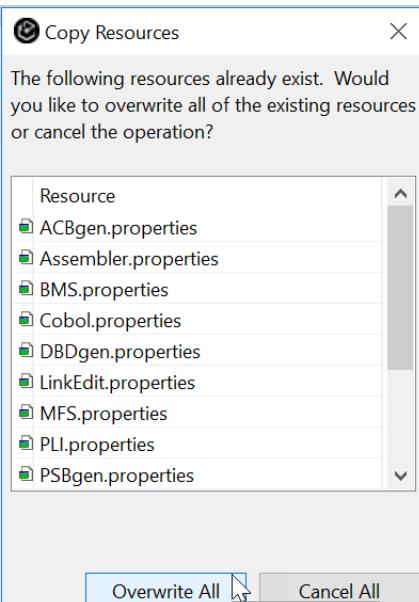


3.2.9 Those assets were once already moved to the z/OS UNIX files and will overwrite the old copies.

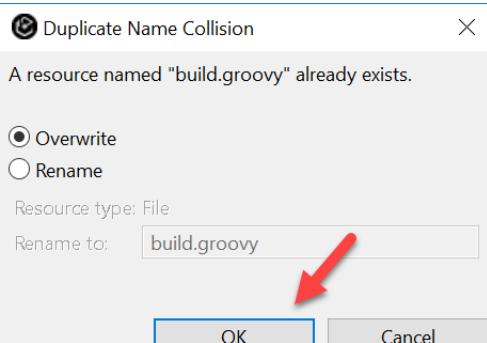
▶ Click OK



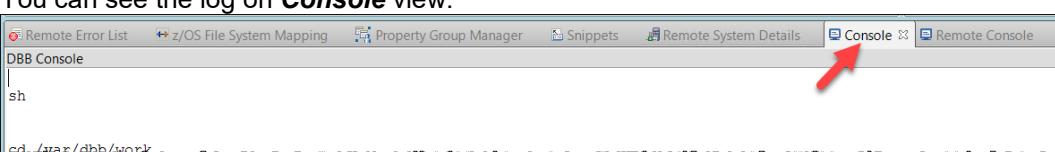
▶ Click Overwrite All



▶ For duplicate name collision click OK



The assets will be moved to **z/OS UNIX Files** and the **build.groovy** will start.
You can see the log on **Console** view.



If the DBB user build fails
If you have errors as below: (Resources temporary unavailable)

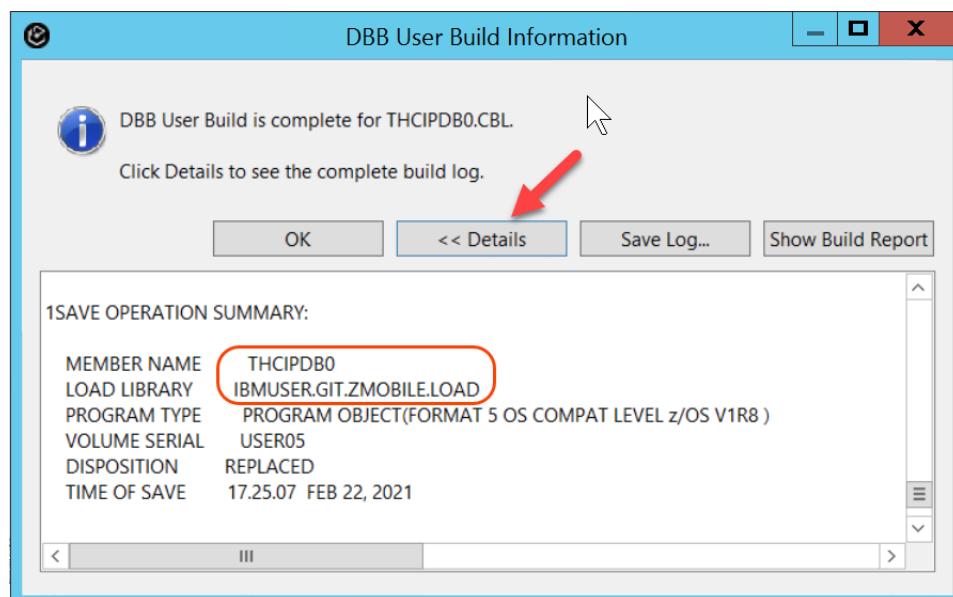


DBB Console
 Cannot run program "sh" (in directory "/var/dbb/work"): EDC5112I Resource temporarily unavailable

It's because your z/OS is stressed., One easy way to free the z/OS resources is logging on TSO as IBMUSER password **SYS1** , go to M.5 and issue **/C CICSTS53** And Try again

3.2.10 This operation will compile, and link the 9 generated COBOL programs **and it may take up to 3 minutes.** You can follow this using the **Console view** opened in the bottom.

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



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

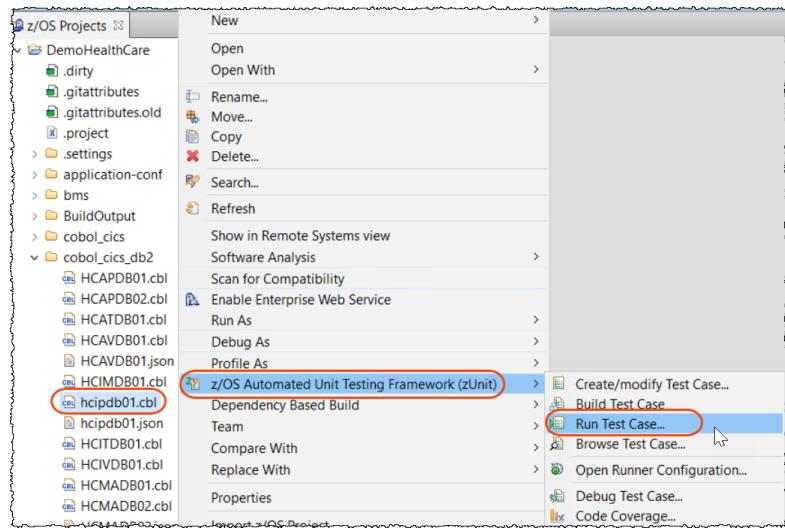
Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote System Details Console

DBB Console
 /S0W1/var/dbb/work>
 ** Build ended at Mon Feb 22 17:26:10 CST 2021
 ** Build State : CLEAN
 ** Total files processed : 1
 ** Total build time : 3 minutes, 11.925 seconds
 /S0W1/var/dbb/work>
 ** Build finished

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.

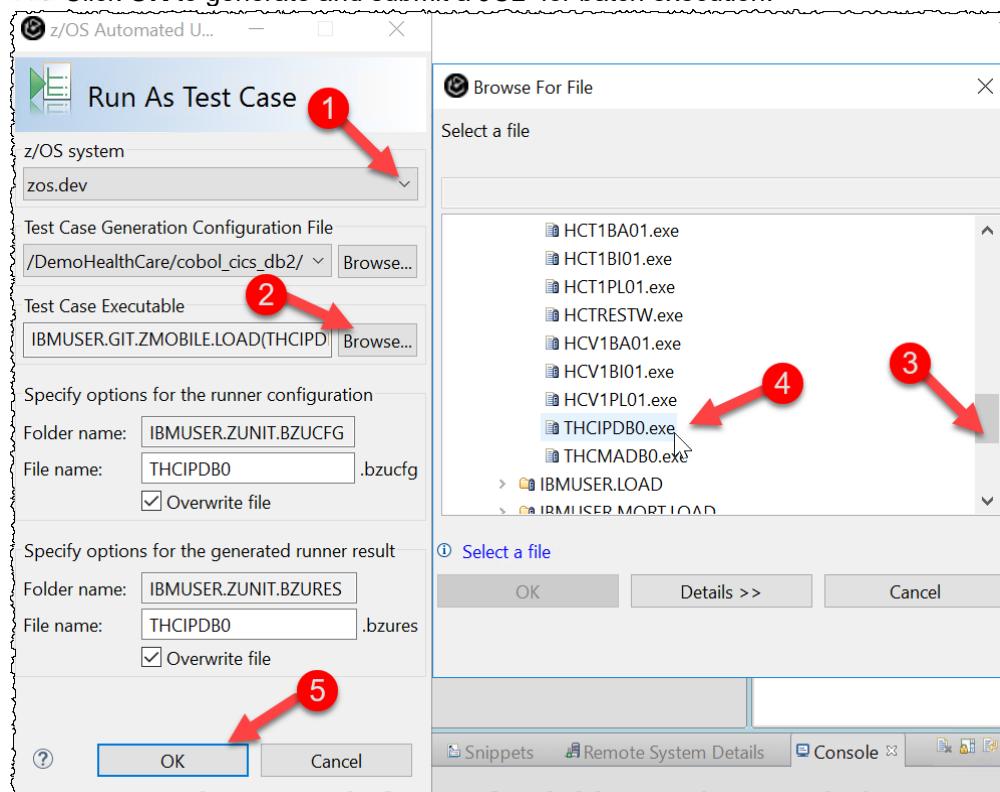
3.3.1 ► Using z/OS Projects view, scroll up, right click on **hcipdb01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)>Run Test Case...**



3.3.2 ► Select **zos.dev** and use the **Browse** button to select the PDS and load module where the test case was created.

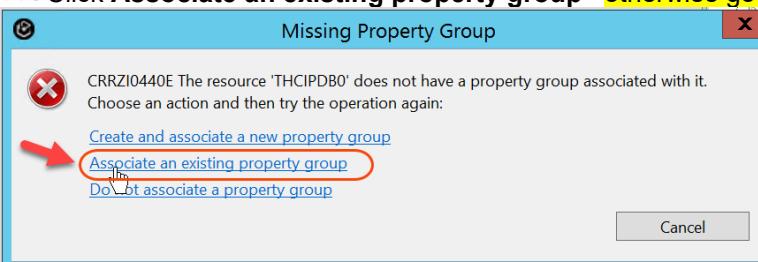
You will find the PDS **IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)** under **My Data Sets ..**

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

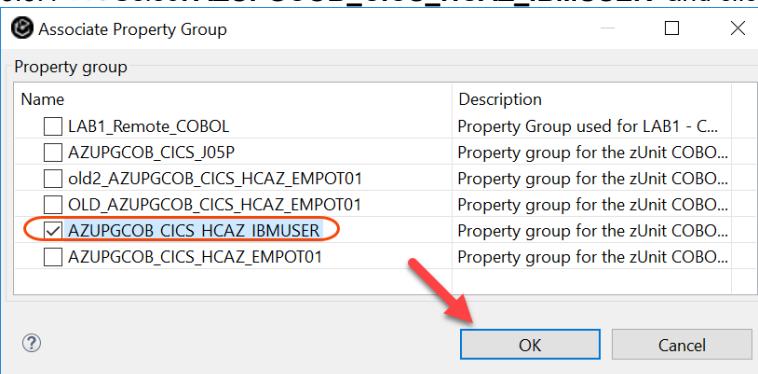


3.3.3 If the *Missing Property Group* dialog will prompt.

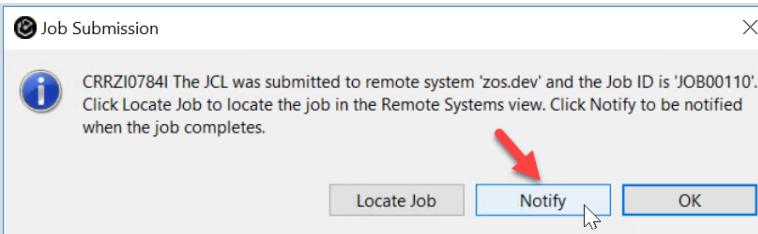
► Click **Associate an existing property group** otherwise go to 3.3.5



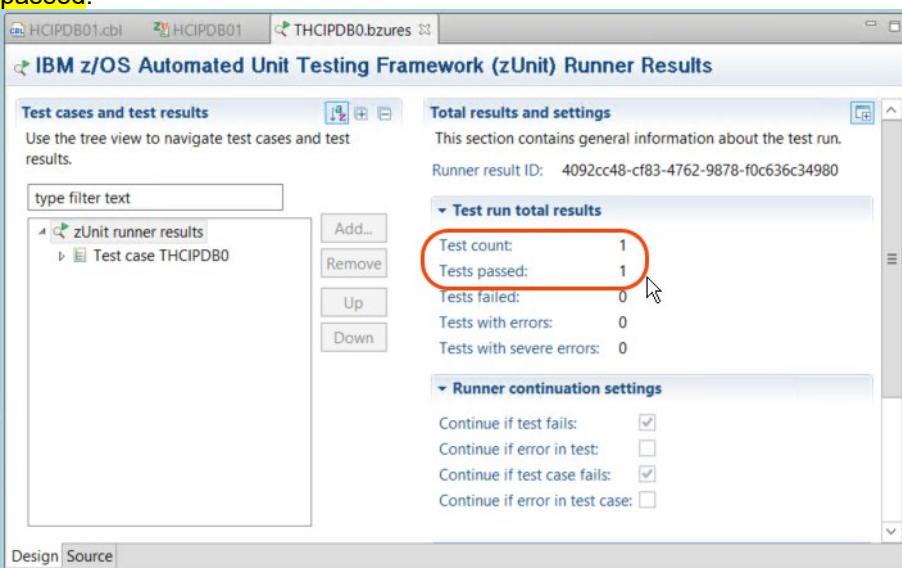
3.3.4 ► Select **AZUPGCOB_CICS_HCAZ_IBMUSER** and click **OK**



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



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



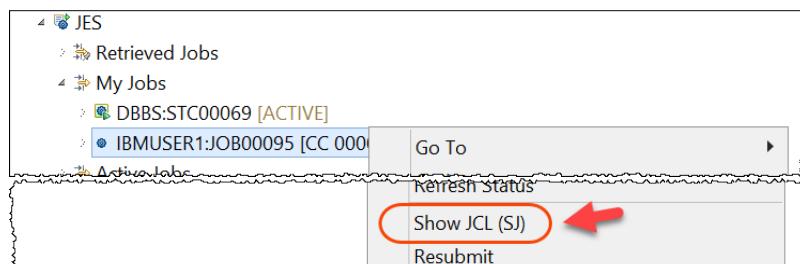
You have now created a test case with data imported from a recorded run and have been successful in testing a CICS program without the need of a CICS and DB2 environment.

3.3.7 ►| 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 that has **CC 000** (NOT the ACTIVE) 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.

```

//IBMUSER1 JOB,
// MSGCLASS=H, MSGLEVEL=(1,1), REGION=0M, COND=(16, LT)
///* Action: Run Test Case...
///* Source: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)
// SET FELJOB=ZUNITGO
// RUNNER EXEC PROC=BZUPPLAY,
// BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),
// BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,
// BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,
// PRM='STOP=E,REPORT=XML'
// REPLAY.BZUPLAY DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.PB1.HCIPDB01
// REPLAY.BZURPT DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)
// STEPLIB DD
// DSN=IBMUSER.LOAD, DISP=SHR
// DSN=IBMUSER.GIT.ZMOBILE.LOAD, DISP=SHR

```

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

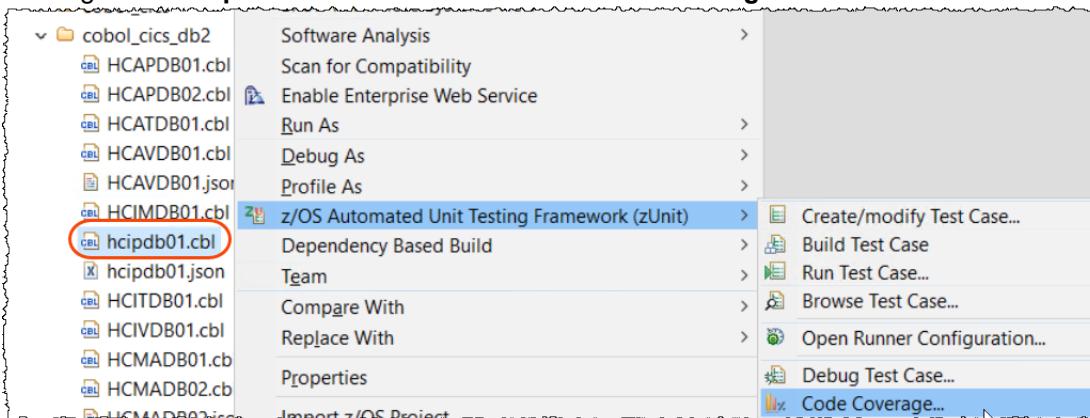
3.5 Running zUnit test case Code Coverage

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.

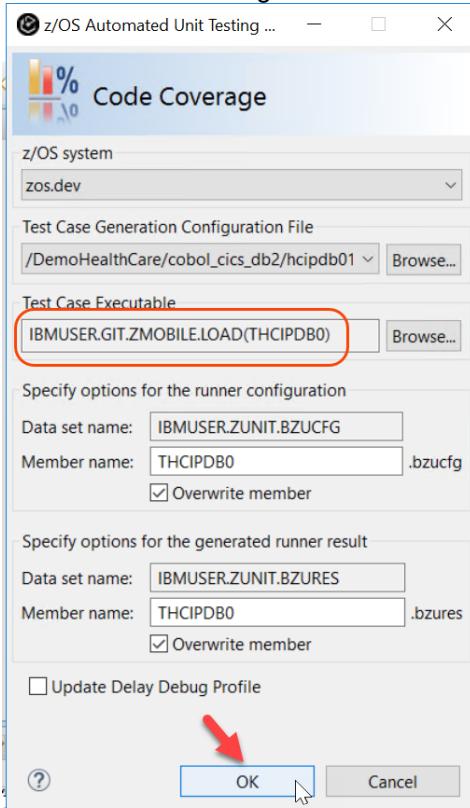
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, which is very handy..

3.5.1 ZUnit also provides the capability to run the test case using Code coverage or Debug.

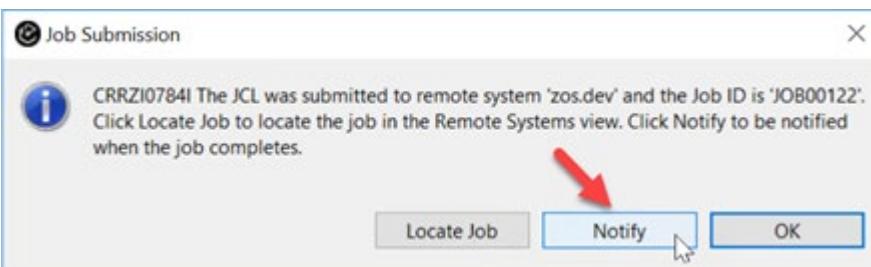
► Right click **hcipdb01.cbl** and select **zUnit > Code Coverage....**



3.5.2 ► Click **OK** to generate and submit a JCL for batch execution.



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



3.5.4 Make sure the job completes with completion code of 0000.

```
[5/12/20, 3:22 PM] Job JOB00614 is being submitted
[5/12/20, 3:22 PM] Job EMPO11:JOB00614 ended with completion code CC 0000
[5/12/20, 3:40 PM] Job JOB00618 is being submitted
[5/12/20, 3:40 PM] Job EMPO11:JOB00618 ended with completion code CC 0000
```

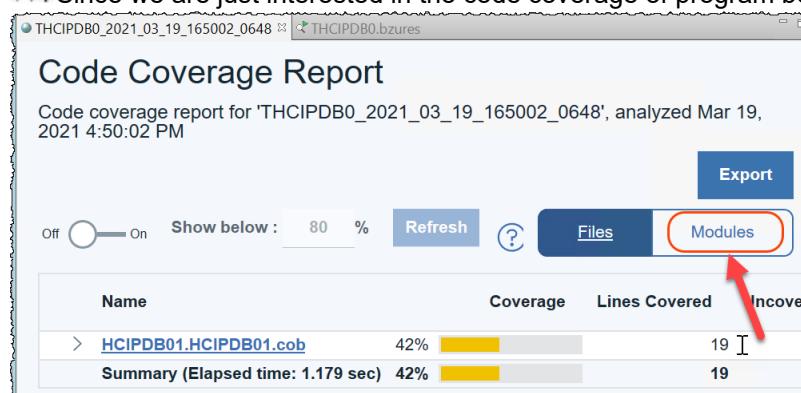
3.5.5 The code coverage report shows all COBOL programs executed for this specific test case.

► Click on title **THCIPDB0_yyyy_mm_dd_xxxxxxx** to see the Code coverage report



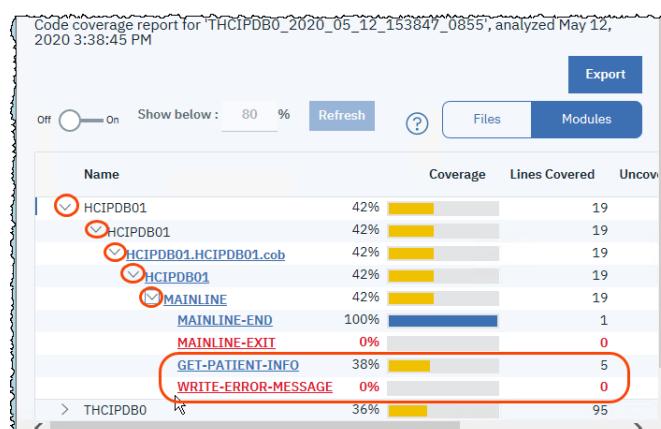
3.5.6 This code coverage report shows the coverage of the COBOL programs generated for zUnit to perform the test as well our *HCIPDB01* COBOL program..

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

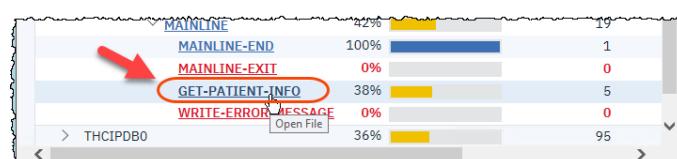


3.5.7 ► Expand the nodes as below to see the COBOL paragraphs.

Notice that **GET-PATIENT-INFO** has 38% of coverage and that **WRITE-ERROR-MESSAGE** was not covered at all on this test case.



3.5.8 ► Click on **GET-PATIENT-INFO** to see more details



3.5.9 ► 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.
(you may have different values than the screen capture below).

```

-----+*A-1-B-----2-----3-----4-----5-----6-----7-----+
255          :CA-ADDRESS,
256          :CA-CITY,
257          :CA-POSTCODE,
258          :CA-PHONE-MOBILE,
259          :CA-EMAIL-ADDRESS,
260          :CA-USERRID
261          FROM PATIENT
262          WHERE PATIENTID = :DB2-PATIENT-ID
263          END-EXEC.
264          Evaluate SQLCODE
265          When 0
266          MOVE '00' TO CA-RETURN-CODE
267          MOVE '01' TO CA-RETURN-CODE
268          When -913
269          MOVE '01' TO CA-RETURN-CODE
270          When Other
271          MOVE '90' TO CA-RETURN-CODE
272          PERFORM WRITE-ERROR-MESSAGE
273          EXEC CICS RETURN END-EXEC
274          TMD-Evaluate

```

3.5.10 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 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.

3.5.11 We also 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/HCIPICC.jcl](#)

► Double click on [jcl/HCIPICC.jcl](#) and verify this JCL. You could just submit this JCL to run the test case and the Code Coverage.

```

-----+-----1-----2-----3-----4-----5-----6-----+
1 //HCIPCC JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
3 // Action: Code Coverage...
4 // * Source: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)
5 // RUNNER EXEC PROC=B2UPLAY,
6 // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),
7 // BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,
8 // BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,
9 // PARM='STOP=E,REPORT=XML'
10 //CEEOPTS DD *
11 TEST(ALL,,PROMPT,DBMDT%IBMUSER:*)
12 ENVAR(
13 "EQA_STARTUP_KEY=CC,THCIPDB0,cclevel=LINE,testid=THCIPDB0")
14 /*
15 //BZUPLAY DD DISP=SHR,
16 // DSN=IBMUSER.ZUNIT.PB1.HCIPDB01
17 //BZURPT DD DISP=SHR,
18 // DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)

```

Section 4. Introduce a bug in the program and rerun the unit test.

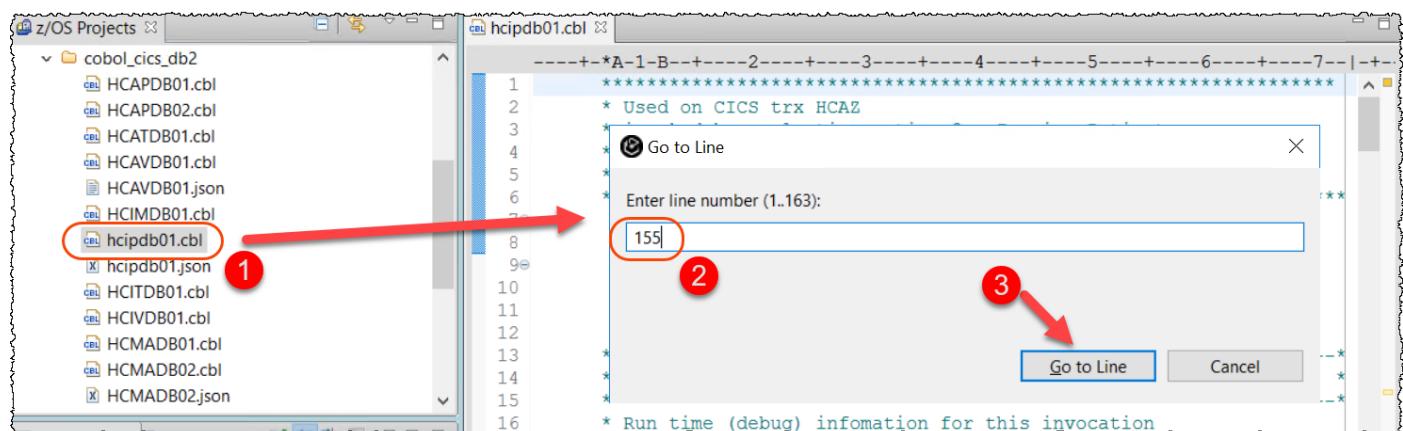
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

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 **Go to Line**



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** also **DO NOT** change line 159.

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

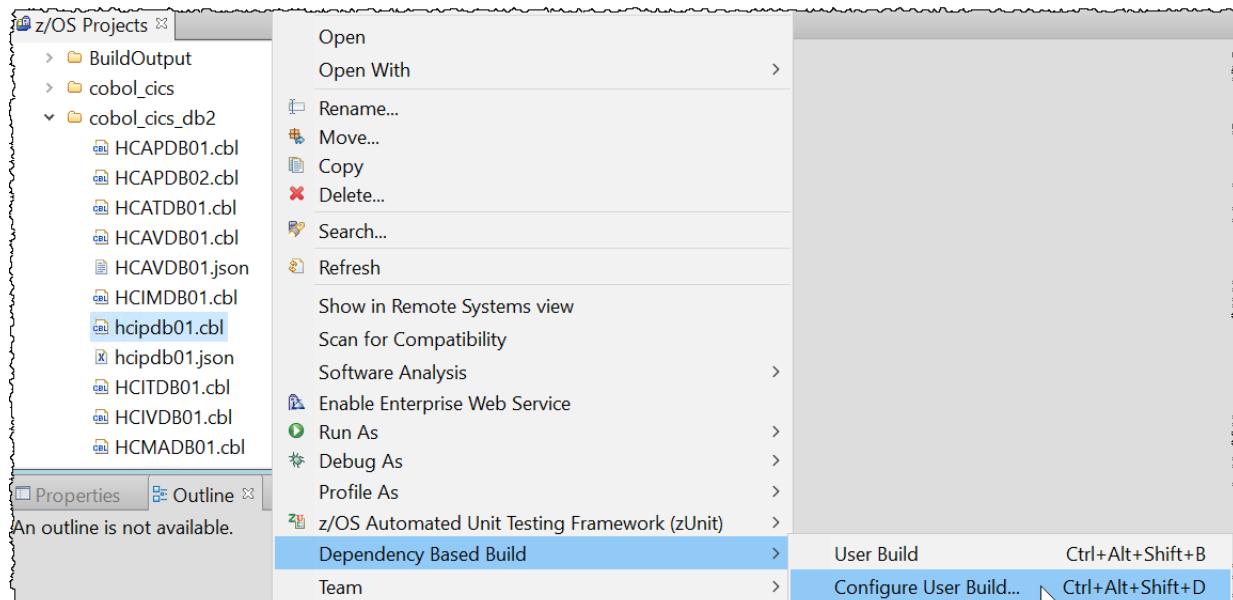
```

141      END-EXEC.
142          Evaluate SQLCODE
143              When 0
144                  MOVE '00' TO CA-RETURN-CODE
145              When 100
146                  MOVE '01' TO CA-RETURN-CODE
147              When -913
148                  MOVE '01' TO CA-RETURN-CODE
149              When Other
150                  MOVE '90' TO CA-RETURN-CODE
151                  PERFORM WRITE-ERROR-MESSAGE
152                  EXEC CICS RETURN END-EXEC
153          END-Evaluate.
154          * %bug2 -- the line below will introduce a BUG
155          *
156          IF DB2-PATIENT-ID = 1
157              MOVE "BAD NAME" to CA-FIRST-NAME
158          END-IF
159          * MOVE "02" to CA-NEWFIELD
160          *
161          EXIT.
162          *
163          COPY HCERRSPD.

```

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** 3 times

DBB User Build

Configure User Build Operation

Specify information for user build operation.

Select the z/OS system to use:
zos.dev

Select the build script to use:
\\DemoHealthCare\\zAppBuild\\build.groovy

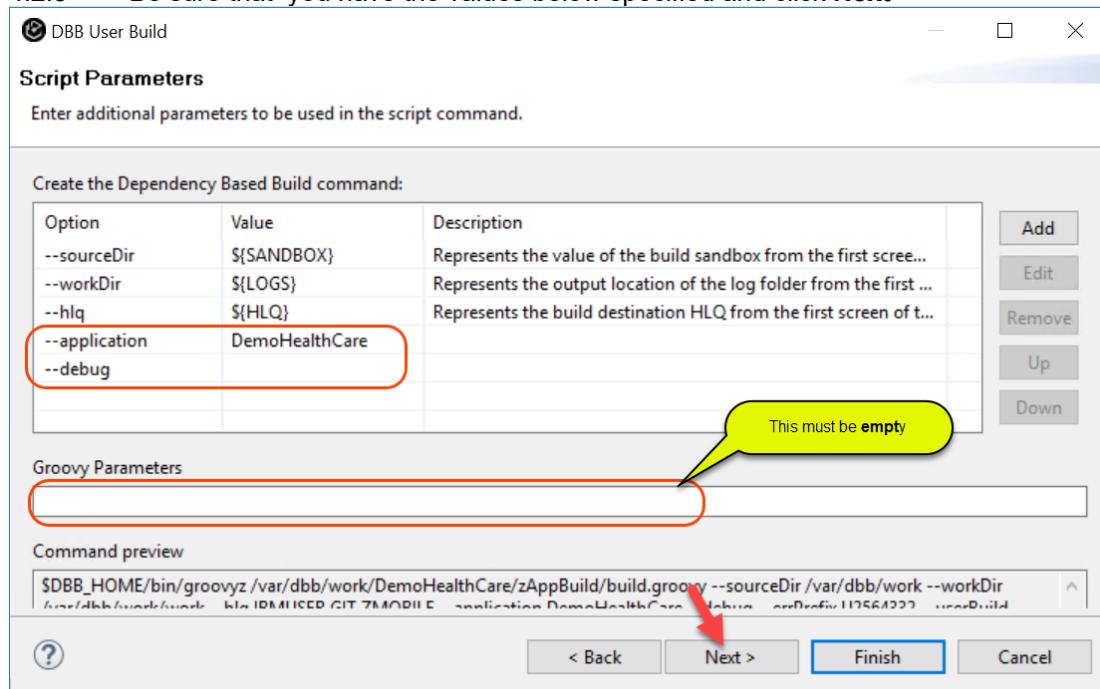
Enter the build sandbox folder:
/var/dbb/work

Enter the build destination HLQ:
IBMUSER.GIT.ZMOBILE

Use this configuration as the project default
[Preferences](#)

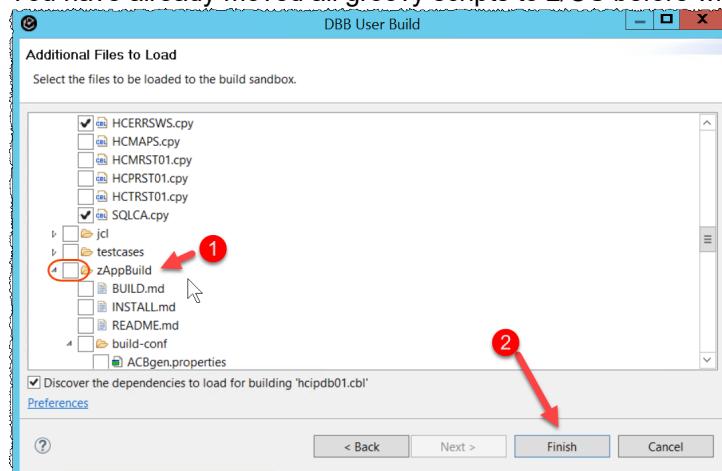
? < Back Next > Finish Cancel

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

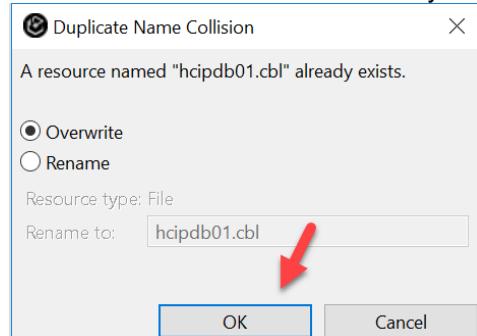


4.2.4 ➡ Expand **DemoHealthCare**, Un-select **zAppBuild** and click **Finish**.

You have already moved all groovy scripts to z/OS before when compiled the Test case program.

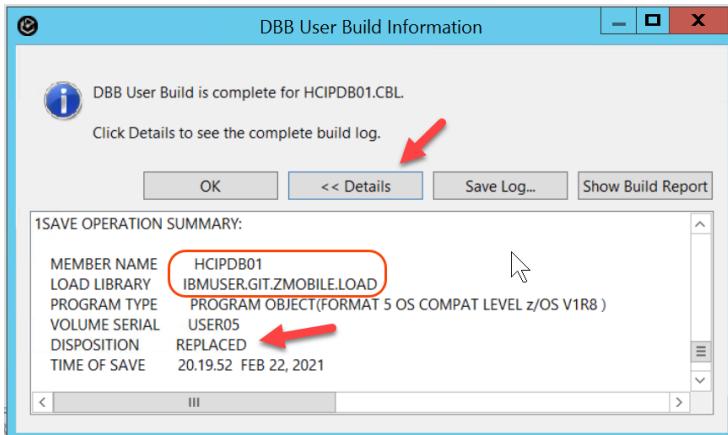


4.2.5 ➡ Overwrite the code already on z/OS as the example below

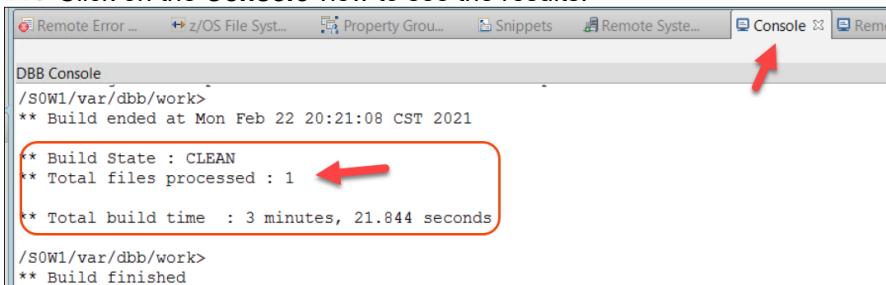


4.2.6 This operation may take up to 2 minutes When finished the dialog below will be displayed.

► Click on **Details** and **OK** after you verified that the load module was replaced.



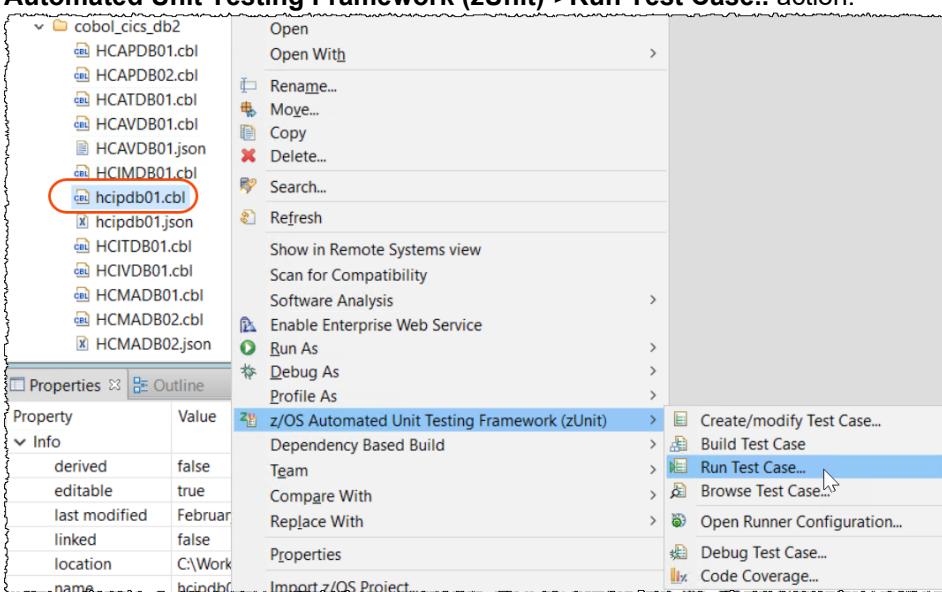
► Click on the **Console** view to see the results:



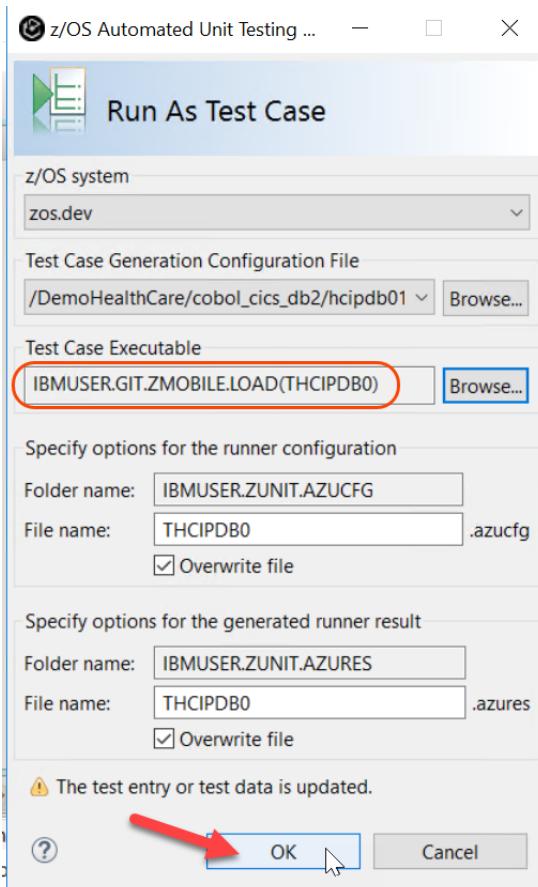
4.3 Running the test case again

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

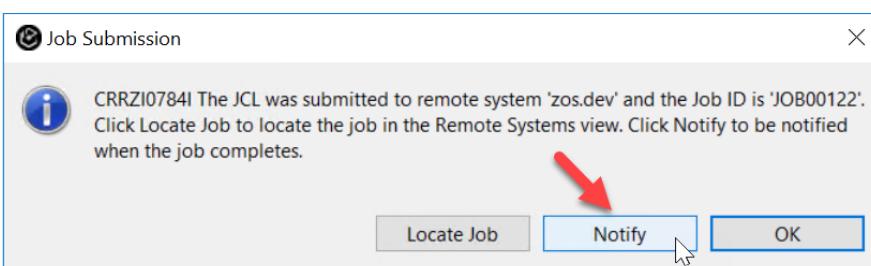
4.3.1 ► On the z/OS Projects view, select **hcipdb01.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..** action.



4.3.2 ► Be sure that the test case load module **IBMUSER.GIT.ZMOBILE LOAD(THCIPDB0)** is selected and click **OK** to submit the execution via JCL.



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



4.3.4 ► Notice that now you have a completion code of **0004** instead of 0000.

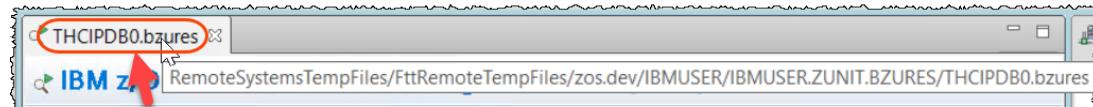
```
[3/17/21, 11:09 AM] Job JOB00113 is being submitted
[3/17/21, 11:09 AM] Job IBMUSER1:JOB00113 ended with completion code CC 0000
[3/17/21, 11:37 AM] Job JOB00122 is being submitted
[3/17/21, 11:37 AM] Job IBMUSER1:JOB00122 ended with completion code CC 0004
```

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

► Click **Test case THCHIPDB0**. The failure is expected because the expected value of the error message is different from the actual value.

Test case ID:	e30e513c-140b-4176-b4f1-0b497de49f44
Module name:	THCHIPDB0
Test case name:	THCHIPDB0
Result:	fail
Test count:	1
Tests passed:	0
Tests failed:	1
Tests with errors:	0
Tests with severe errors:	0

4.3.6 ► Double click on title **THCHIPDB0.bzures** to maximize the windows and better check the results



4.3.7 ► Expand **Test case THCHIPDB0, Test INQ01 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure

Exception details

If the result of the test is "fail" or "error", this section contains the details of the exception.

```
BZUPU00W ASSERT=COMPARE FAILED IN PROCEDURE DIVISION.
BZUP220I TEST RUN TEST2 REGISTERED FOR SUBSYS=CICS FILTER ON=HCIPDB01 STUB CALL=Y
BZUP400I STARTING TRANSACTION=HCP1 USING PROGRAM=TEST2
BZUPU00W ASSERT=COMPARE FAILED IN PROCEDURE DIVISION.
BZUPT001 ITEM NAME=CA-VISIT-TIME OF CA-VISIT-REQUEST OF DFHCOMMAREA
BZUPT001 VALUE=BAD NAME
BZUPT001 EXPECTED VALUE=Ralph
BZUP002I FINISHED EXECUTION RC=04
```

Message:

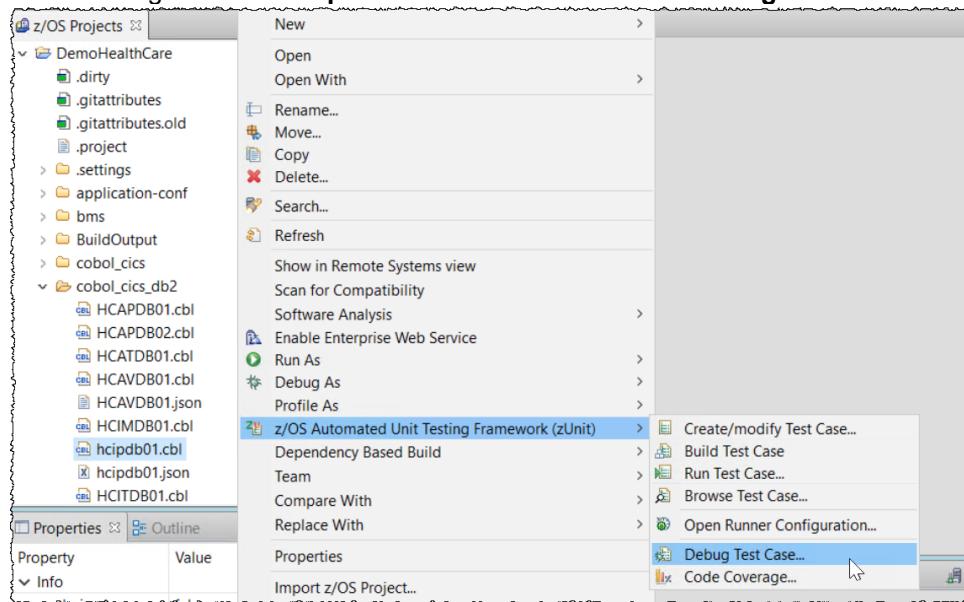
Component code: BZU
Message number: 3000
Message severity: 4

4.4 Running zUnit test case with debugging

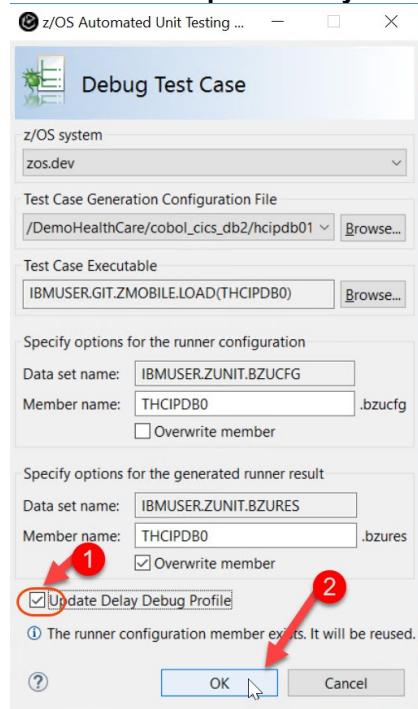
The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example. **Notice that you are debugging a batch job without need to have CICS and DB2 active, which is very handy.**

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

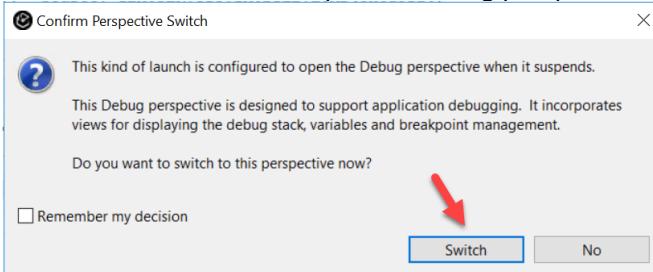
4.4.2 ➡ Right click on **hcipdb01.cbl** and select **zUnit → Debug Test Case ...**



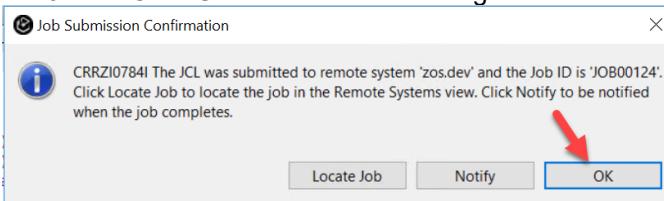
4.4.3 ➡ Click **Update Delay Debug Profile** and click **OK**



4.4.4 ► Click **Switch** to open the debug perspective.



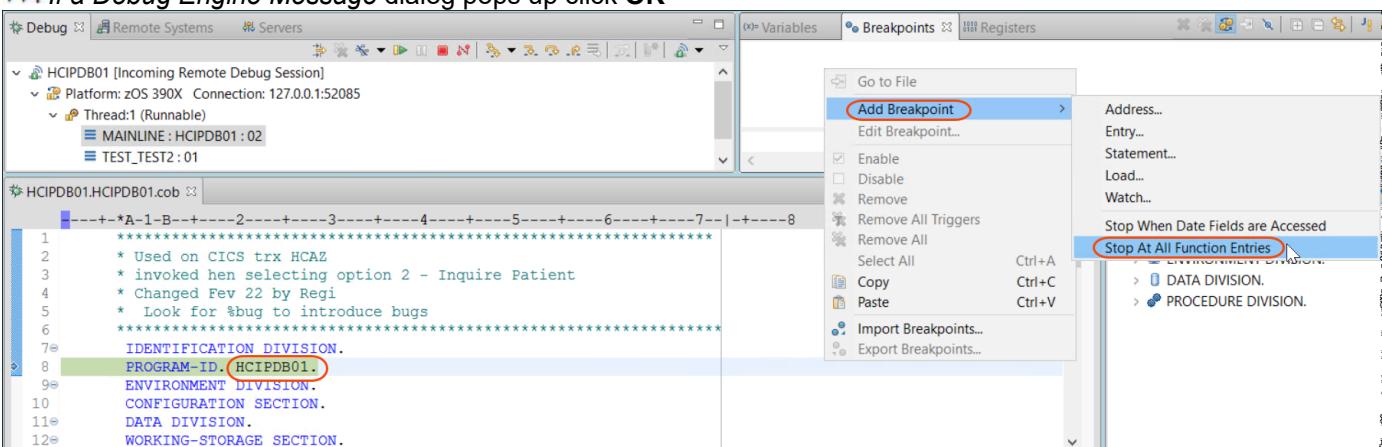
4.4.5 ► Click **OK** to dismiss this dialog



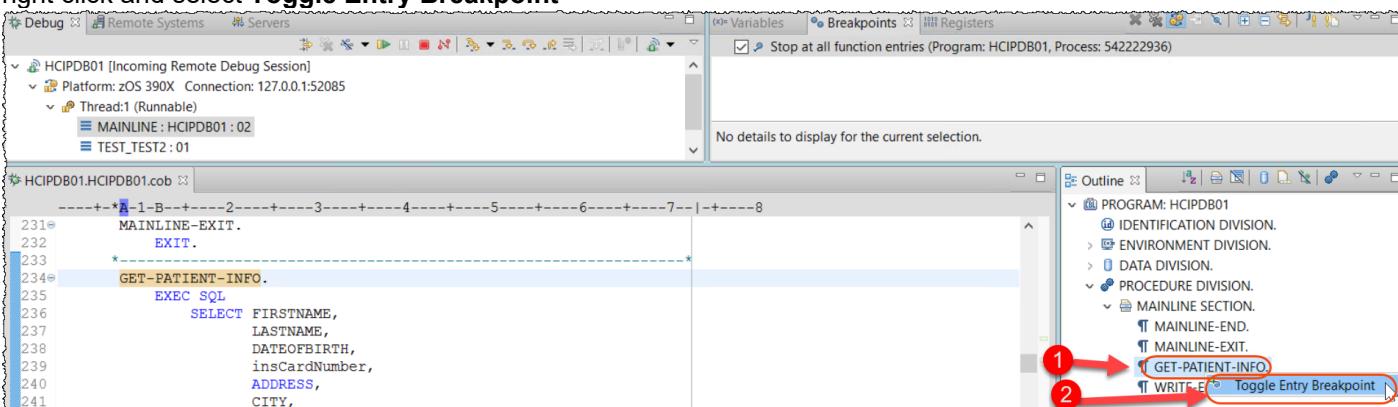
4.4.6 Notice that the first program being debugged is the program that you introduced the bug. (**Hcipdb01**)
The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

► 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*.

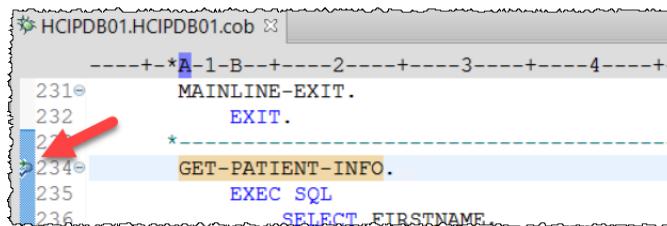
► If a **Debug Engine Message** dialog pops up click **OK**



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



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



```

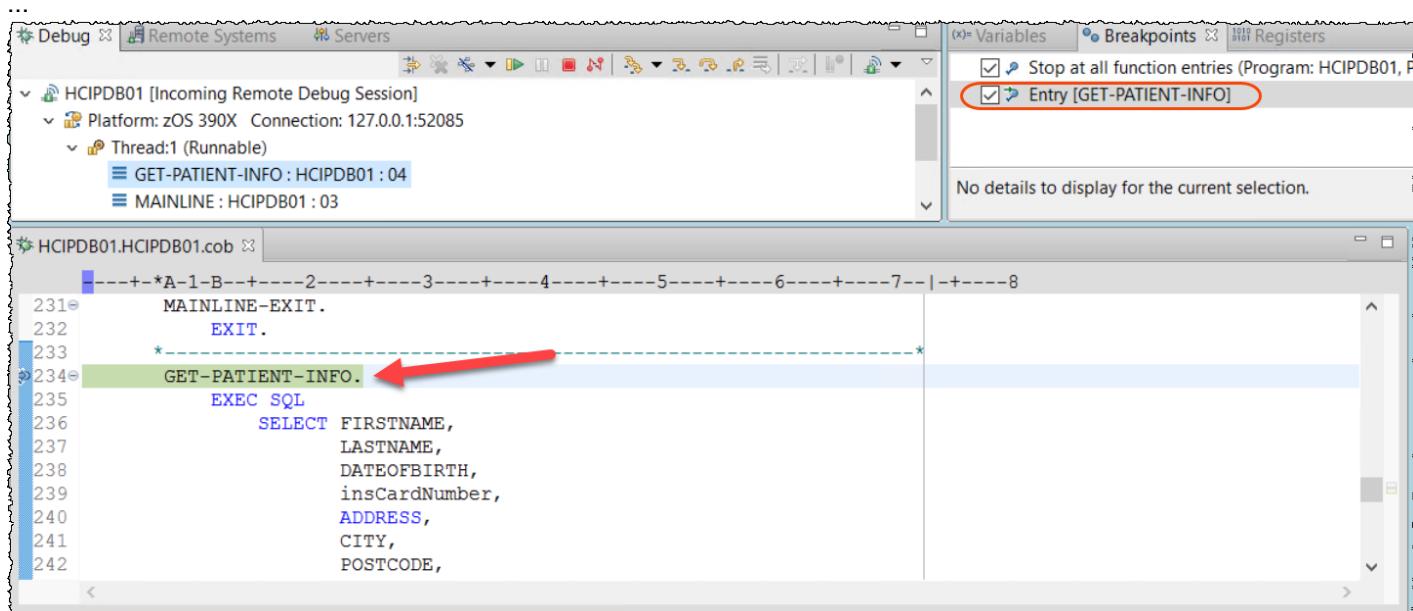
HCPDB01.HCPDB01.cob
-----+---+---+---+---+---+---+---+
231*   MAINLINE-EXIT.
232      EXIT.
233*
234* GET-PATIENT-INFO.
235      EXEC SQL
236      SELECT FIRSTNAME.

```

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



4.4.10 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..

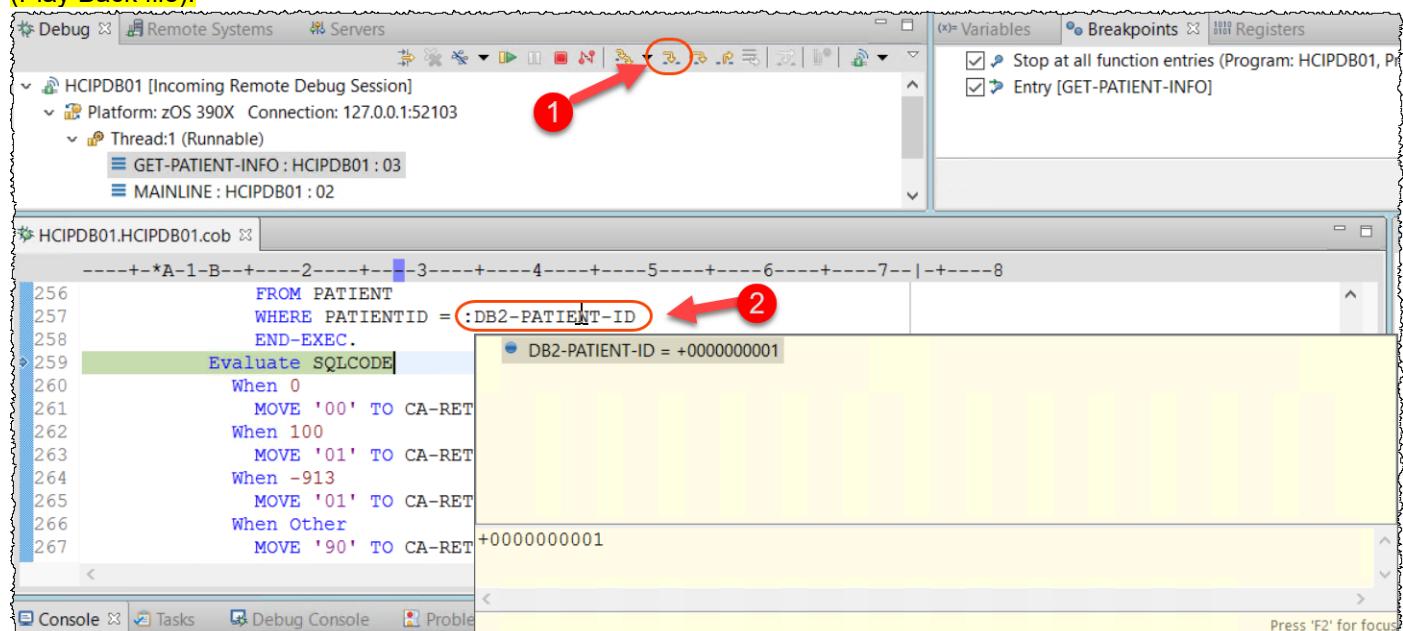


The screenshot shows the IBM Rational Application Developer interface. The top navigation bar includes 'Debug', 'Remote Systems', and 'Servers'. Below the navigation bar, the 'Breakpoints' tab is selected in the toolbar. In the 'Breakpoints' view, there is a checkbox labeled 'Entry [GET-PATIENT-INFO]' which is checked. The main workspace shows a COBOL source code editor with the file 'HCPDB01.HCPDB01.cob'. Line 234 contains the SQL SELECT statement, which is highlighted with a green background. A red arrow points from the 'Breakpoints' view to this highlighted line in the code editor.

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

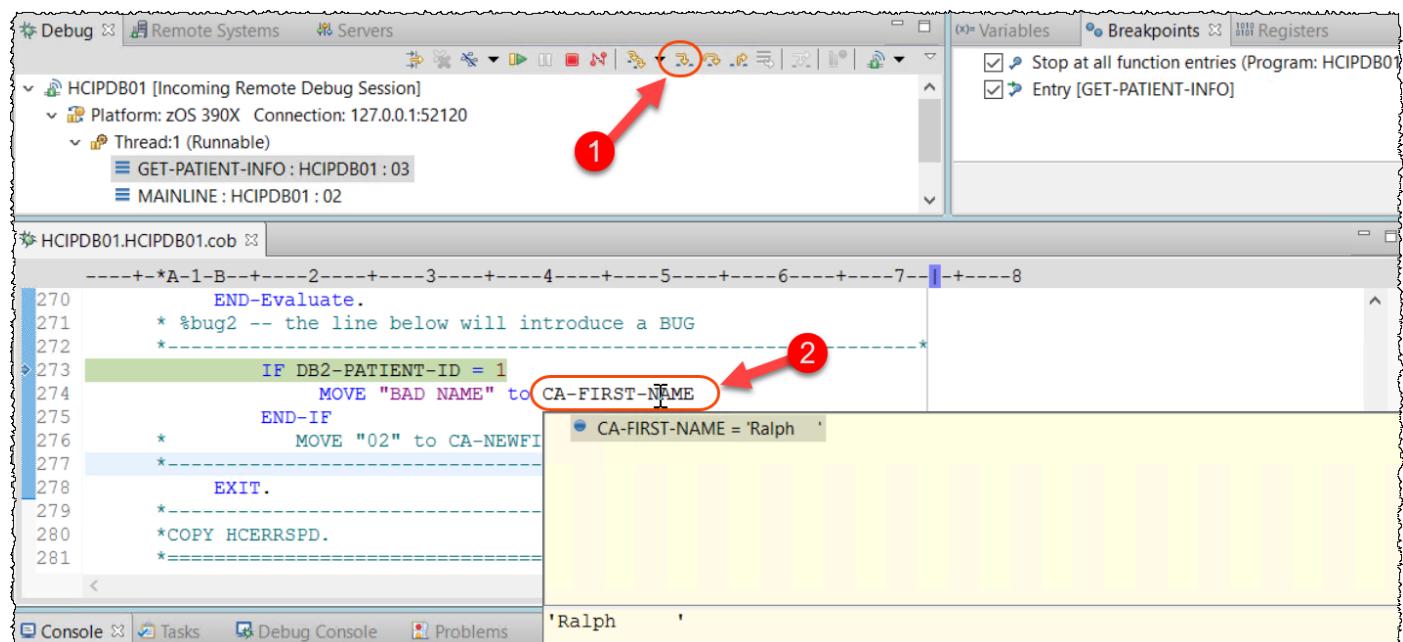
► You can move the mouse to : DB2-PATIENT-ID and see its contents

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



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

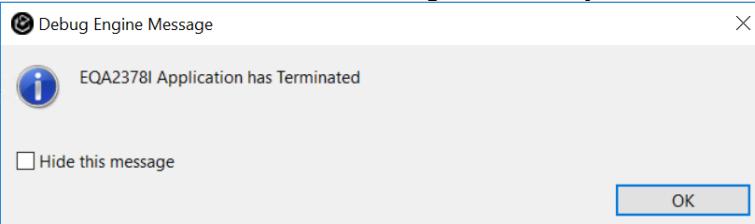
► You also can see the BUG that you introduced. Move the mouse to CA-FIRST-NAME to see the field content (**Ralph**), which will be replaced by **"BAD NAME"**.



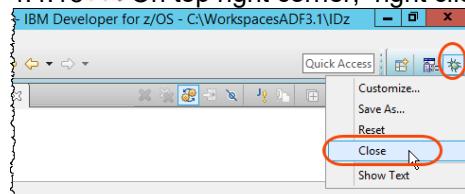
4.4.13 ► Click on icon or press **F8** to resume.. Or if you prefer keep clicking on *Step Into*.



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



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



Section 5. (Optional) 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.

```

//HCIPZRUN JOB ,MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
// Action: Run ZUnit in batch...
// Source: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)
//RUNNER EXEC PROC=BZUPLAY,
// BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),
// BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,
// BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,
// PARM='STOP=E,REPORT=XML'
//BZUPLAY DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.PB1.HCIPDB01
//BZURPT DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)
//
```

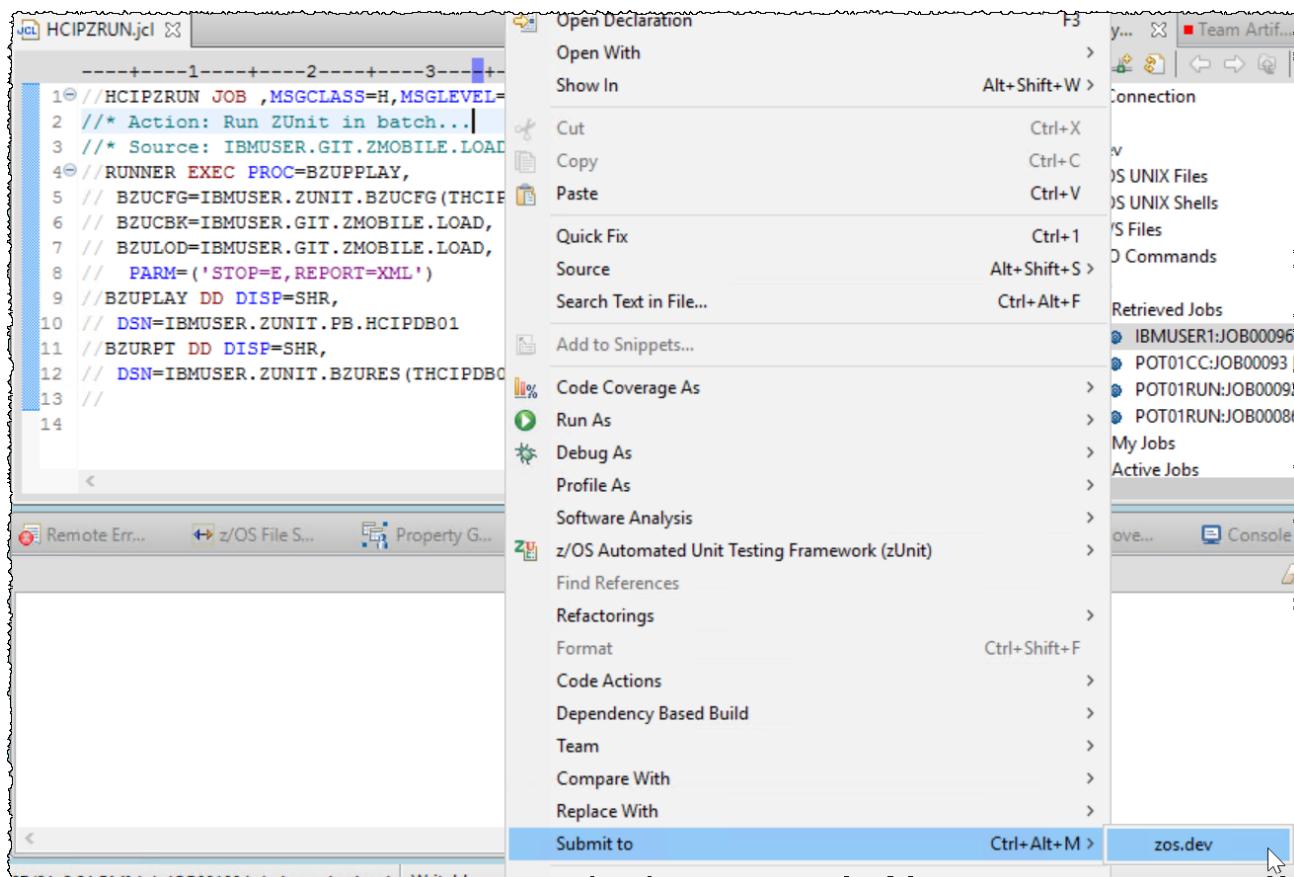
5.1.3 ► Be sure that on the line 10 the dataset name is **IBMUSER.ZUNIT.PB.HCIPDB01** (instead of **PB1**).
You may need to change this and then click **Ctrl+S**

```

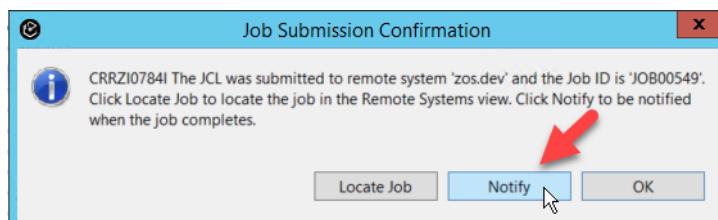
//BZUPLAY DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.PB.HCIPDB01
//BZURPT DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)

```

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

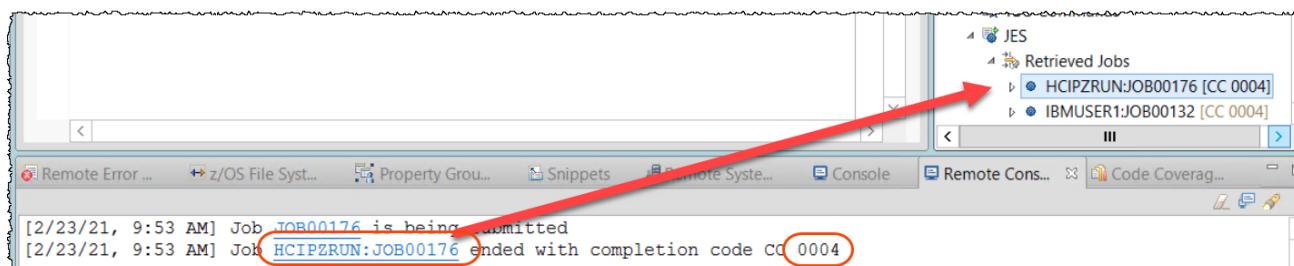


5.1.5 ► Click **Notify** on the Job Submission Confirmation dialog.



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

► Click on **HCIPZRUN:JOB00xxx**



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

```

HCIPZRUN.jcl IBMUSER.HCIPZRUN.JOB00103.D0000104?.spool
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
1 TEST_INQ01 STARTED...
2 CALL HCIPDB01
3 DB2_INPT ...
4 DB2_OUTP ...
5 CICS_OE08_HCIPDB01 CHECK VALUES...
6 EXEC CICS RETURN X'0000' L=00230
7 AREA ALLOCATED FOR RECORD COUNT:00000048
8 CICS_OE08_HCIPDB01 SUCCESSFUL.
*****AZU2001W THE TEST "INQ01" FAILED DUE TO AN ASSERTION.
11 AZU1101I COMPARE FAILED IN PROCEDURE DIVISION.
12 DATA ITEM NAME : CA-FIRST-NAME OF CA-PATIENT-REQUEST OF DFHCOMMAREA
13 VALUE : BAD NAME
14 EXPECTED VALUE: Ralph
*****TEST_INQ01 SUCCESSFUL.
16
17

```

Remote Systems Team Artifacts

- IBMUSER.GIT.ZMOBILE.LOAD
- IBMUSER.GIT.ZMOBILE.MACRO
- IBMUSER.GIT.ZMOBILE.MFS
 - ...163 more
- EMPOT05.*
- EMPOT01.*
- My Favorites
- TSO Commands
- JES
 - Retrieved Jobs
 - HCIPZRUN:JOB00103 [CC 0004]
 - JES2::JESMSGGL
 - JES2::JESJCL
 - JES2::JESYMSG
 - RUNNER:REPLAY:BZUMSG [000]
 - RUNNER:REPLAY:SYSOUT
 - RUNNER:REPLAY:SYSOUT

5.1.8 This JCL could be executed using DBB as 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**

```

RunZUnitJCL.groovy
import com.ibm.dbb.build.CopyToHFS
import com.ibm.dbb.build.DBBConstants
import com.ibm.dbb.build.JCLExec
// ****
5 * Changed Dec 27, 2019 by Regi
6 * The following sample shows how to use JCLExec API to execute a ZUnit and
7 * display the results in the console.
8 * This sample assumes that user has setup the ZUnit and a JCL to execute the
9 * ZUnit.
10 * This sample requires:
11 *   1. The data set contains the JCL.
12 *   2. The data set contains the output of the ZUnit result.
13 * Sample output:
14 *   Running ZUnit in JCL 'IBMUSER.ZUNIT.JCL(ZUNIDB01)'
15 *   The JCL Job completed with Max-RC CC 0004
16 *   ***** Module [J05CMORT] *****
17 * zUnit Test Runner 2.0.0.1 started at 2019-11-06T13:57:22.885...
18 * Test count: 1
19 * Tests passed: 1
20 * Tests failed: 0
21 * Tests in error: 0
22 *
23 ****
24
25 /* DBB_CONF must be set for running JCLExec */
26 /* def confDir = System.getenv("DBB_CONF") */
27 /* added by regi - was above statement only before */
28 def confDir = "/var/dbb/1.0.7/conf"
29
30 /* The data set contains the ZUnit JCL */
31 /* For example: IBMUSER.ZUNIT.JCL */
32 def jclDataset = "IBMUSER.ZUNIT.JCL"

```

Remote Systems Team Artifacts

- zos.dev
 - z/OS UNIX Files
 - My Home
 - Root
 - /var/zosconnect/servers/server30
 - groovy_samples
 - ADMIN.pw
 - BMSProcessing.groovy
 - build.groovy
 - build.properties
 - build.sh
 - CobolCompile.groovy
 - CobolCompile.groovy1.0.1
 - CobolCompile_OLD.groovy
 - Compile.groovy
 - datasets.properties
 - deploy.groovy
 - deploy.sh
 - deploy_OLD.groovy
 - file.properties
 - files.txt
 - impacts.groovy
 - JCLgroovy
 - LinkEdit.groovy
 - MFSGENUtility.groovy
 - RunZUnitJCLgroovy
 - Tools.groovy

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

Congratulations! You have completed the Lab 6C.

LAB 6D – (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

DB2BATCH.cbl

```

121 EXEC SQL
122   DECLARE C1 CURSOR FOR
123     SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF),
124       MIN(HOURS), MAX(HOURS), AVG(HOURS)
125     FROM RBAROSA.EMPL_E, RBAROSA.PAY_P
126     WHERE E.NBR = P.NBR
127     AND PERF > :PERF
128     GROUP BY DEPT
129   END-EXEC.
130*
131   EXIT.
132*
133 * THIS STATEMENT OPENS THE "ACTIVE SET" IN PREPARATION OF
134 * ROW FETCH PROCESSING.
135   EXEC SQL OPEN C1
136   END-EXEC.
137*
138   EXIT.
139*
140 * THIS PARAGRAPH SETS UP THE SQL PARAMETERS, PERFORMS THE
141 * PARAGRAPH TO FETCH THE ROW, AND PRINT THE RESULTS.
142*
143   PERFORM 250-FETCH-A-ROW THRU 250-EXIT.
144   ----- Added by REGI just for demo content of SQLCA
145   MOVE SQLCA TO W-SQLCA.
146   IF SQLCODE = ZERO
147     THEN
148       MOVE DEPT-TBL TO DEPT-RPT
149       MOVE PERF-TBL-AVG TO PERF-RPT-AVG
150       MOVE PERF-TBL-MIN TO PERF-RPT-MIN
151       MOVE PERF-TBL-MAX TO PERF-RPT-MAX
152       MOVE HOURS-TBL-AVG TO HOURS-RPT-AVG
153       MOVE HOURS-TBL-MAX TO HOURS-RPT-MAX
154       MOVE HOURS-TBL-MIN TO HOURS-RPT-MIN
155*
156   * invoke called proc to
157   *      WRITE REPORT-LINE FROM DETAIL-LINE
158   *      AFTER ADVANCING 1 LINES
159   * IF YES a Report will be printed by DB2PRT
160   IF PRINT-REPORT = 'YES'
161     MOVE "DB2PRT" TO PROGRAM-TO-CALL
162     CALL PROGRAM-TO-CALL USING DETAIL-LINE,
163                                     RECEIVED-FROM-CALLED
164   END-IF
165*

```

DB2PRT.cbl

```

52*
53*
54*
55*
56*
57*
58*
59*
60*
61*
62*
63*
64*
65*
66*
67*
68*
69*
70*
71*
72*
73*
74*
75*
76*
77*
78*

```

IBMUSER.BATCHRUNJOB00097.D0000106.?spool

1	*DEPT*	*PERF-AVG*	*PERF-MIN*	*PERF-MAX*	*HOURS-AVG*	*HOURS-MIN*	*HP
1	ACC	.00	8.00	8.00	15.99	15.99	1
2	FIN	.00	3.00	9.00	22.69	32.45	
3	MKT	.00	1.00	3.00	22.82	32.41	1
4	R&D	.00	1.00	1.00	22.82	32.41	1
5	REG	.00	9.00	9.00	26.75	26.75	2
6	N/A	.00	.00	.00	35.49		
7							
8							

Scenario: COBOL Batch/DB2

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**

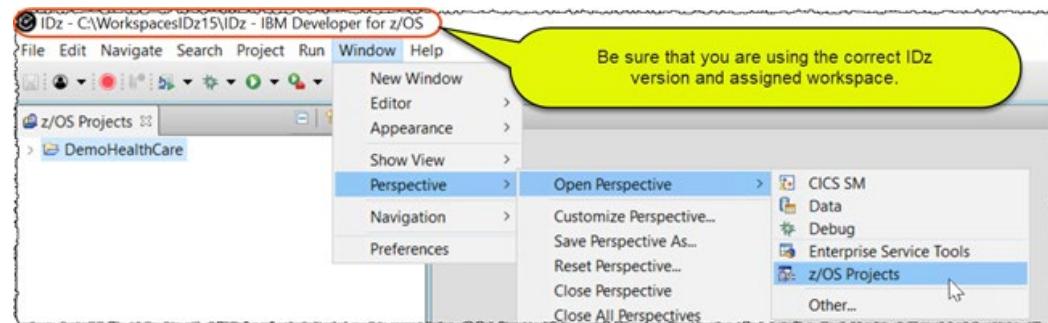
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.



1.1 Submit a provided JCL for execution

You will use IDz to submit a provided JCL .

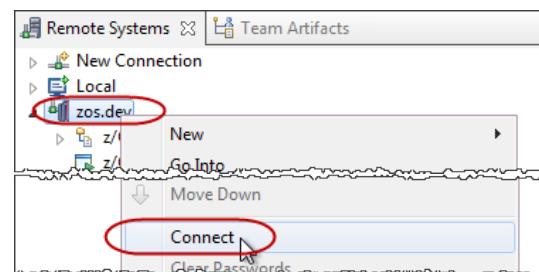
- 1.1.1 ► Open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



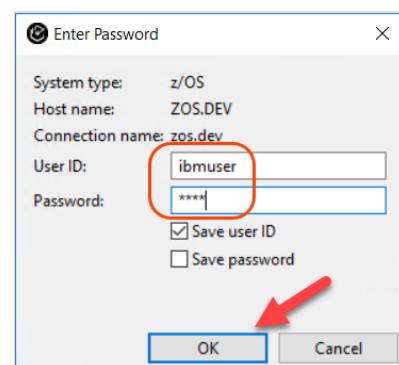
Nothing will happen you are already at this [perspective].

- 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.

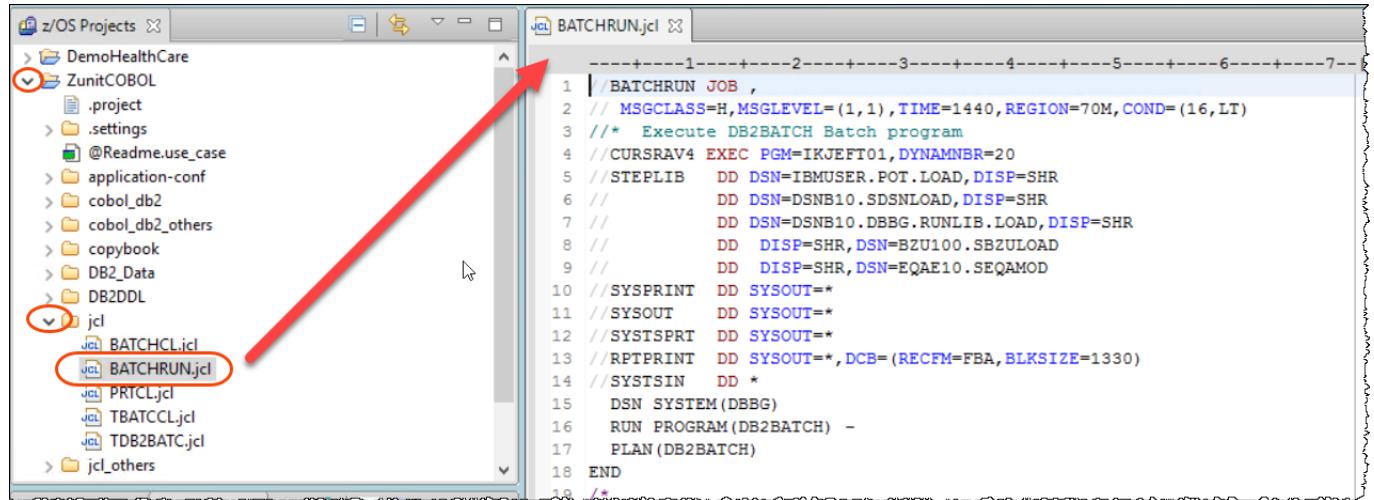
- Using *Remote Systems* view, right click on **zos.dev** and select **Disconnect** and then right click on **zos.dev** again and select **Connect**



- 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.



```

z/OS Projects X
> DemoHealthCare
> ZunitCOBOL
  .project
  .settings
  @Readme.use_case
  application-conf
  cobol_db2
  cobol_db2_others
  copybook
  DB2_Data
  DB2DDL
  jcl
    JCL BATCHCL.jcl
    JCL BATCHRUN.jcl (highlighted)
    JCL PRCL.jcl
    JCL TBATCCL.jcl
    JCL TDB2BATC.jcl
  jcl_others

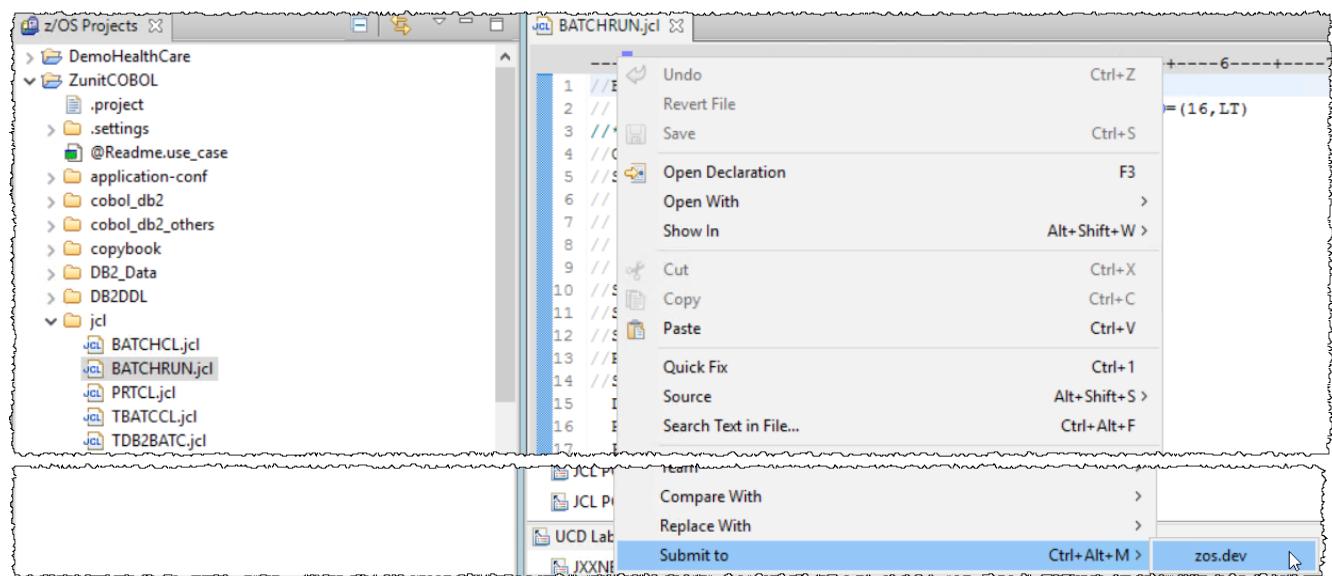
```

```

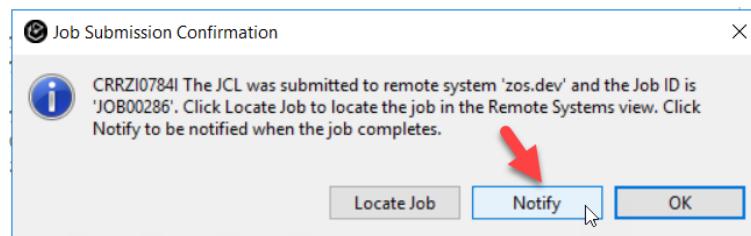
JCL BATCHRUN.jcl X
-----+-----+-----+-----+-----+-----+-----+-----+
1 //> BATCHRUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 /* Execute DB2BATCH Batch program
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=IBMUSER.POT.LOAD,DISP=SHR
6 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
7 //          DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
8 //          DD DISP=SHR,DSN=BZU100.SBZULOAD
9 //          DD DISP=SHR,DSN=EQAE10.SEQAMOD
10 //SYSPRINT DD SYSOUT=*
11 //SYSOUT DD SYSOUT=*
12 //SYSTSPRT DD SYSOUT=*
13 //RPTPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
14 //SYSTSIN DD *
15 DSN SYSTEM(DBDBG)
16 RUN PROGRAM(DB2BATCH) -
17 PLAN (DB2BATCH)
18 END

```

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

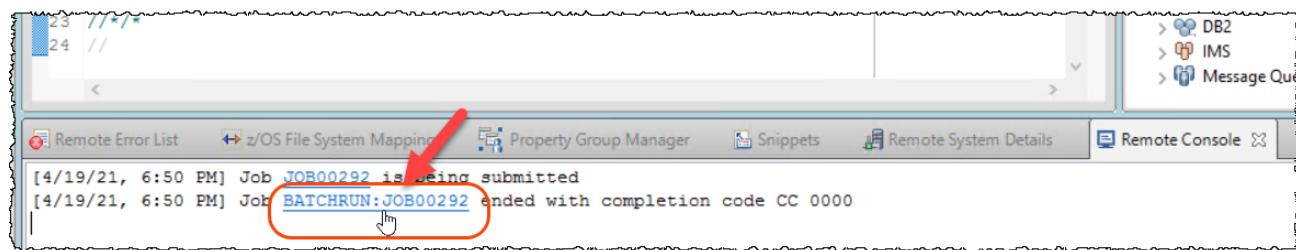


- 1.1.6 ► Click **Notify** to be notified when the execution is complete.



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 00xxx 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**.

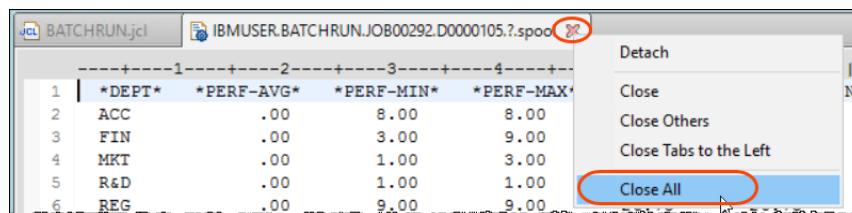
The screenshot shows the 'Remote Systems' interface. On the left, a terminal window displays a DB2PRT report with the following data:

	DEPT	*PERF-AVG*	*PERF-MIN*	*PERF-MAX*	*HOURS-AVG*	*HOURS-MIN*	*HOURS-MAX*
1	*DEPT*	.00	8.00	8.00	15.99	15.99	15.99
2	ACC	.00	8.00	8.00	15.99	15.99	15.99
3	FIN	.00	3.00	9.00	22.69	32.45	8.89
4	MKT	.00	1.00	3.00	22.82	32.41	13.23
5	R&D	.00	1.00	1.00	22.82	32.41	13.23
6	REG	.00	9.00	9.00	26.75	26.75	26.75
7	N/A	.00	.00	35.45	35.45	35.45	

A yellow callout bubble points to the 'ACC' value in the first row with the text: "Notice on first line the DEPT value of ACC. When you introduce a bug this value will be different."

On the right, the 'Retrieved Jobs' tree view is expanded to show the 'CURSRAV4::RPTPRINT' step, which is also circled in red.

- 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..

```

z/OS Projects [x]          DB2BATCH.cbl [x]
> DemoHealthCare
< ZunitCOBOL
  .project
  .settings
  @Readme.use_case
  application-conf
< cobol_db2
  DB2BATCH.cbl (highlighted)
  DB2PRT.cbl
  cobol_db2_others
  copybook
  DB2_Data
  DB2DDL
< jcl

-----+-----+-----+-----+-----+-----+-----+-----+
1  IDENTIFICATION DIVISION.
2  PROGRAM-ID. DB2BATCH.
3  *REMARKS. THIS PROGRAM JOINS TABLES, GROUPS DATA BY DEPT,
4  * AND PRINT THE AVERAGE, MAXIMUM AND MINIMUM
5  * HOURS, AND PERFORMANCE EVALUATION BY DEPT.
6  * Modified by Regi to pass report via COMMAREA Apr 06,2021
7  * pass line to be printed via CALL to DB2PRT
8  * Modified by Regi to use PRINTER instead of DISPLAY Mar 26,2021
9  * Modified by Regi to add DEAD CODE - Jan/2-14
10 * Modified by Regi to added test if -204 - Mar/2015
11 ENVIRONMENT DIVISION.
12 DATA DIVISION.
13
14 WORKING-STORAGE SECTION.
-----+-----+-----+-----+-----+-----+-----+-----+

```

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 [x]          DB2BATCH.cbl [x]
> DemoHealthCare
< ZunitCOBOL
  .project
  .settings
  @Readme.use_case
  application-conf
< cobol_db2
  DB2BATCH.cbl (highlighted)
  DB2PRT.cbl
  cobol_db2_others
  copybook
  DB2_Data
  DB2DDL
< jcl
  BATCHCL.jcl
  BATCHRUN.jcl
  PRTCL.jcl
  TBATCL.jcl

Properties [x]             Outline [x]
  PROCEDURE DIVISION.
    001-MAIN-ROUTINE.
    000-EXIT.
    100-DECLARE-CURSOR-RTN. (highlighted)
    100-EXIT.

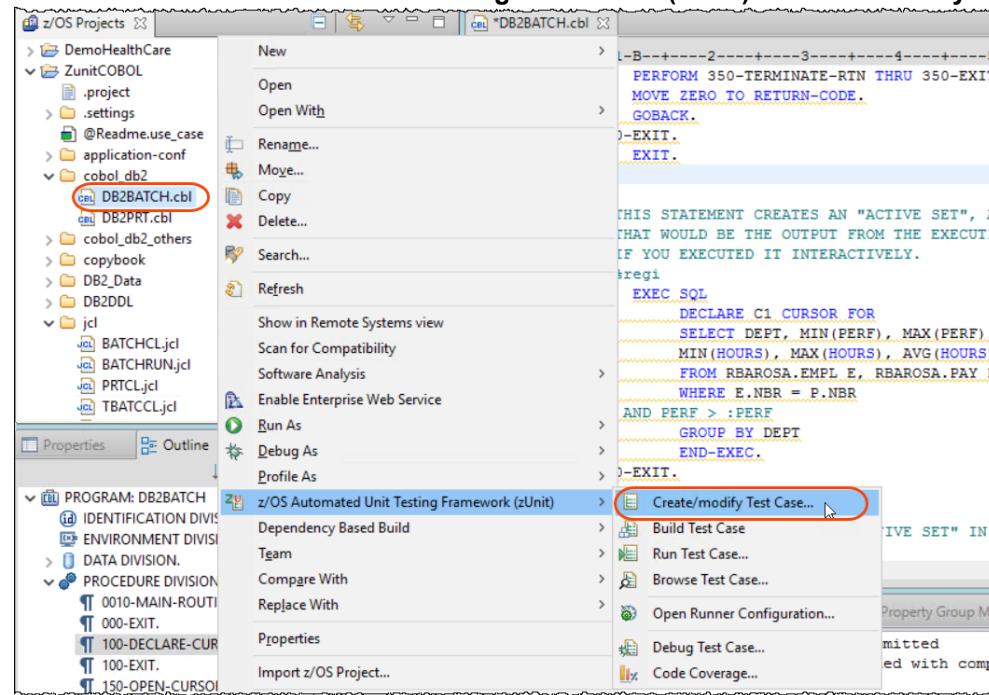
-----+-----+-----+-----+-----+-----+-----+-----+
108  UNTIL SQLCODE = +100.
109    PERFORM 300-CLOSE-CURSOR-RTN THRU 300-EXIT.
110    PERFORM 350-TERMINATE-RTN THRU 350-EXIT.
111    MOVE ZERO TO RETURN-CODE.
112    GOBACK.
113    000-EXIT.
114    EXIT.
115  100-DECLARE-CURSOR-RTN. (highlighted)
116    * THIS STATEMENT CREATES AN "ACTIVE SET", A GROUP OF ROWS
117    * THAT WOULD BE THE OUTPUT FROM THE EXECUTION OF THE STATEMENT
118    * IF YOU EXECUTED IT INTERACTIVELY.
119    * $regi
120      EXEC SQL
121        DECLARE C1 CURSOR FOR
122          SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF),
123          MIN(HOURS), MAX(HOURS), AVG(HOURS)
124          FROM RBAROSA.EMPL E, RBAROSA.PAY P
125          WHERE E.NBR = P.NBR
126          * AND PERF > :PERF
127          * GROUP BY DEPT
128          * END-EXEC.
129
130  100-EXIT.
131    EXIT.
132  150-OPEN-CURSOR-RTN.

-----+-----+-----+-----+-----+-----+-----+-----+

```

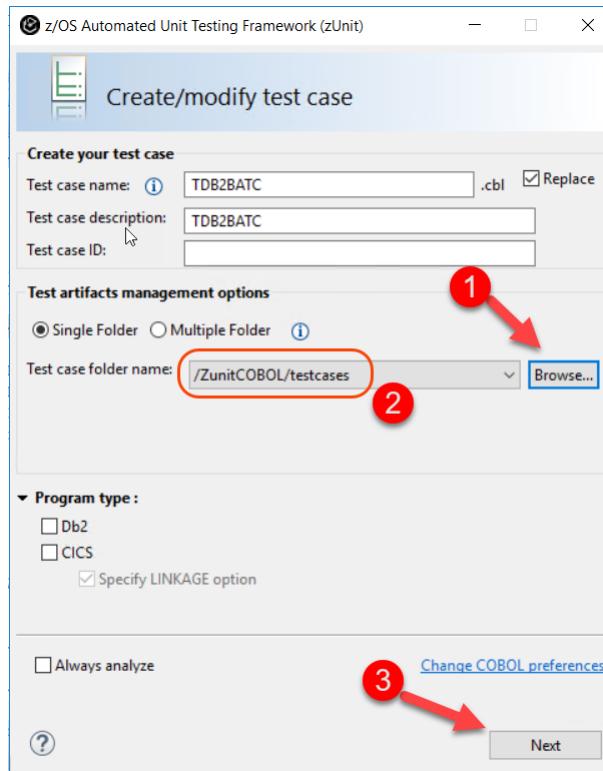
2.2 Recording the batch JCL execution

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..**

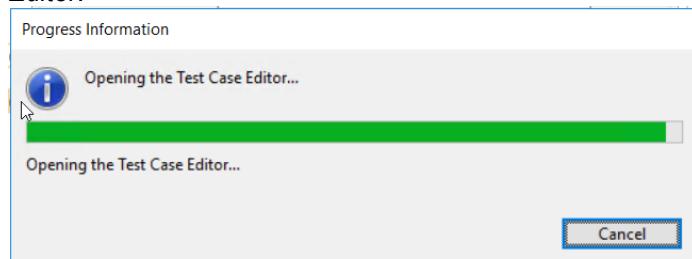


2.2.2 This opens a dialog where you can name your test case.

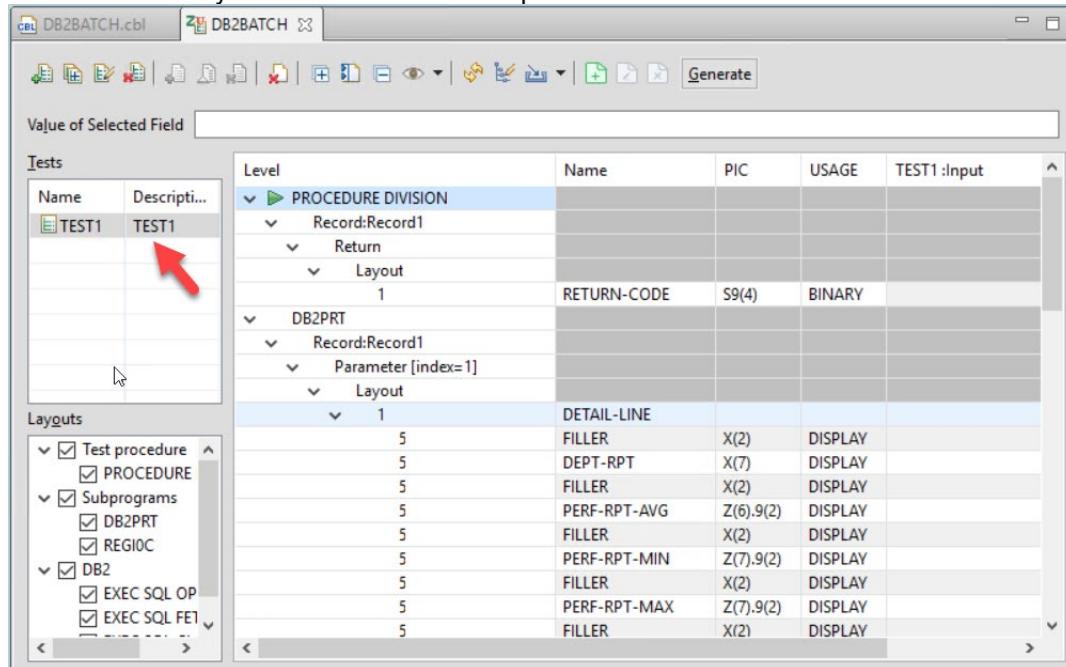
► Using the Browse button select **/ZunitCOBOL/testcases** as folder name and click **Next**.



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.

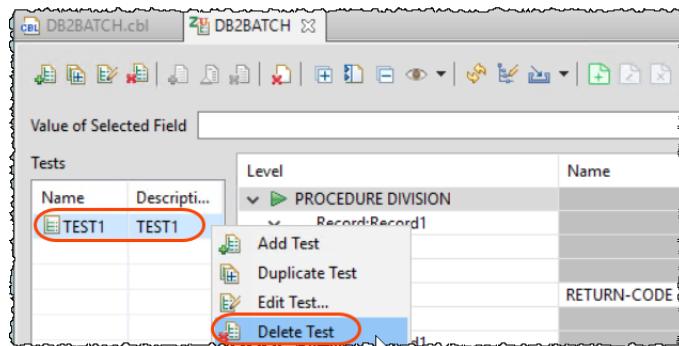


The test entry Input and Expected output columns represent the flow of data into and out of the main program, that is, the program being tested by zUnit. When data is added to these columns in a subroutine, the flow of data is:

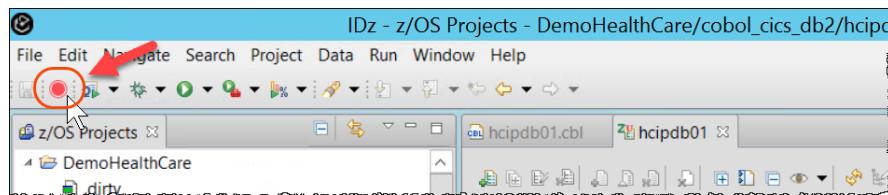
Input: data passed into the main program: that is, what is passed to the main program when the subroutine completes execution.

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.

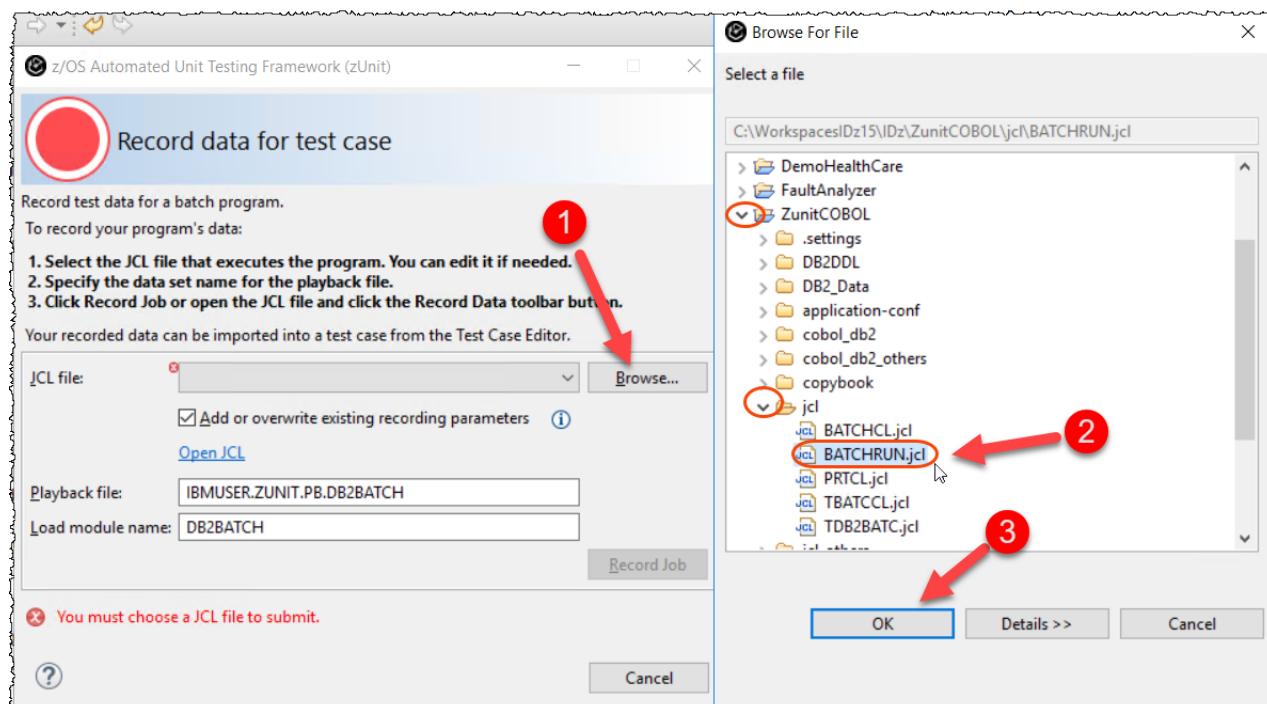
2.2.5 ► Since we will be recording the program execution, delete the **TEST1** or any other entry (if it exists) by right clicking and selecting **Delete Test**



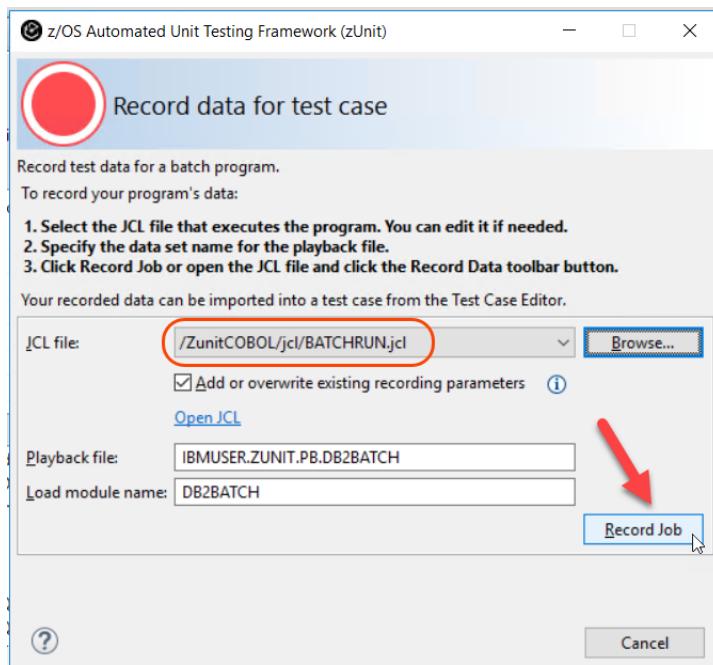
2.2.6 ► To record from a JCL batch execution into the test case, click the **Record** button on the IDz toolbar.



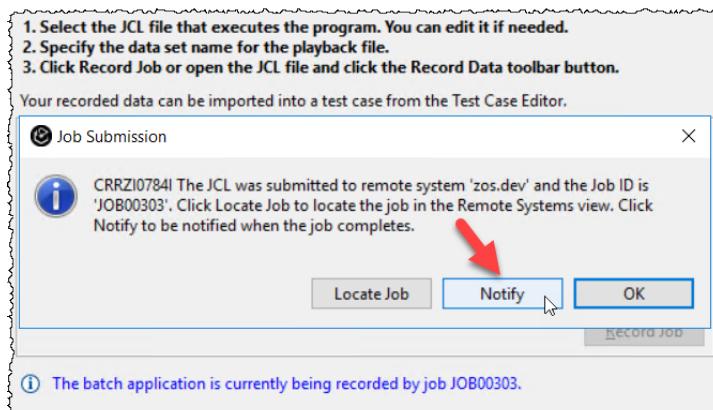
2.2.7 ► In the dialog that comes up, use the **Browse** button to select **BATCHRUN.jcl** under **ZunitCOBOL/jcl** and click **OK**.



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

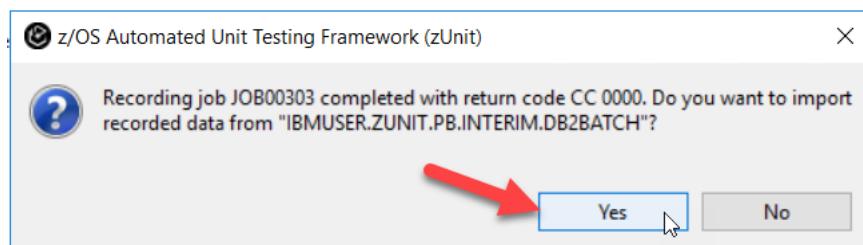


2.2.9 ► Click **Notify** for the dialog below. The JCL will be submitted to run on z/OS.

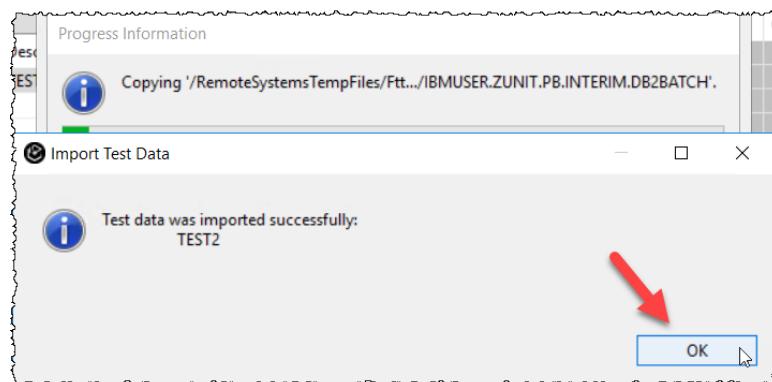


2.2.10 When the job is completed the dialog below is displayed.

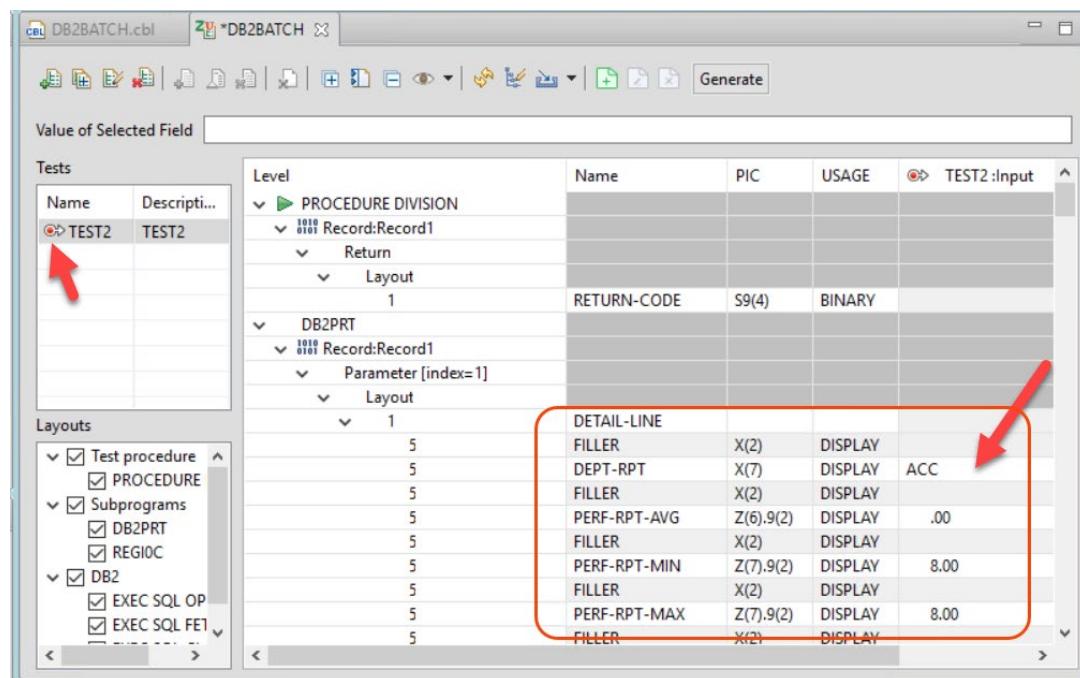
► Click **Yes** to create the playbackfile and import the test data.



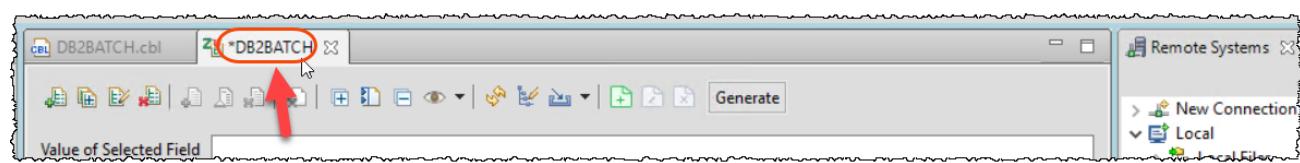
2.2.11 ► Click **OK** after the data is successfully imported to the test data and playback file.



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.

Level	Name	PIC	USAGE	TEST2:Input	TEST2:Expect...
1	DETAIL-LINE				
5	FILLER	X(2)	DISPLAY		
5	DEPT-RPT	X(7)	DISPLAY	REG	REG
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-AVG	Z(6),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MIN	Z(7),9(2)	DISPLAY	9.00	9.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MAX	Z(7),9(2)	DISPLAY	9.00	9.00
5	FILLER	X(2)	DISPLAY		
5	HOURS-RPT-AVG	Z(7),9(2)	DISPLAY	26.75	26.75
5	FILLER	X(2)	DISPLAY		
5	HOURS-RPT-MAX	Z(7),9(2)	DISPLAY	26.75	26.75
5	FILLER	X(5)	DISPLAY		
5	HOURS-RPT-MIN	Z(7),9(2)	DISPLAY	26.75	26.75
3	RECEIVED-FROM-C...	9(2)	DISPLAY	0	0
1	DETAIL-LINE				
5	FILLER	X(2)	DISPLAY		
5	DEPT-RPT	X(7)	DISPLAY	N/A	N/A
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-AVG	Z(6),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MIN	Z(7),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MAX	Z(7),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		

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**

Level	Name	PIC	USAGE	TEST2:Input	TEST2:Expect...
1	EXEC SQL OPEN [C1]	line=135			
1	Record:Record1				
1	LineNumber=135				
1	SQLCA				
1	EXEC SQL FETCH [C1]	line=199			
1	Record:Record1				
1	LineNumber=199				
1	INTO				
5	DEPT-TBL	X(3)	DISPLAY	ACC	
5	DEPT-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MIN	S9(4)	BINARY	8	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MAX	S9(4)	BINARY	8	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-AVG	S9(5)V9(2)	PACKED...	8.00	
5	PERF-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-MIN	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-MAX	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-AVG	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
1	SQLCA				
1	Record:Record2				
1	LineNumber=199				
1	INTO				
5	DEPT-TBL	X(3)	DISPLAY	FIN	
5	DEPT-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MIN	S9(4)	BINARY	3	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MAX	S9(4)	BINARY	9	
5	PERF-NULL	S9(4)	BINARY	0	

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

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

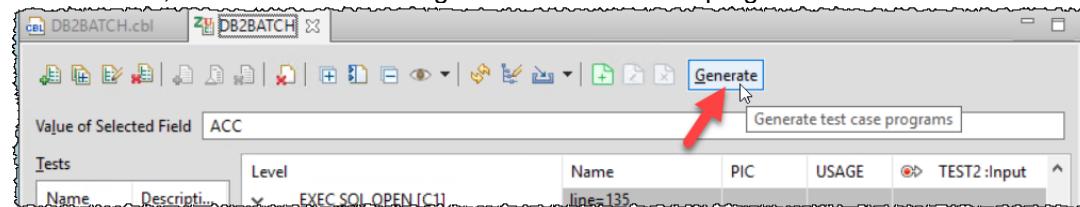


Section 3. Generate, build, and run the unit test generated program.

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

3.1 Generating the COBOL test case programs.

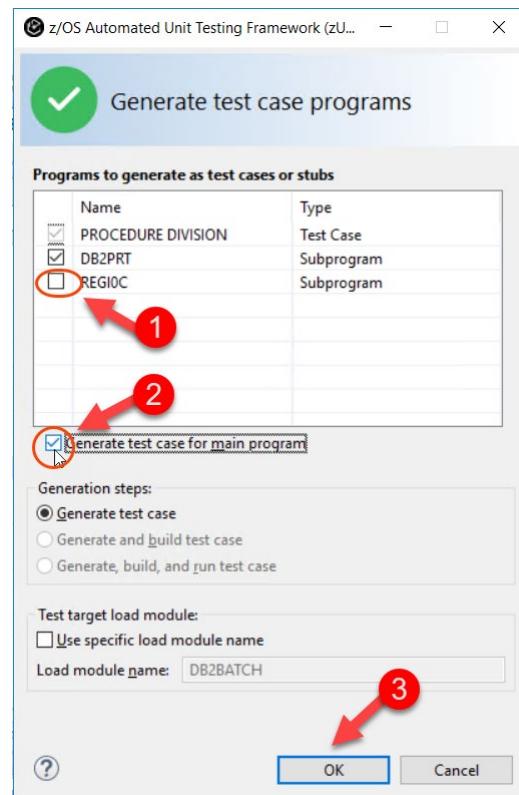
3.1.1 ►► Now that the expected input and output values have been set in the test case editor from the recorded run, click on **Generate** to generate the test case programs.



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

3.1.2 ►► Un-select **REGI0C** (you don't need stubs for this subprogram since it 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.

The screenshot shows the IBM Rational Developer for z/OS interface. In the top right, there are three tabs: DB2BATCH.cbl, DB2BATCH, and TDB2BATC.cbl. The TDB2BATC.cbl tab is active, displaying a COBOL source code listing. The code includes sections like PROCESS, IDENTIFICATION DIVISION, and DATA DIVISION. In the bottom left, the Outline view is open, showing a list of generated programs: TEST_TEST2, BZU_TEST, BZU_INIT, BZU_TERM, PGM_DB2PRT, GTMEMRC, and AZU_GENERIC_DB2. A red box highlights this list, and a red arrow points from the z/OS Projects view (left side) to the Outline view.

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

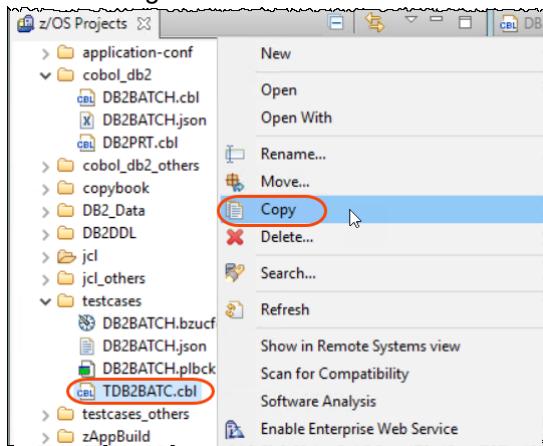
The 7 generated COBOL programs need to be compiled/link edited to be executed. This must be done on z/OS using the COBOL compiler.

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 compiled and linked. Here we will show the traditional JCL way.

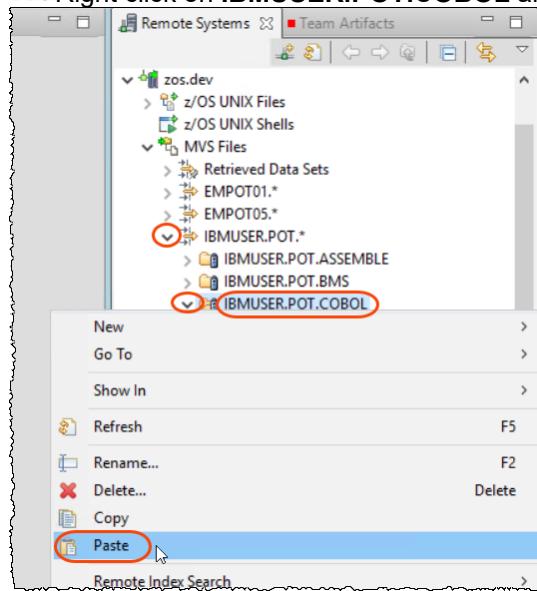
3.2.1 ► Use **Ctrl + Shift + F4** to close all opened editors.

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

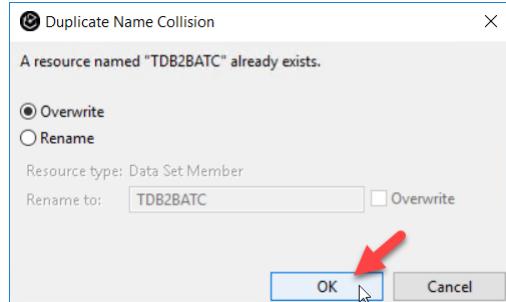


3.2.3 ► Using the Remote systems (on right), expand **IBMUSER.POT.*** and **IBMUSER.POT.COBOL**

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

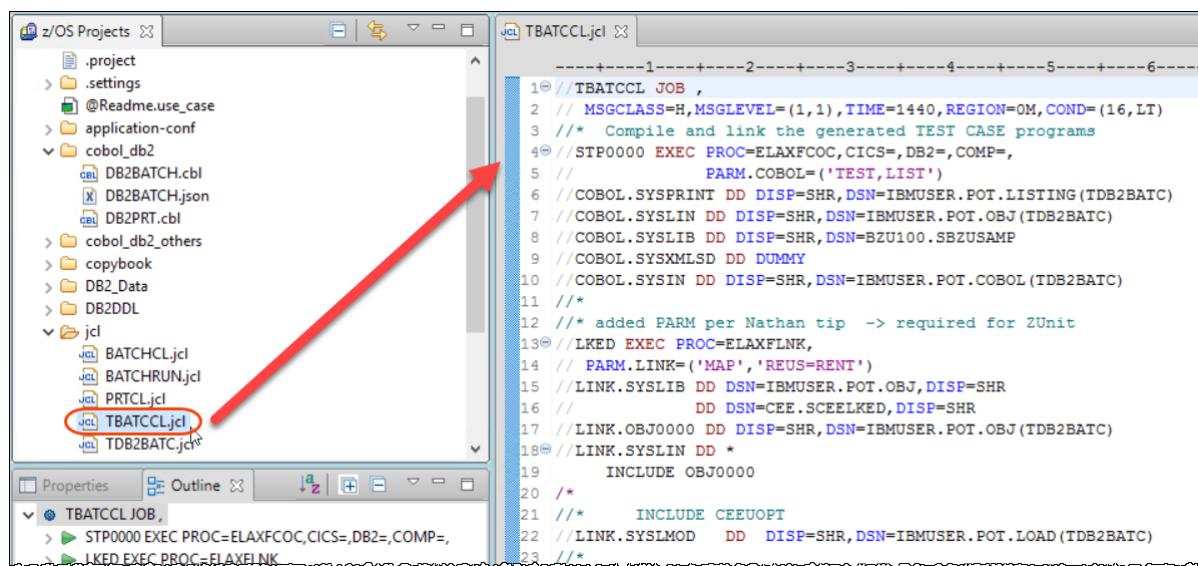


3.2.4 ► Select **OK** to overwrite this member since it already existed on z/OS

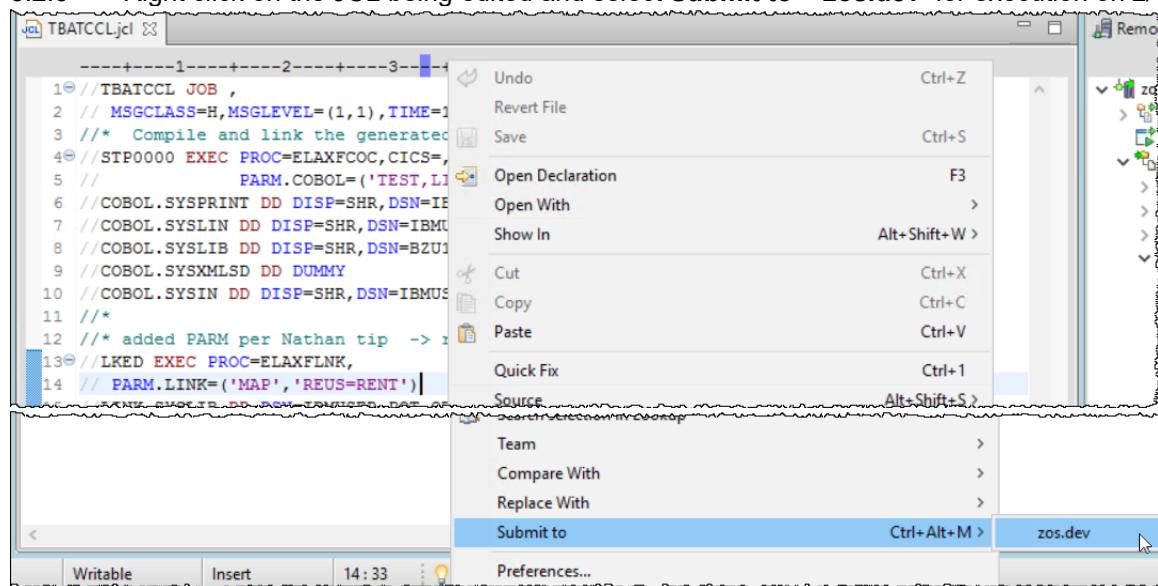


3.2.5 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **TBATCCL.jcl**.

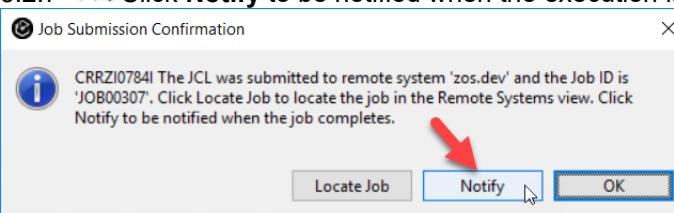
This JCL will compile and link the 7 programs that will create the test case load module.



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

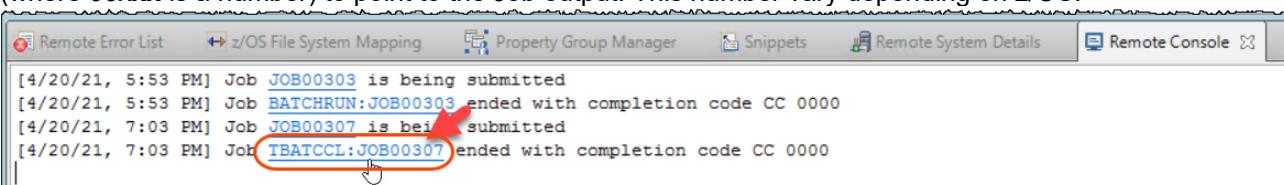


3.2.7 ► Click **Notify** to be notified when the execution is complete.



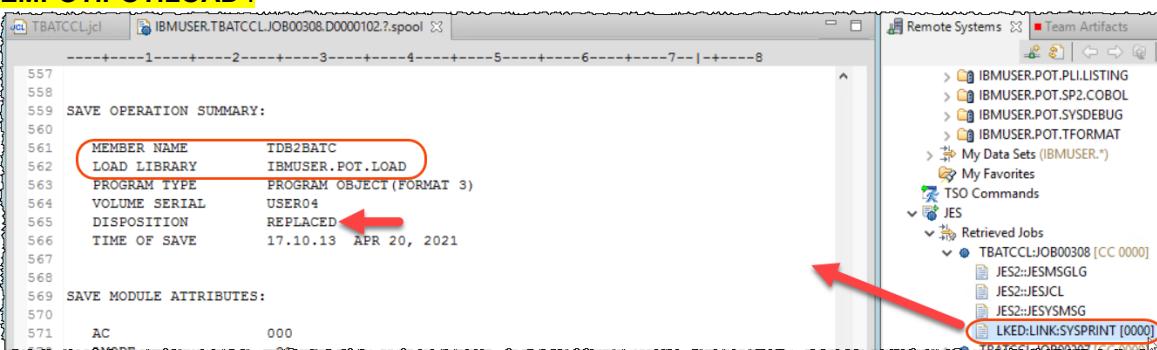
3.2.8 Under *Remote Console*, you will be notified when execution is completed.

► Once the execution ends, **click on the link TBATCCL:JOB00xxx** (where 00xxx 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?



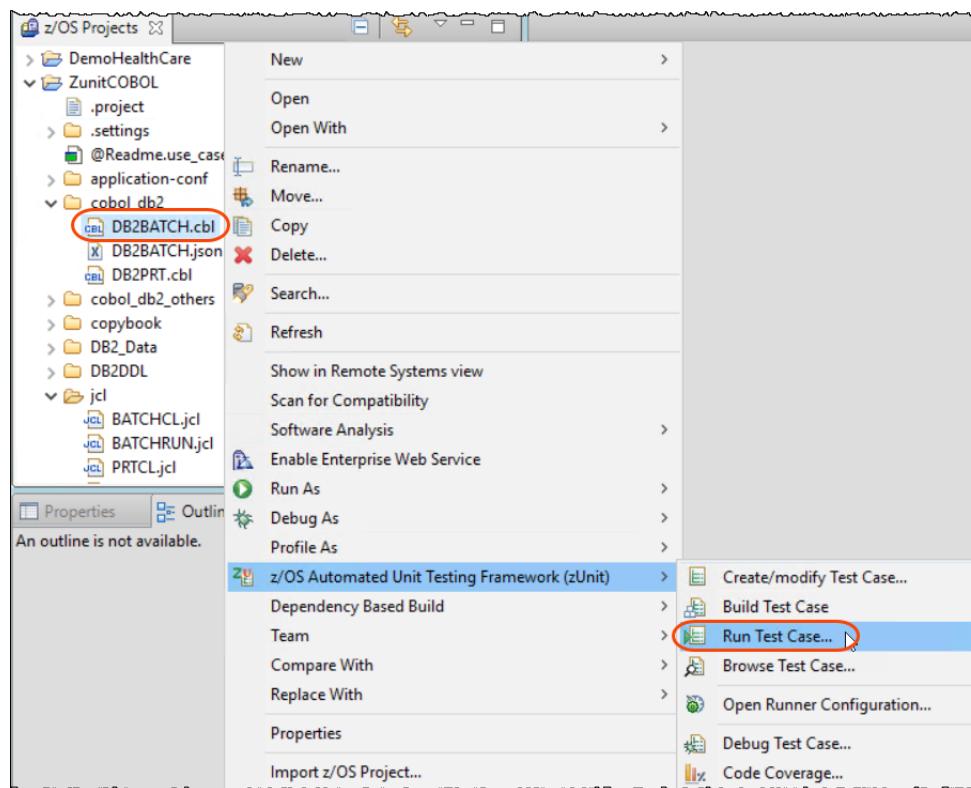
You generated a test case load module (**TDB2BATC**). This load module when executed can verify the correct input and output values handled by the program being verified. The test case can run each time that the program is modified, and the test case must pass it. You will see that

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,

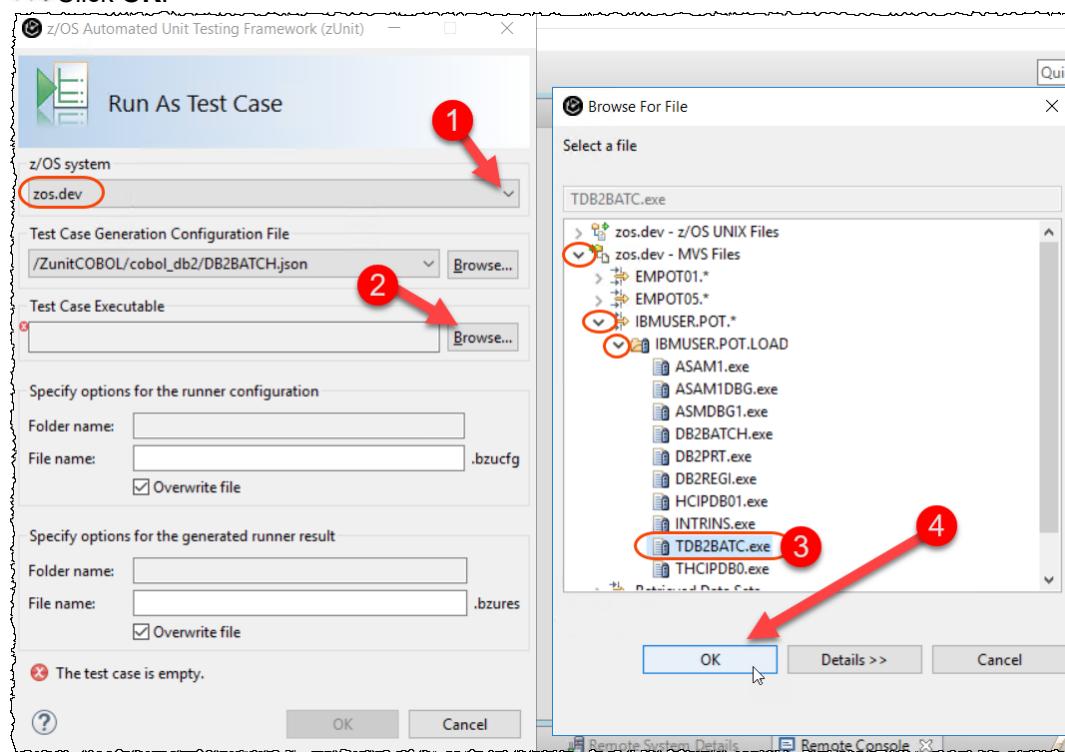
Once you have the test cases load module created you can run the test case against the *DB2BATCH* COBOL program. Since you did not make changes to the program the return code must be zero and all tests should pass.

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...**

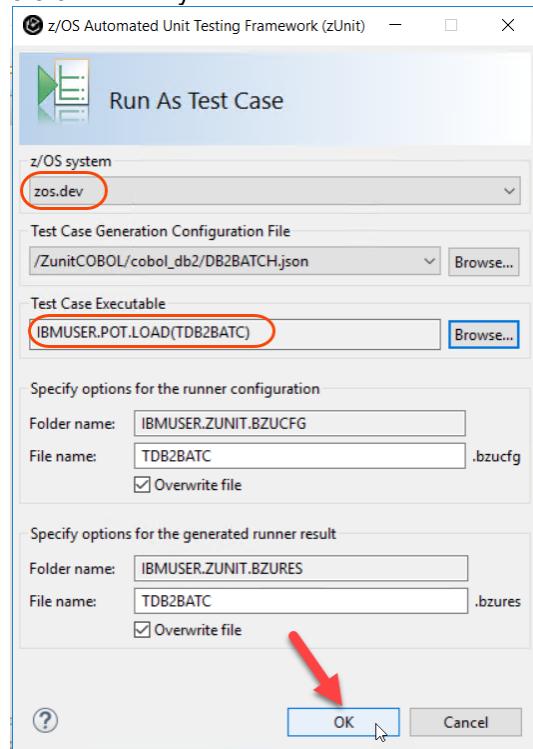


3.3.2 ►| Select **zos.dev** for z/OS system,
Use the second **Browse** button to select **IBMUSER.POT.LOAD (TDB2BATC)**
under **MVS Files** and **IBMUSER.POT.***.

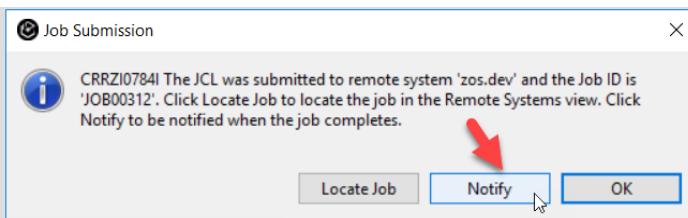
►| Click **OK**.



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.

Statistic	Value
Test count	1
Tests passed	1
Tests failed	0
Tests with errors	0
Tests with severe errors	0

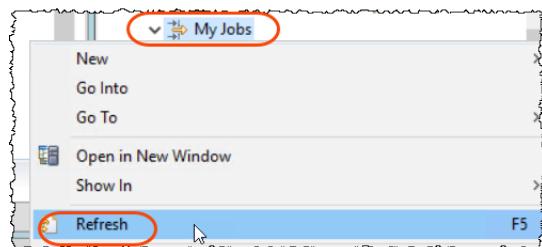
You have now created a test case with data imported from a recorded run and have been successful in testing a COBOL/DB2 batch program without the need of the DB2 environment.

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

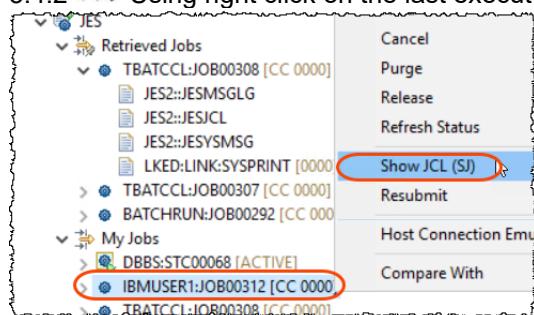
3.4 Verify the JCL submitted that runs the test case

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

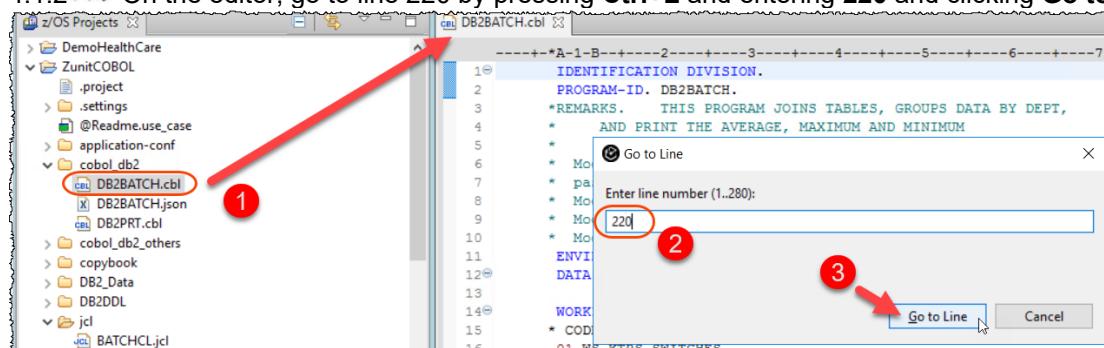
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

4.1 Modifying the program and introduce a bug

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

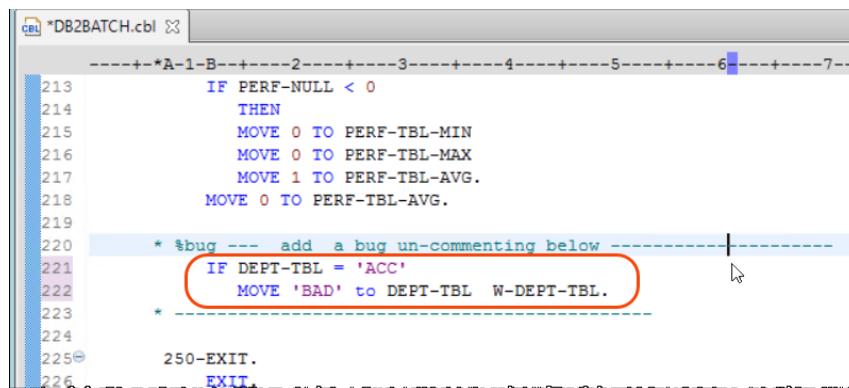
► Open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it in the **z/OS Projects** view.

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.



```

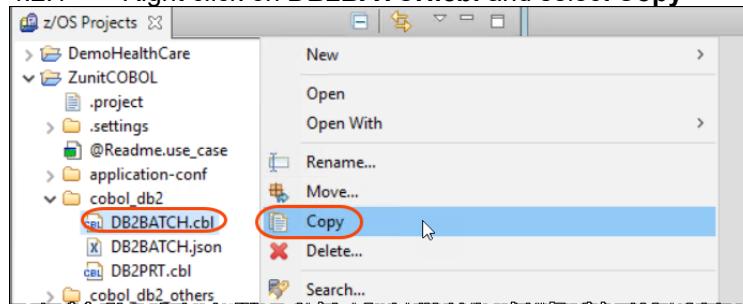
213      IF PERF-NUL < 0
214          THEN
215              MOVE 0 TO PERF-TBL-MIN
216              MOVE 0 TO PERF-TBL-MAX
217              MOVE 1 TO PERF-TBL-AVG.
218              MOVE 0 TO PERF-TBL-AVG.
219
220      * %bug --- add a bug un-commenting below
221      IF DEPT-TBL = 'ACC'
222          MOVE 'BAD' to DEPT-TBL W-DEPT-TBL.
223      *
224
225      250-EXIT.
226      EXIT.

```

4.2 Re-compile and link the changed program

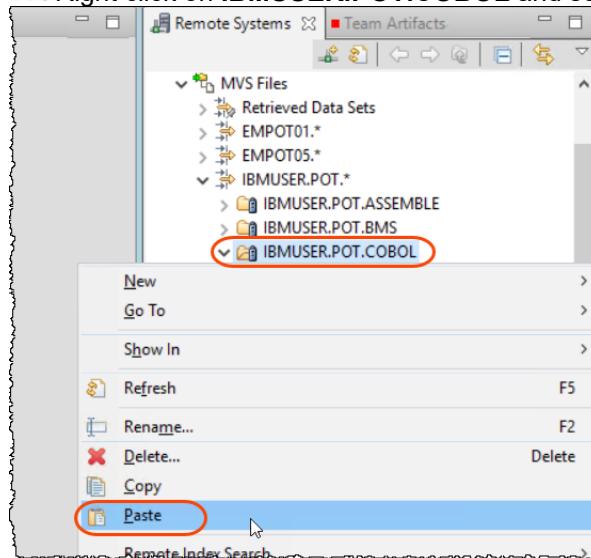
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar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

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

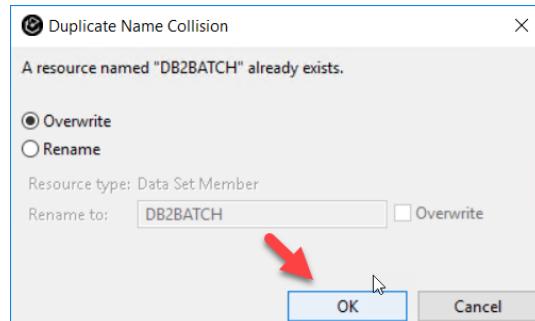


4.2.2 ► Using the Remote systems (on right),

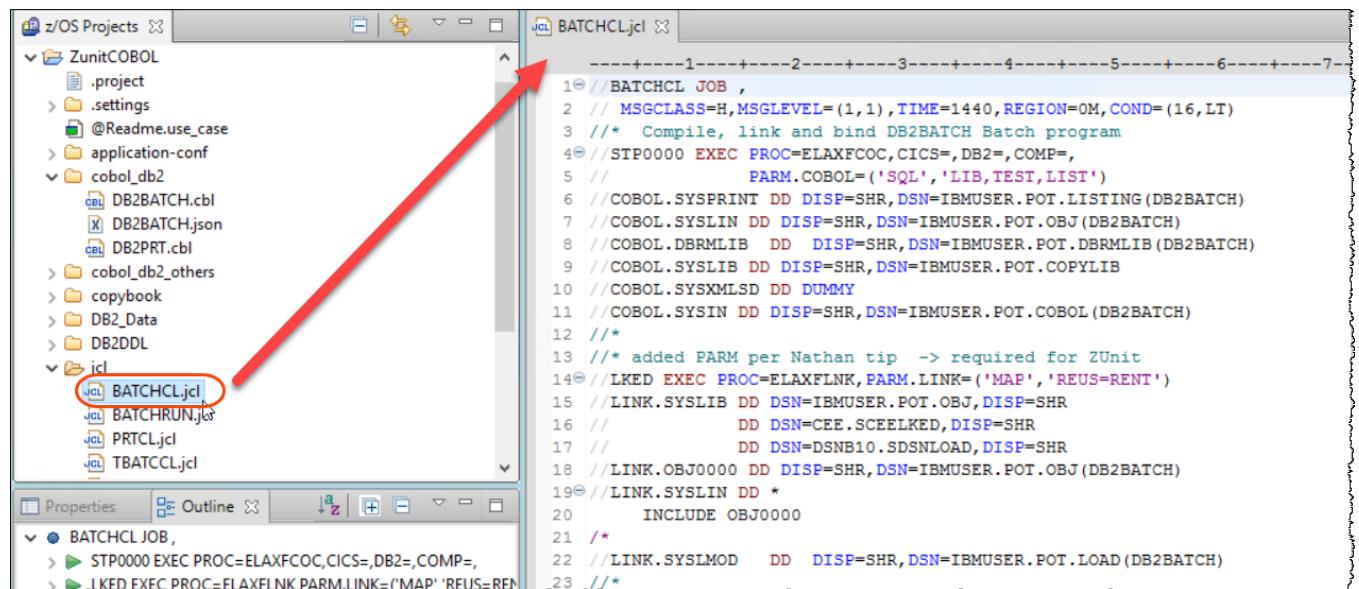
► Right click on **IBMUSER.POT.COBOL** and select **Paste**



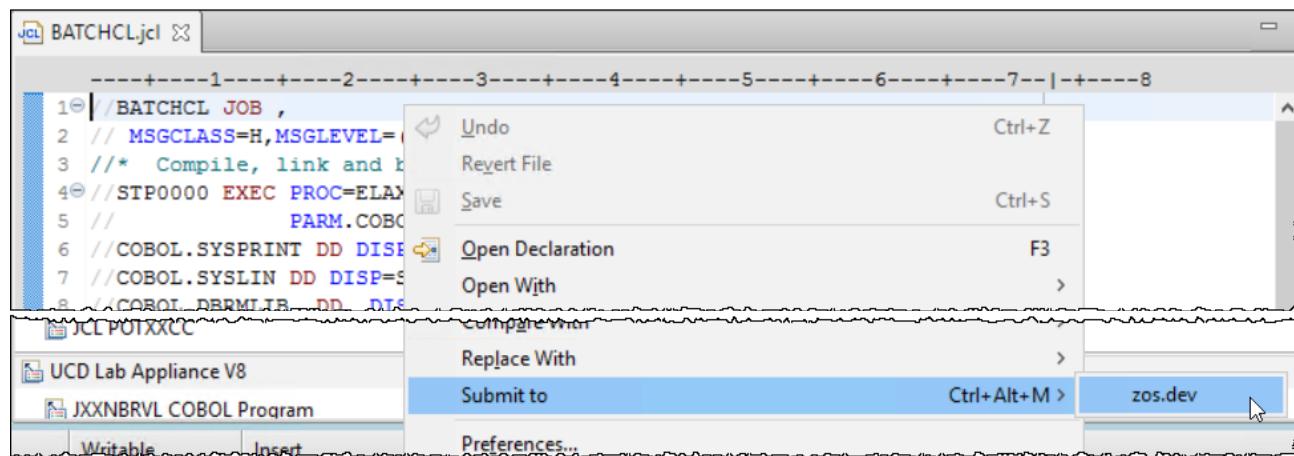
4.2.3 ►| Select **OK** to overwrite this member since it already existed on z/OS



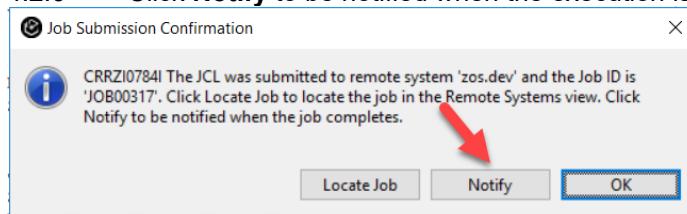
4.2.4 ►| Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl**. This JCL will compile, link and bind the program being tested.



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

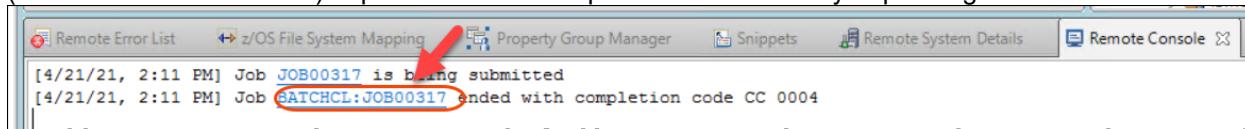


4.2.6 ► Click **Notify to be notified when the execution is complete.**



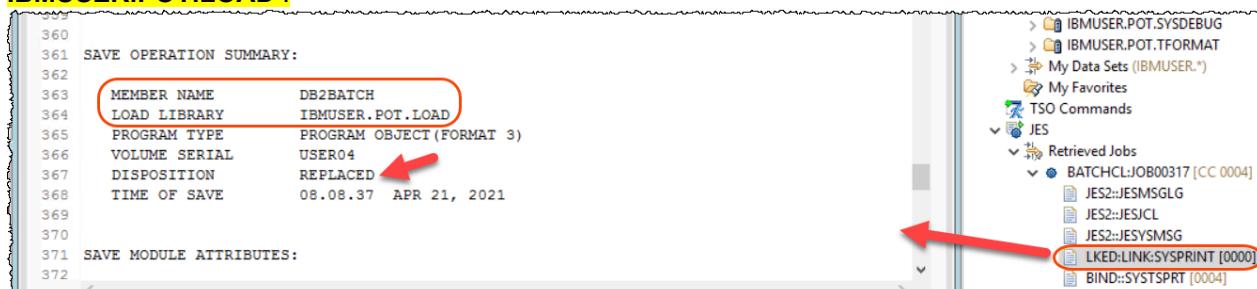
4.2.7 Under **Remote Console, you will be notified when execution is completed.
The completion code must be 4.**

► Once the execution ends, click on the link **BATCHJCL:JOB00xxx (where 00xxx 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 **LKD:LINK:SYSPRINT** step.**

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

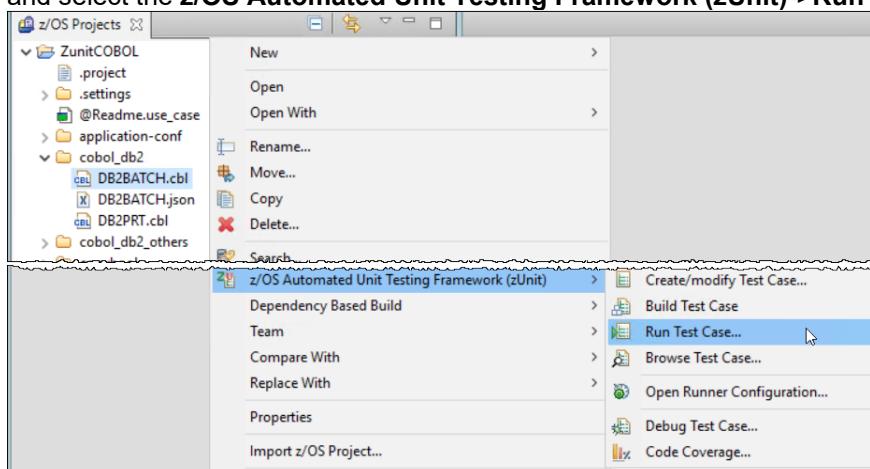


4.2.9 ► Use **Ctrl + Shift + F4 to close all opened editors.**

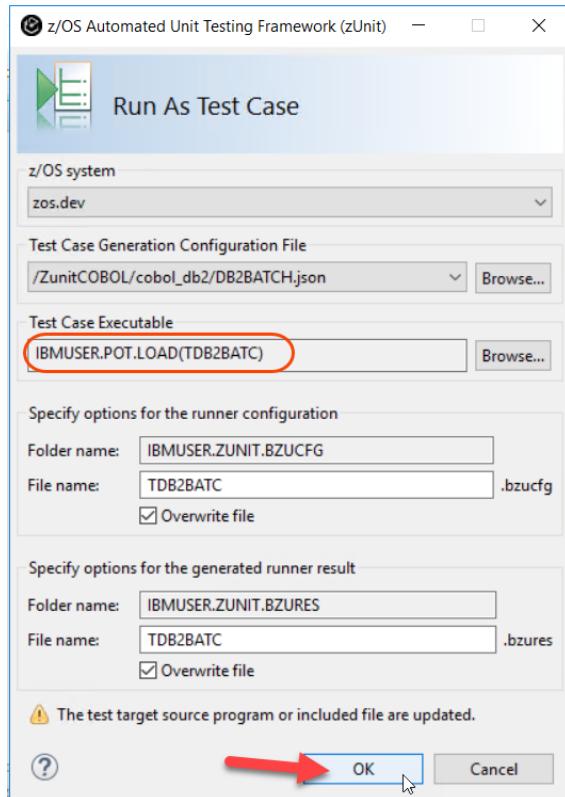
4.3 Running the test case again

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

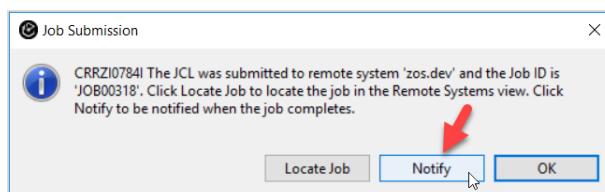
4.3.1 ► On the z/OS Projects view, select **DB2BATCH.cbl, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..****



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

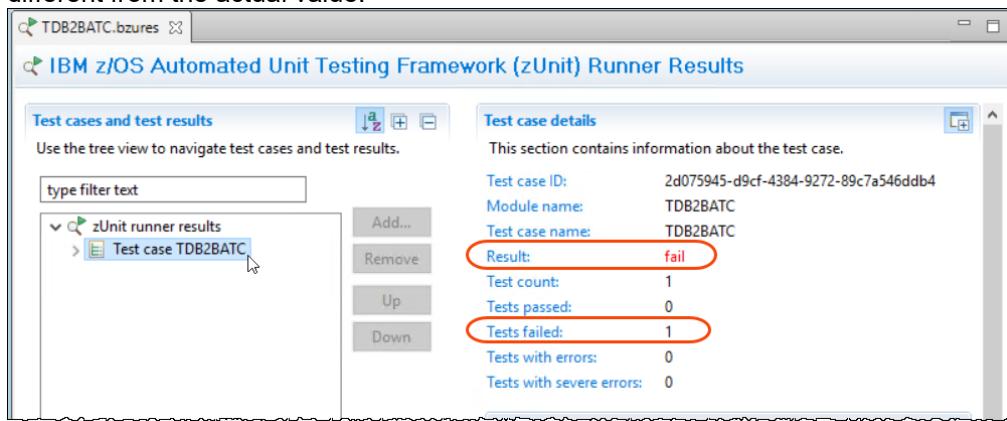


4.3.3 ► Click **Notify** on the Job Submission Confirmation dialog.



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

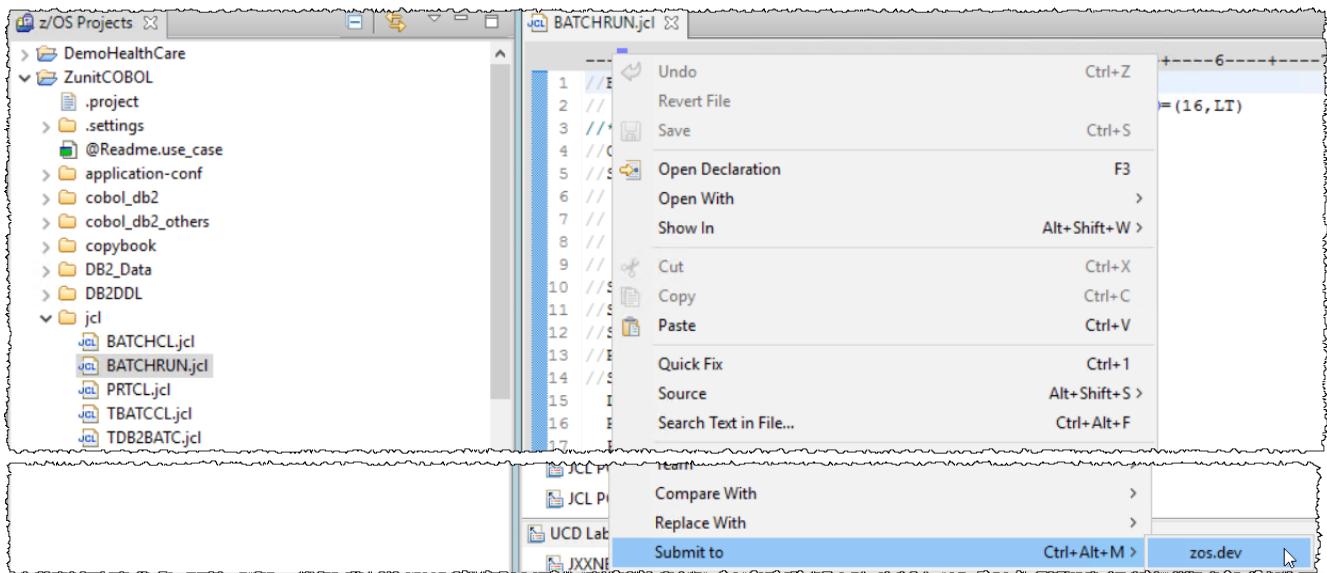
- 5.1.1 ► Under **ZunitCOBOL** expand **jcl** and double click on **BATCHRUN.jcl** to edit the JCL that will be submitted for execution.

```

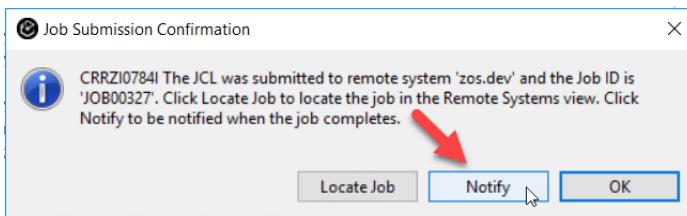
JCL BATCHRUN.jcl
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7---
1 //BATCHRUN JOB ,
2 //MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 //* Execute DB2BATCH Batch program
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=IBMUSER.POT.LOAD,DISP=SHR
6 //          DD DSN=DSN10.SDSNLOAD,DISP=SHR
7 //          DD DSN=DSN10.DBDBG.RUNLIB.LOAD,DISP=SHR
8 //          DD DISP=SHR,DSN=BZU100.SBZULOAD
9 //          DD DISP=SHR,DSN=EQAE10.SEQAMOD
10 //SYSPRINT DD SYSOUT=*
11 //SYSOUT DD SYSOUT=*
12 //SYSTSPRI DD SYSOUT=*
13 //RPTPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
14 //SYSTSIN DD *
15 DSN SYSTEM(DBDBG)
16 RUN PROGRAM(DB2BATCH) -
17 PLAN(DB2BATCH)
18 END
19 /*

```

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

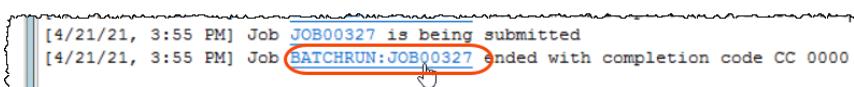


5.1.3 ► Click **Notify** to be notified when the execution is complete.

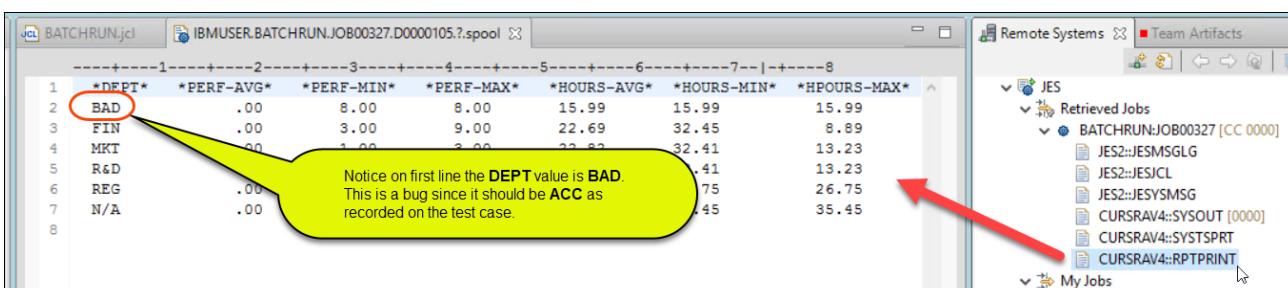


5.1.4 Under **Remote Console**, you will be notified when execution is completed.

► Once the execution ends, **click on the link** **BATCHRUN:JOB00xxx** (where 00xxx 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**,

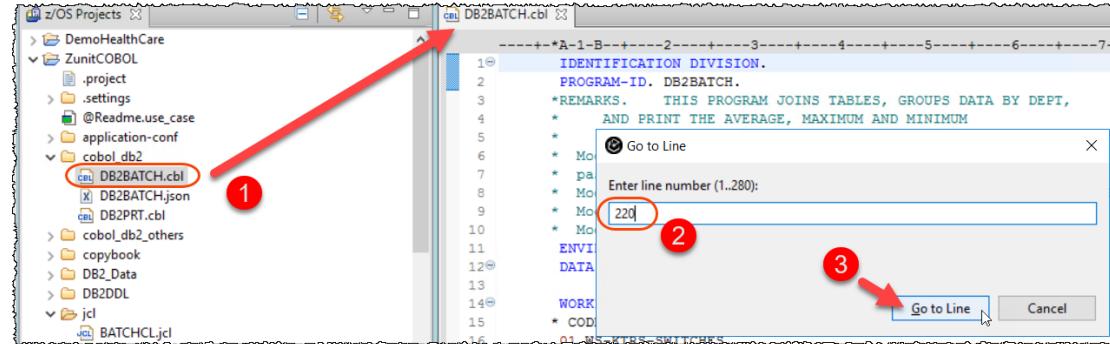
Section 6. Use IDz to fix the bug, recompile the COBOL/DB2 program.

You will fix the using IDz, and rebuilt the executable

6.1 Modifying the program to eliminate the bug

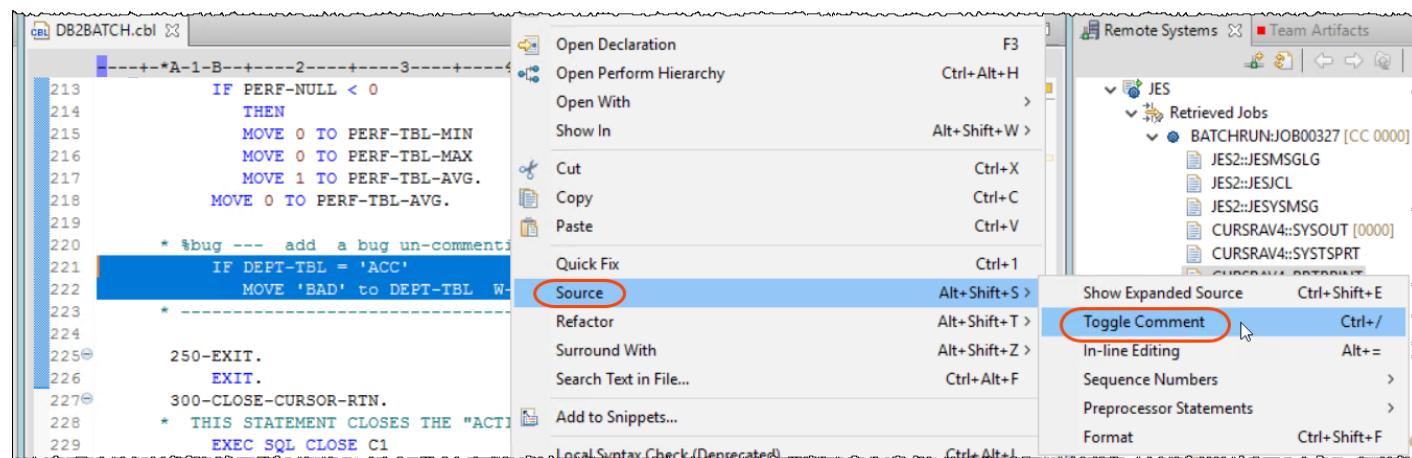
6.1.1 ► Using z/OS Projects view open DB2BATCH.cbl under cobol_db2 by double clicking on it

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

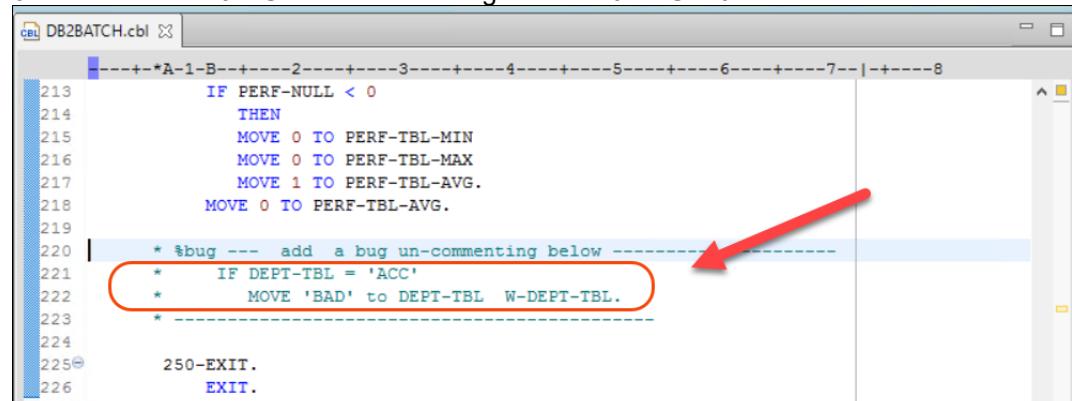


6.1.3 ► Change the lines 221 and 222 adding an * on the column 7.

The easiest way to do that is **selecting those 2 lines**, right click and select **Source > Toggle Comment**



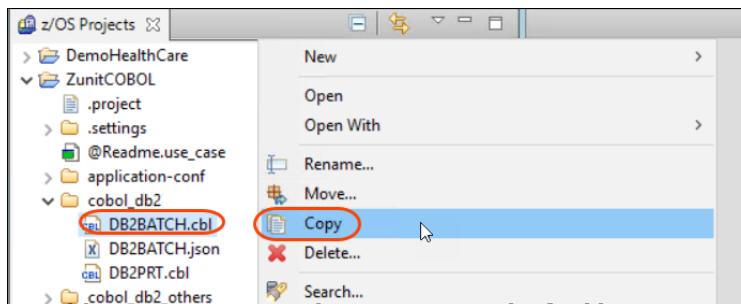
6.1.4 ► Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



6.2 Re-compile and link the changed program

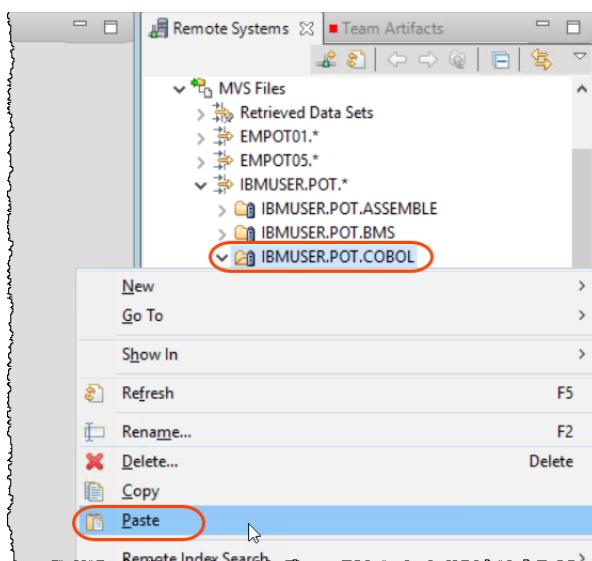
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the “old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar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

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

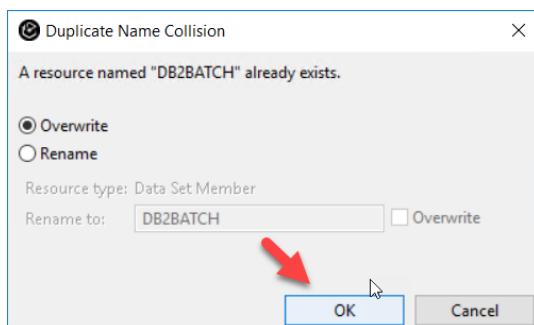


6.2.2 ► Using the Remote systems (on right),

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



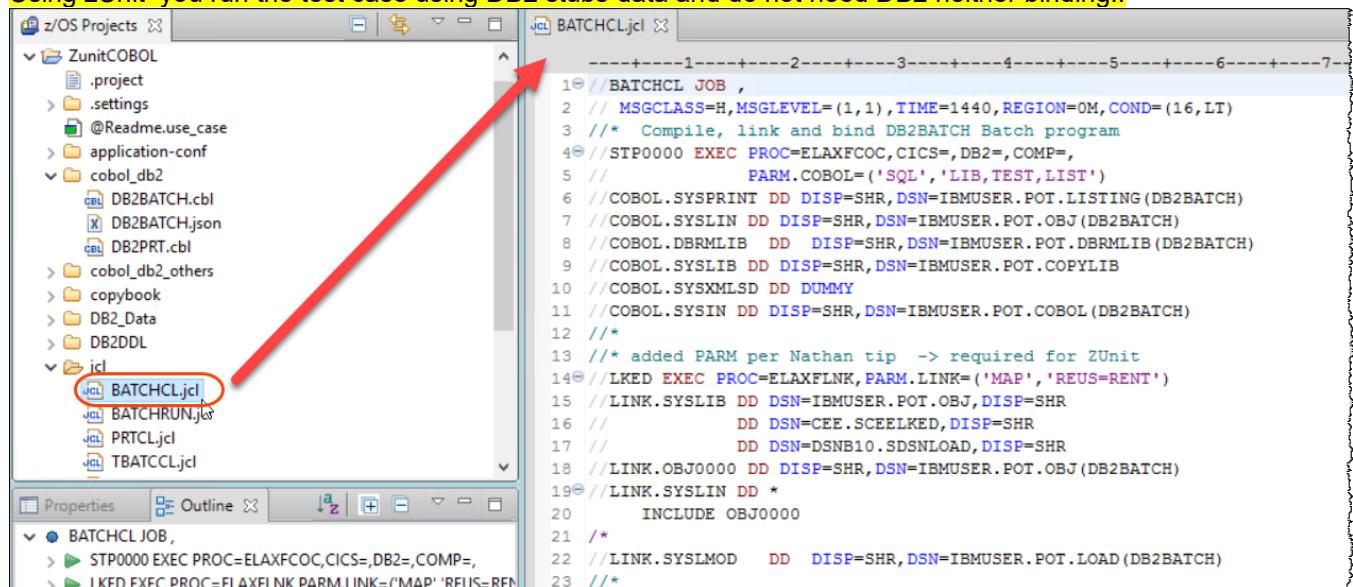
6.2.3 ► Select **OK** to overwrite this member since it already existed on z/OS



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..



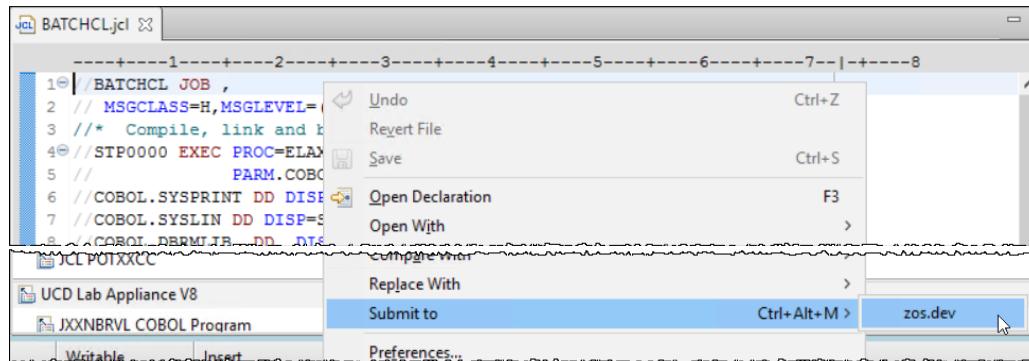
```

z/OS Projects ✎
  ZunitCOBOL
    .project
    .settings
    @Readme.use_case
    application-conf
    cobol_db2
      DB2BATCH.cbl
      DB2BATCH.json
      DB2PRT.cbl
      cobol_db2_others
      copybook
      DB2_Data
      DB2DDL
    jcl
      BATCHCL.jcl (highlighted)
      BATCHRUN.jcl
      PRTCL.jcl
      TBATCL.jcl

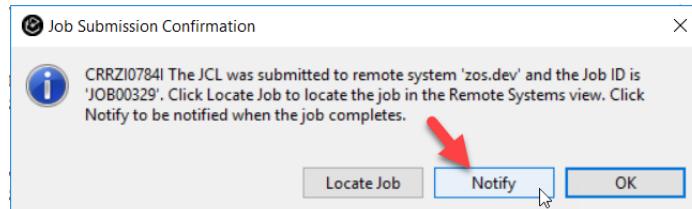
JCL BATCHCL.jcl ✎
-----+-----+-----+-----+-----+-----+-----+-----+
1 //BATCHCL JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=0M,COND=(16,LT)
3 /* Compile, link and bind DB2BATCH Batch program
4 //STP0000 EXEC PROC=ELAXFCOC,CICS=,DB2=,COMP=
5 // PARM.COBOL=('SQL','LIB,TEST,LIST')
6 //COBOL.SYSPRINT DD DISP=SHR,DSN=IBMUSER.POT.LISTING(DB2BATCH)
7 //COBOL.SYSLIN DD DISP=SHR,DSN=IBMUSER.POT.OBJ(DB2BATCH)
8 //COBOL.DBRMLIB DD DISP=SHR,DSN=IBMUSER.POT.DBRMLIB(DB2BATCH)
9 //COBOL.SYSLIB DD DISP=SHR,DSN=IBMUSER.POT.COPYLIB
10 //COBOL.SYSMLSD DD DUMMY
11 //COBOL.SYSIN DD DISP=SHR,DSN=IBMUSER.POT.COBOL(DB2BATCH)
12 /*
13 /* added PARM per Nathan tip -> required for ZUnit
14 //LKED EXEC PROC=ELAXFLINK,PARMLINK='MAP','REUS=RENT'
15 //LINK.SYSLIB DD DSN=IBMUSER.POT.OBJ,DISP=SHR
16 // DD DSN=CEE.SCEELKED,DISP=SHR
17 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR
18 //LINK.OBJ0000 DD DISP=SHR,DSN=IBMUSER.POT.OBJ(DB2BATCH)
19 //LINK.SYSLIN DD *
20 INCLUDE OBJ0000
21 /*
22 //LINK.SYSLMOD DD DISP=SHR,DSN=IBMUSER.POT.LOAD(DB2BATCH)
23 /*

```

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



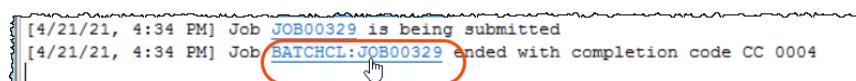
6.2.6 ► Click **Notify** to be notified when the execution is complete.



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

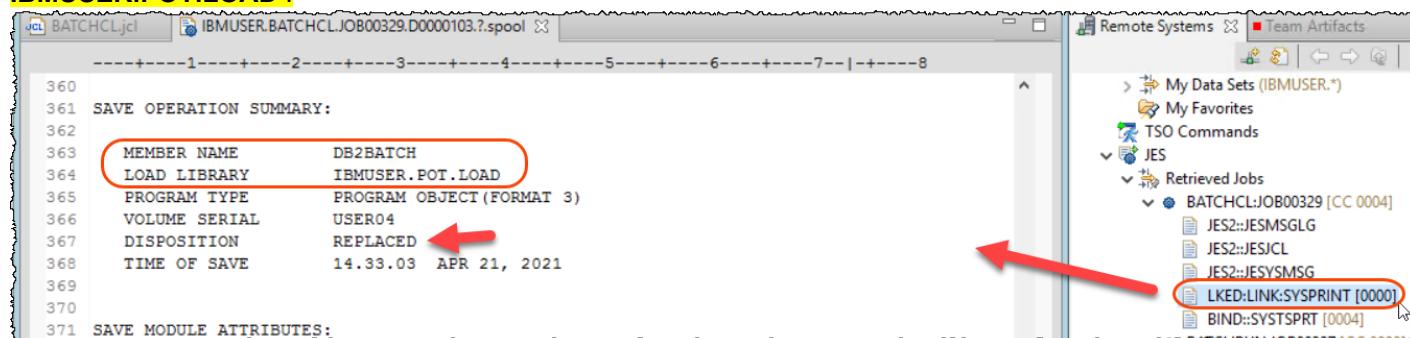
The completion code must be 4.

► Once the execution ends, **click on the link **BATCHJCL:JOB00xxx**** (where 00xxx 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.

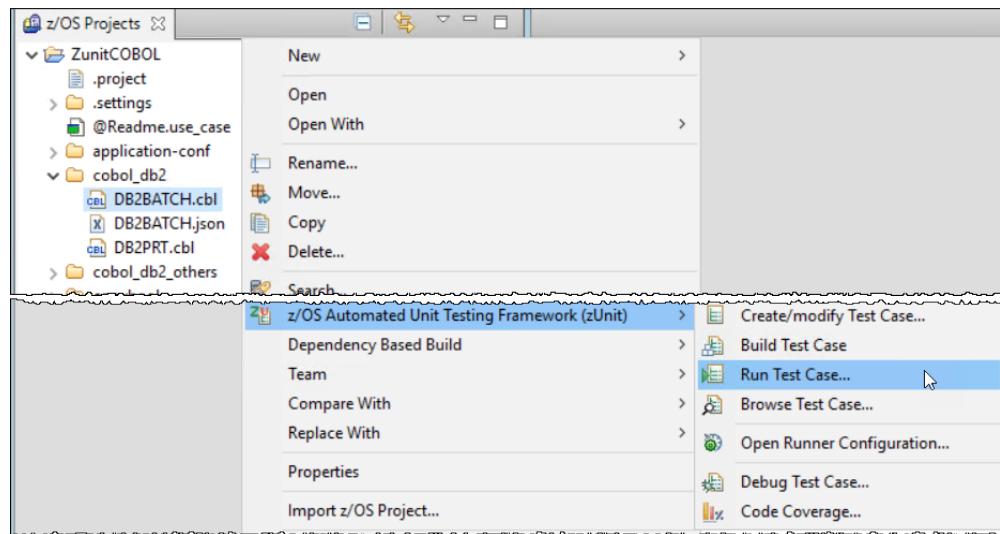
Section 7. Rerun the zUnit and verify that the bug is eliminated.

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

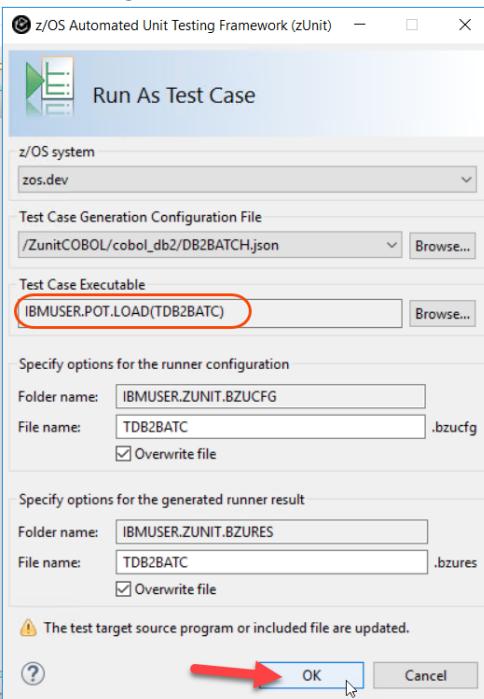
7.1 Running the test case again

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

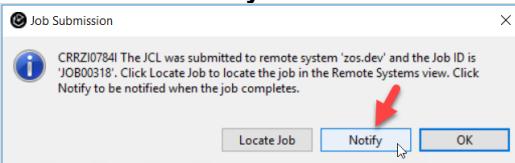
7.1.1 ► On the z/OS Projects view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..**



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

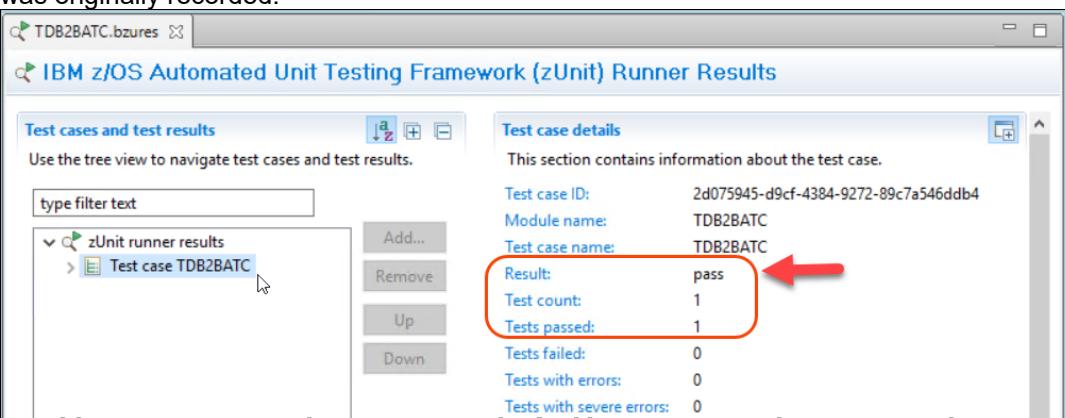


- 7.1.3 ► Click **Notify** on the Job Submission Confirmation dialog.



- 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 6D.

LAB 7 – Using IBM Dependency Based Build with Git, Jenkins and UCD on z/OS (60 minutes)

Updated June-25 2021 by Regi, (Reviewed by Wilbert)

This lab will take you through the steps of using [IBM Dependency Based Build](#) (DBB) along with [Git](#), [Jenkins](#) and [UrbanCode Deploy](#) (UCD) on z/OS.

On this lab you will modify an existing COBOL/CICS application stored in Git.

You will use **IDz** (which is part of the **ADFz** "package") to change the code and perform a personal test for later delivery and commit to *Git* and then use *Jenkins* for the final build and continuous delivery.

The updated code will be deployed to CICS using *UrbanCode Deploy* (UCD)

The process would be similar for a PL/I program or IMS instead of CICS.

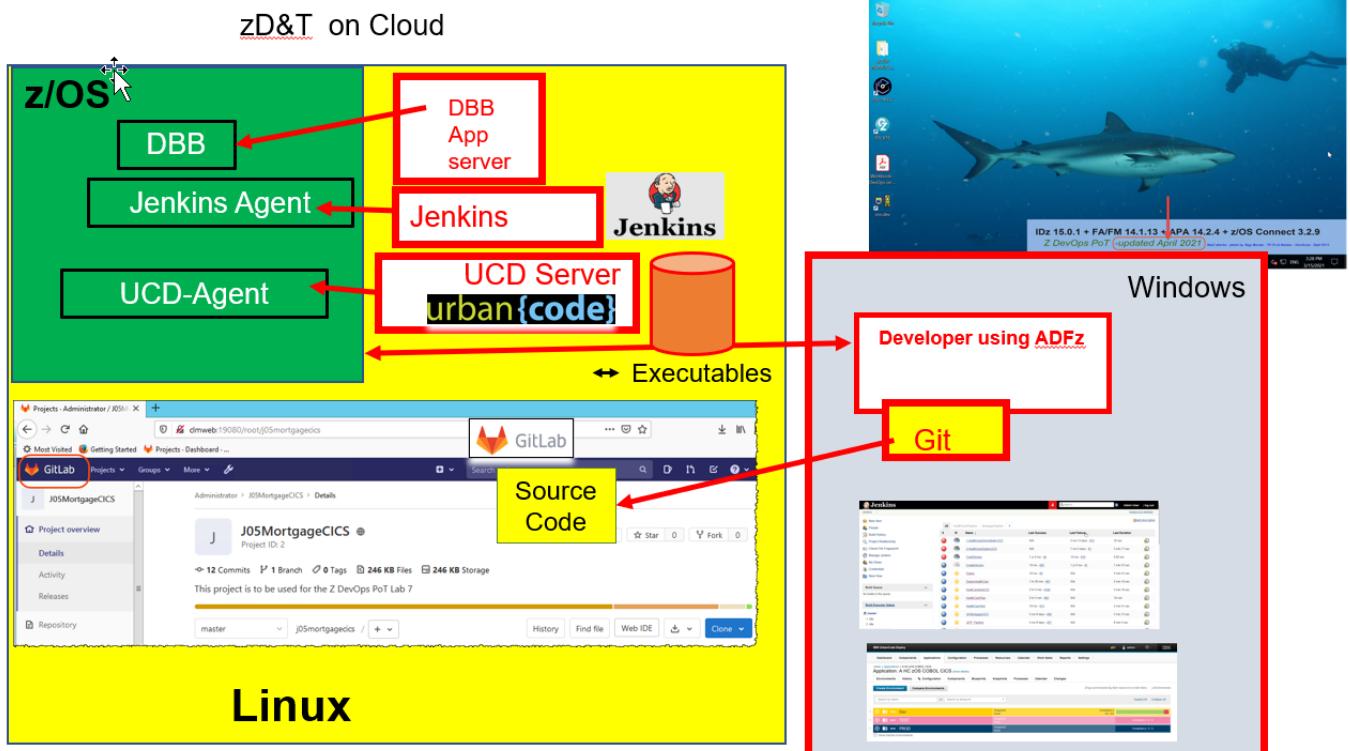
More about DBB https://www.ibm.com/support/knowledgecenter/SS6T76_1.1.0/welcome.html

The environment used on this Lab is pictured below.

Note that we have few “servers” running on Linux like **UCD server**, **Jenkins**, **DBB Application Server**. Also the **Git Repository** is on Linux.

The z/OS has many other running tasks including **CICS**, **DB2**, **DBB code** and agents that interact with the Linux servers.

Topology used on the labs



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 **J05P** to become familiar with the Application that you intend to modify.
2. **Load the source code from Git to the local IDz workspace**
→ You will load the COBOL code that is stored on Linux to your windows client to be modified.
3. **Modify the COBOL code using IDz.**
→ Using *IDz* you will modify the COBOL code to have a different message in a *C/ICS* dialog.
4. **Use IDz DBB User Build to compile/bind and perform personal tests.**
→ You will compile and link the modified code using the *DBB User Build* Function. When complete you will run the code using *C/ICS* for a personal test and verify that the change is correctly implemented.
5. **Push and Commit the changed code to Git .**
→ You will commit the changes to *Git*.
6. **Use Jenkins with Git plugin to build all the modified code committed to Git.**
→ You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using *UCD*.
Note that this step makes a build of all code committed to Git, while on step 4 only the selected COBOL program was built.
7. **Use Jenkins and UCD plugin to deploy results and test the CICS transaction again**
→ You will verify the results after the final deploy to *C/ICS* using *UCD*
8. **(Optional) Understanding DBB Build Reports**
→ You will understand the reports generated by *DBB* during the build.

What is Git and DBB ?

Git is an open Source Code Management tool that is very popular in the distributed world.

In early 2017, Rocket Software ported Git into the mainframe – with the necessary checks to handle EBCDIC to UTF-8 conversions and vice-versa.

In Q3 2017, IBM released an Open Beta of **Dependency Based Build (DBB)**. DBB provides a build tool that provides the build framework, dependency understanding, and tracking for builds run on z/OS. This build system is not dependent on any SCM or Continuous integration automation tool. In this lab, we use DBB to build our z/OS COBOL source code which resides in the distributed Git Repositories.

DBB is part of IBM Developer for Z Systems EE (Enterprise Edition) or ADFz and was announced on March 13, 2018.



GitHub vs. Bitbucket vs. GitLab ?

More at: <https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab>

GitHub, **Bitbucket**, and **GitLab** are code collaboration and version control tools offering repository management. They each have their share of fans, though **GitHub** is by far the most used of the three.

Of the three, only **GitLab** is open source, though all three support open source projects. For that reason we are using **GitLab** in this lab, but the steps would be exactly the same if using other tools.

GitHub offers free public repositories; **Bitbucket** also offers free private repositories; **GitLab** offers a Community Edition which is entirely free

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

You will start a 3270 emulation and execute a transaction named J05P to become familiar with the Application that you intend to update.

Tip → Use the notepad and edit **C:\ADF_POTILAB7** to copy/paste values if you don't want to type).

1.1 Connect to z/OS and emulating a CICS 3270 terminal

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

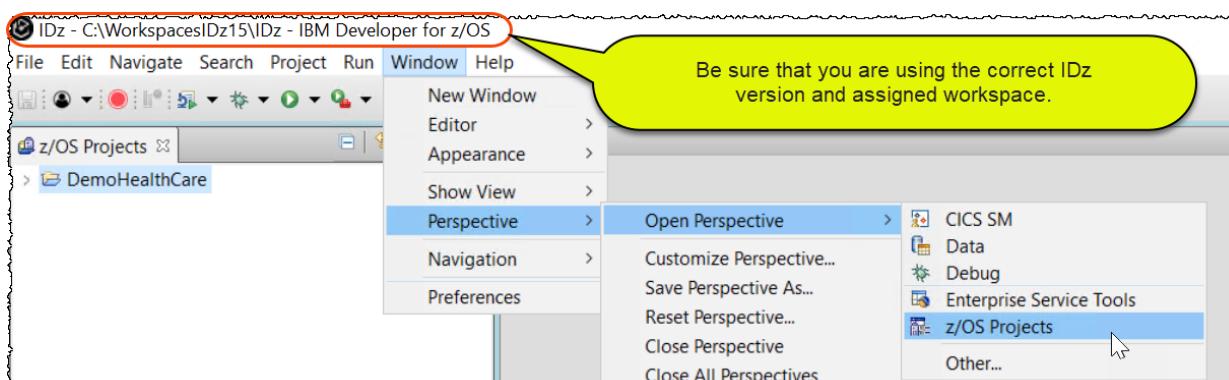
► Using the desktop double click on **IDz V15** icon.

► 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**



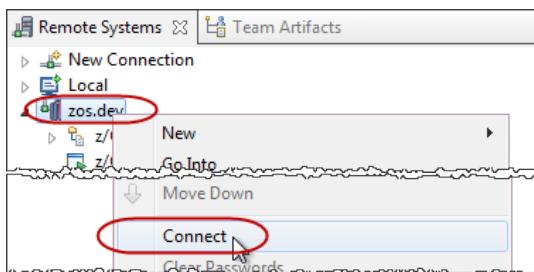
1.1.3 On other labs you used userid **empot01** or **ibmuser**.

Your new userid now will be **empot05**.

You need to disconnect and reconnect to the other userid.

► If you are already connected, right click on **zos.dev** and select **Disconnect**.

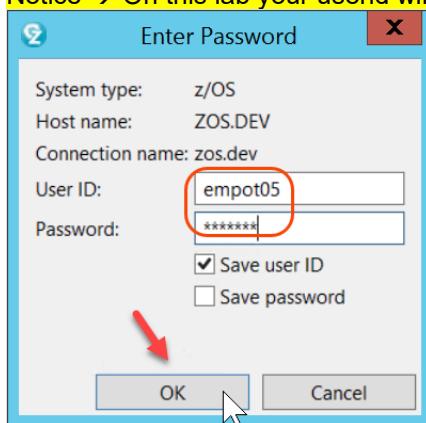
1.1.4 ► Right click on **zos.dev** and select **Connect**.



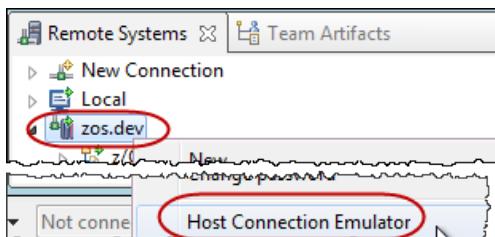
1.1.5 ► Type **empot05** as userid and **empot05** as password.

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

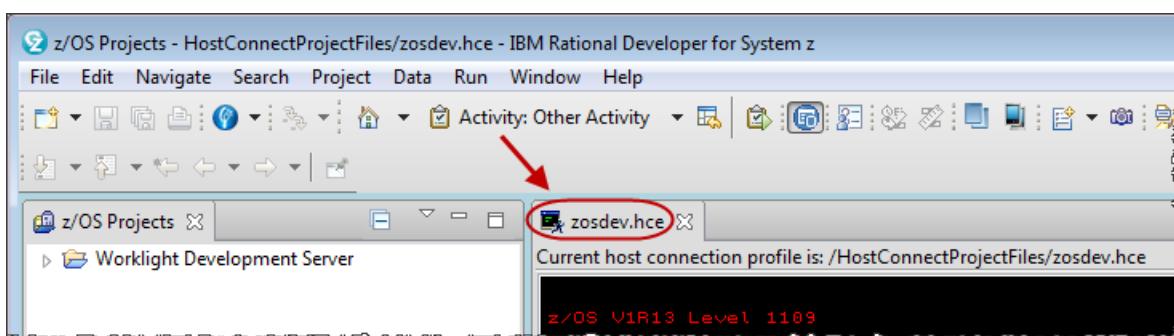
Notice → On this lab your userid will be empot05 (not empot01 used in previous labs)



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



1.1.7 ► Since you will need more space, **double-click** on the **zosdev.hce** title



1.1.8 ► Type **I cicsts53**. (where "I" is the lower case of letter "L") and press **Enter key**.

```
==> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"
l cicsts53_
24/011
```

1.1.9 ► Logon using **empot05** and password **empot05** and press **Enter**.

```
Signon to CICS
WELCOME TO CICS TS 5.3

Type your userid and password, then press ENTER:
--> Userid . . . empot05      Groupid . . .
--> Password . . . -          Language . . .
--> New Password . . .

MA a
```

1.1.10 The sign-on message is displayed

```
DFHCE3549 Sign-on is complete (Language ENU).
MA a
```

1.2 Run CICS transaction J05P

You should now be in the z/OS CICS region named *C/CSTS53*. This is the CICS instance where you will make the program changes.

1.2.1 ► Type the CICS transaction **j05p** and press the **Enter key**.

```
zosdev.hce x
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
j05p
```

1.2.2 ► Press **enter** to see the monthly payment for the default data .

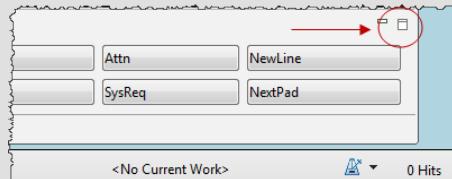
```
JKE MORTGAGE CALCULATOR

Amount of Loan: 1000
Length of Loan in Years: 10
Interest Rate: 5

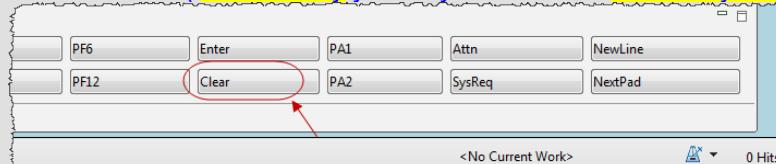
Press F3 to quit or Enter to calculate loan
Press PF9 to see companies that can match or beat this rate

Monthly Payment: 10.61
```

You may need to use the **clear** key. If the clear button is not displayed, look in the right lower corner, select this icon  . This will display possible keys, including the clear button.



Click on Clear (If necessary you may also use the Reset key after clicking NextPad)



1.2.3 ► Press **F1** key and verify the message displayed “**INVALID KEY PRESSED**” .

Your mission will be modifying the COBOL program that send this message.

Your new message must be “**YYYYMMDD - INVALID KEY PRESSED**” .

Where **YYYYMMDD** will be the today's date.

```
JKE MORTGAGE CALCULATOR

Amount of Loan:      1000
Length of Loan in Years: 10
Interest Rate:        5

Press F3 to quit or Enter to calculate loan
Press PF9 to see companies that can match or beat this rate

Monthly Payment:      10.61

INVALID KEY PRESSED.
```

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

```
zosdev.hce ✘
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
END OF TRANSACTION - THANK YOU
```

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

What have you done so far?

You emulated a 3270 terminal using IBM Developer for System z.

You also executed the CICS transaction **J05P** and verified a simple interaction with the Mortgage application. The objective here was to show the code that you will update.

Section 2 – Load the source code from Git to the local IDz workspace

You will load the COBOL code that is stored on Linux to your Windows client to be modified.

2.1 Cloning the Git Repository

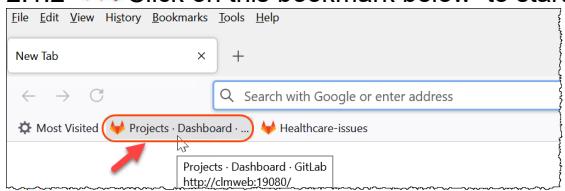
The Mortgage Application used in this lab has its source code in a *Git Repository* that is on Linux system. You must connect to the Git Repository and clone the Git project into the IDz. This brings the source code into your IDz workspace for edits and build. *GitLab* is open source and for that reason we are using *GitLab* in this lab. This lab would work also if using GitHub or Bitbucket.

2.1.1 Before cloning the Git repository, you should visualize the code stored at GitLab.

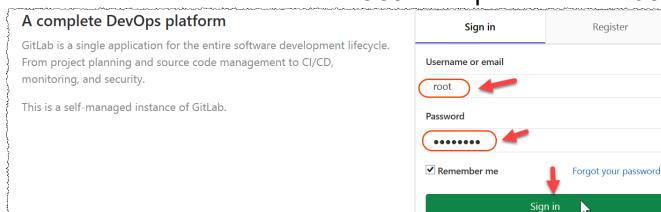
▶ Start a browser clicking in the icon in the bottom of your screen



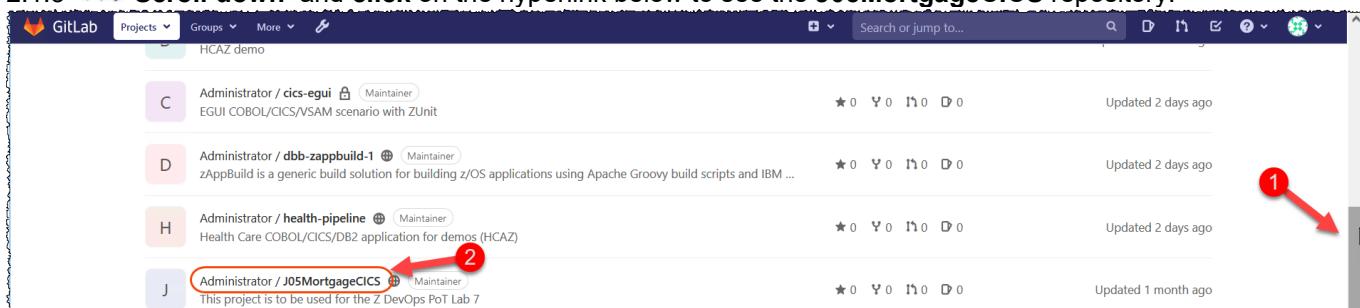
2.1.2 Click on this bookmark below to start *GitLab*. The URL used is <http://clmweb:19080/>



▶ If asked for credentials use **root** and password: **zdtlinux**

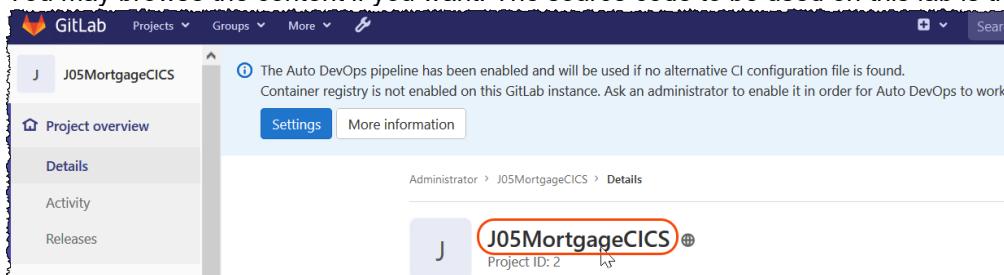


2.1.3 Scroll down and click on the hyperlink below to see the **J05MortgageCICS** repository.



2.1.4 You can see the **J05MortgageCICS** repository.

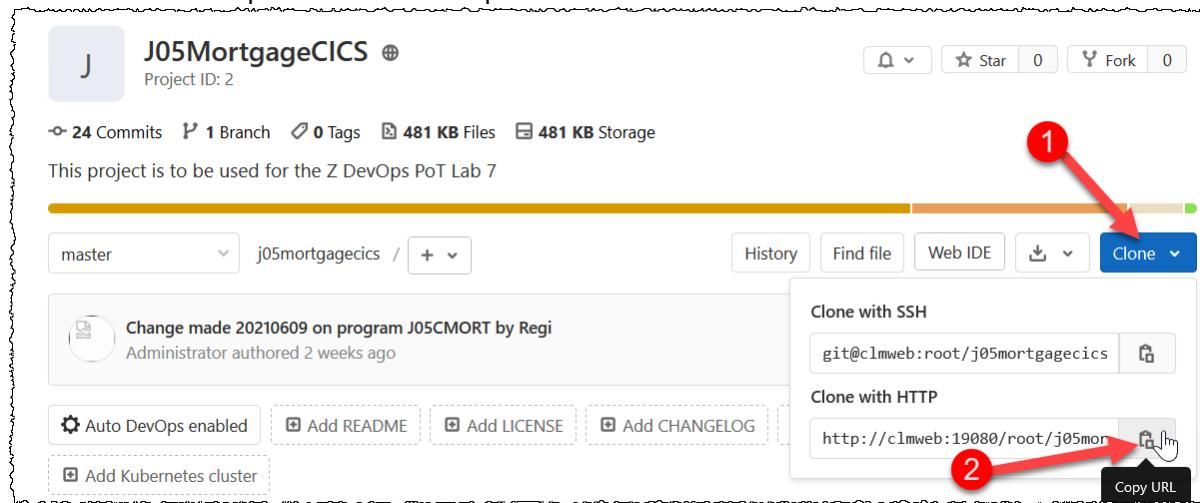
You may browse the content if you want. The source code to be used on this lab is there



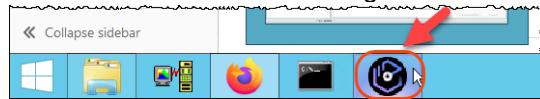
2.1.5 In order to clone it at your window desktop you could use **HTTP** or **SSH**.

Since **SSH** is blocked in our environment we need to use **HTTP**.

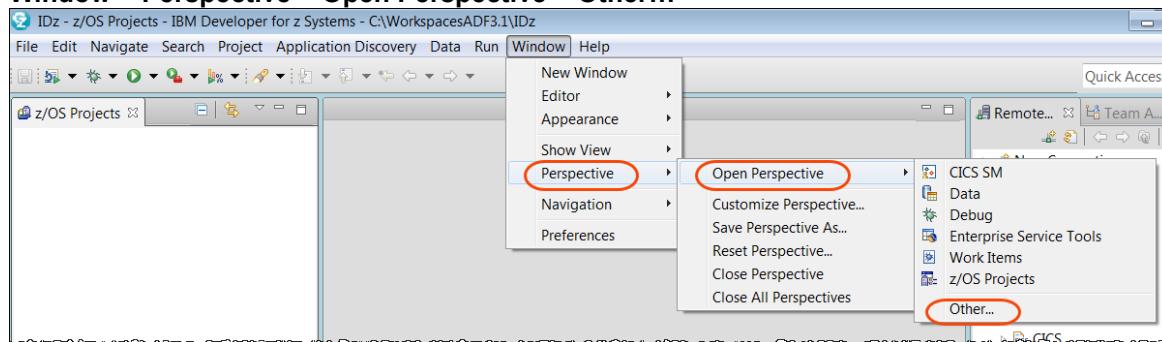
- ▶ 1 Click on **Clone** (the blue button) and 2 in the icon to **copy the URL**.
This value will be kept in the windows clipboard.



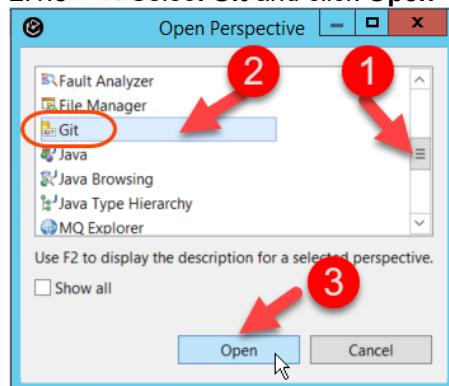
2.1.6 ▶ Go back to IDz using the icon that is on the base of your screen:



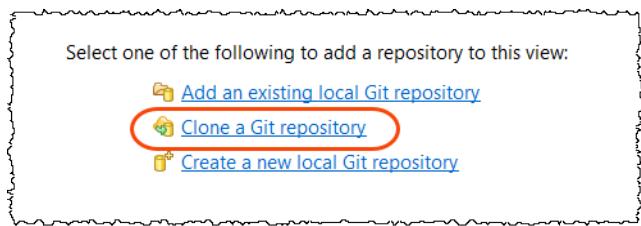
2.1.7 ▶ Open the **Git** perspective by selecting
Window > Perspective > Open Perspective > Other...



2.1.8 ▶ Select **Git** and click **Open**

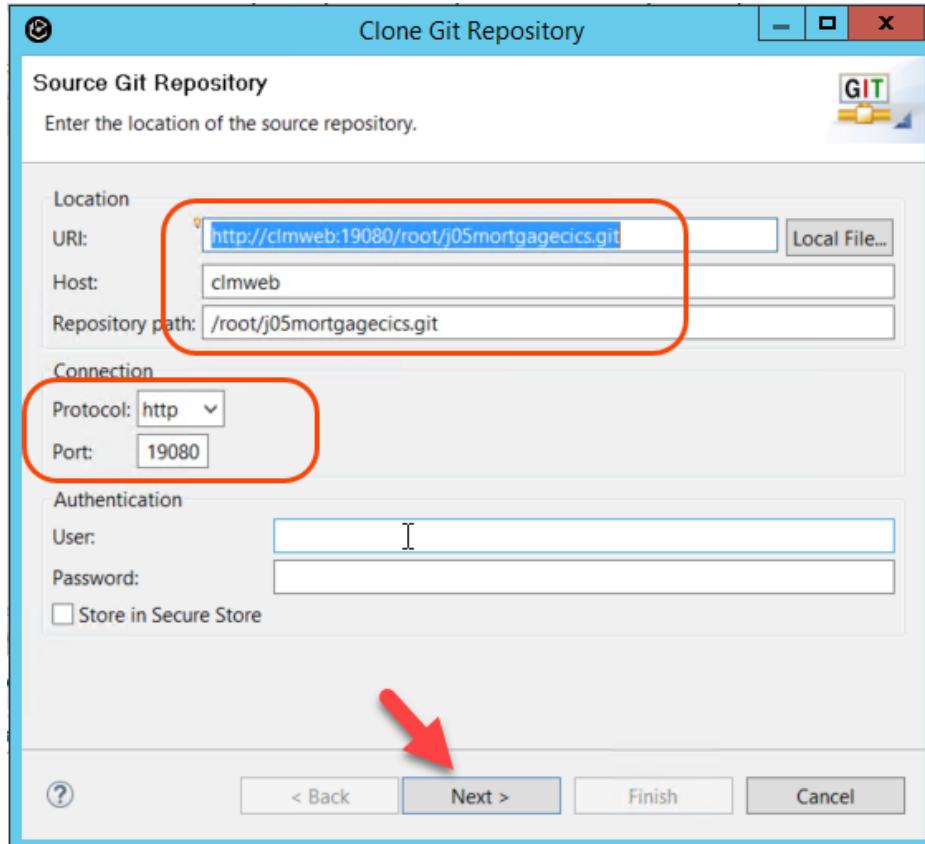


- 2.1.9 ► In the *Git Repositories* tab, click on the hyperlink to ‘Clone a Git repository’.



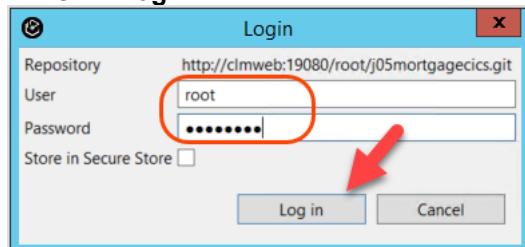
- 2.1.9 ► The values copied from the web page (copy URL) will be shown in the Clone Git Repository.
Tip: In case you don't see it, got back to the page and copy it again (steps 2.1.1-2.1.5 above).

► Click **Next**



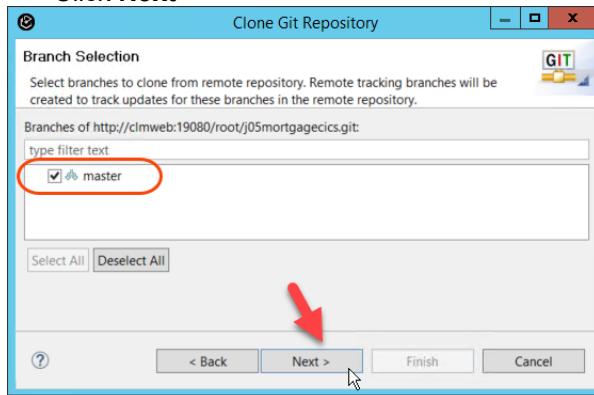
- 2.1.10 If a dialog asks for login the credentials are a user user: **root** and password **zdtlinux**

► Click **Log in**



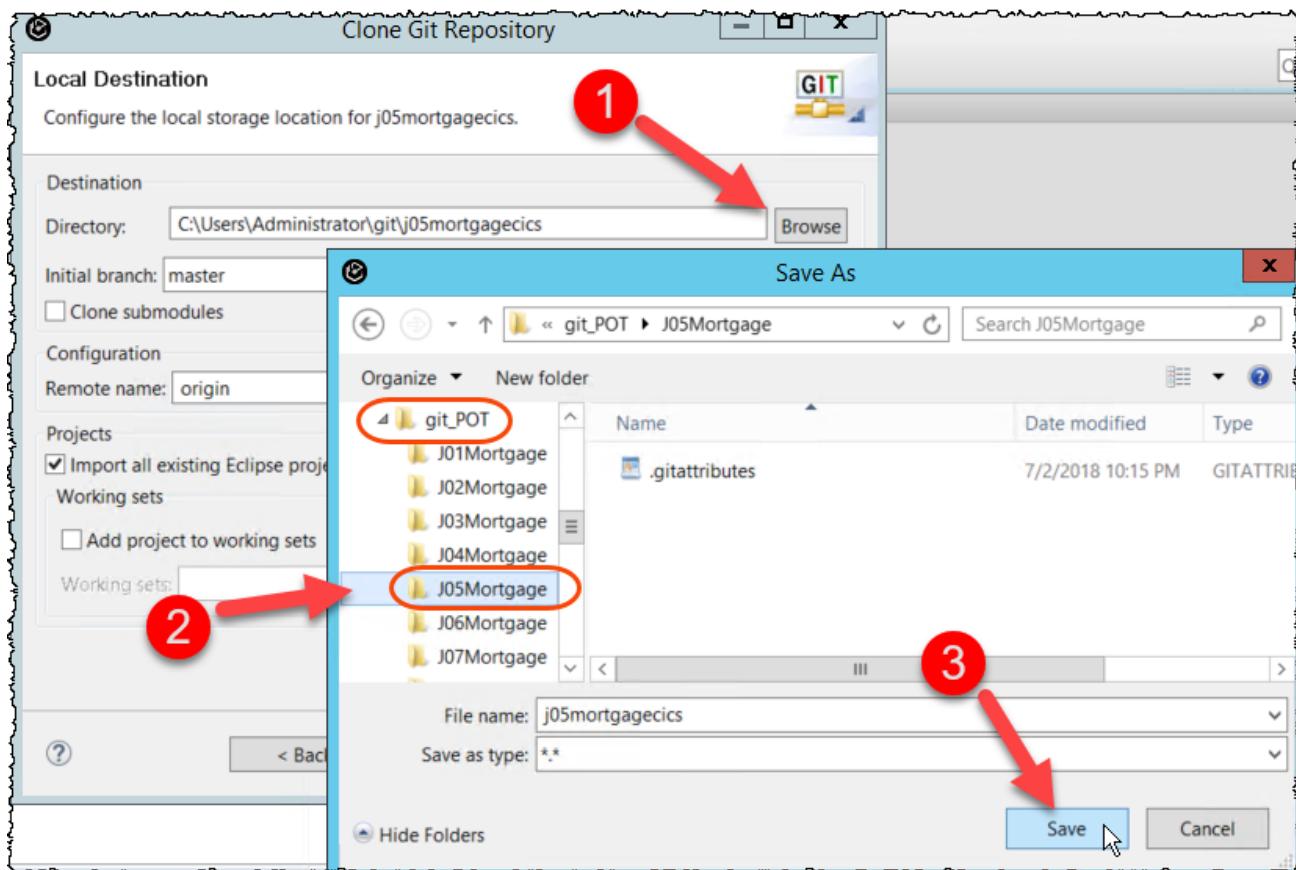
2.1.11 We have only one branch named master.

▶▶ Click Next



2.1.12 We will clone the repository on your local windows and import to be shown on IDz

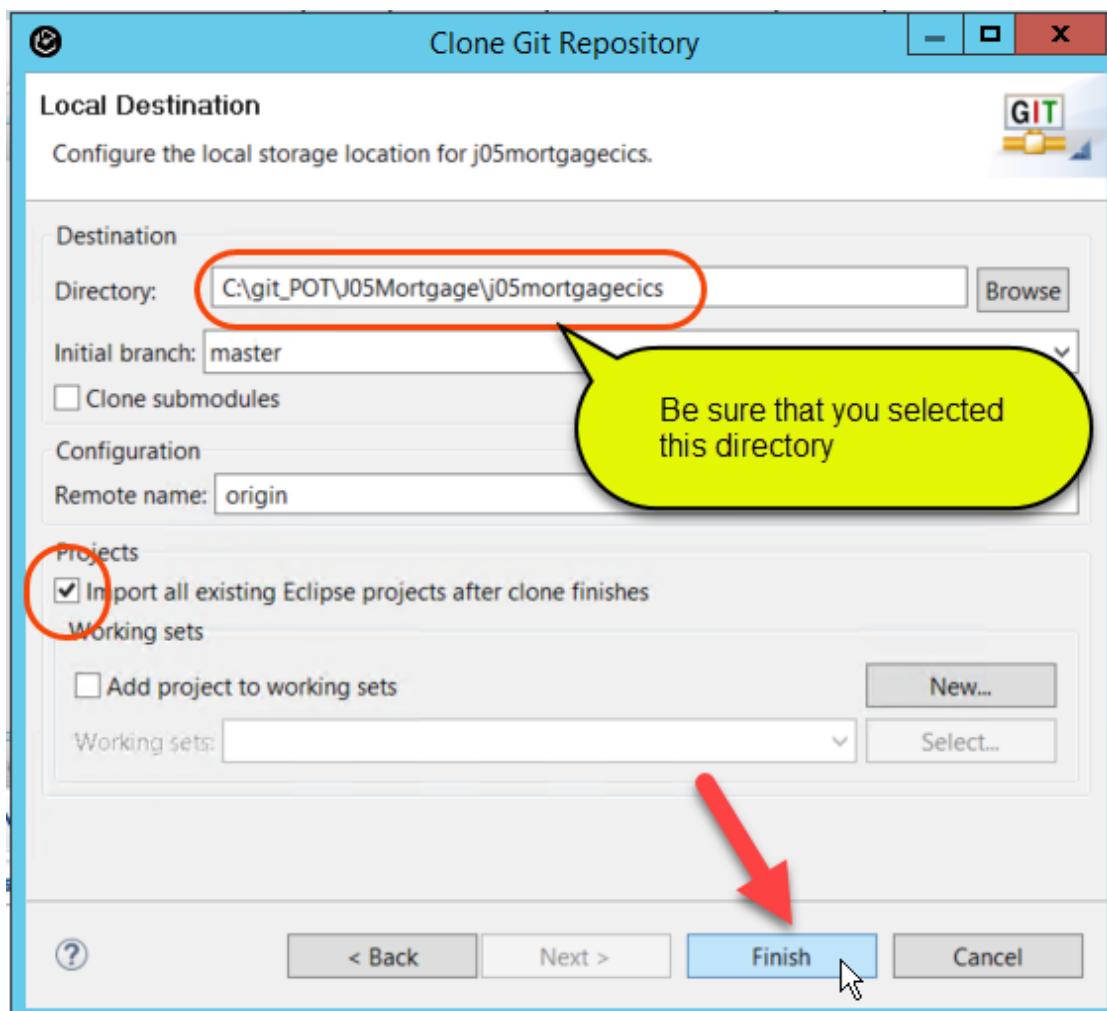
- 1 ▶▶ Use **Browse** button to select the directory **C:\git_POT\J05Mortgage** (on left).
- 2 ▶▶ Click folder **J05Mortgage** on left
- 3 ▶▶ Click **Save**



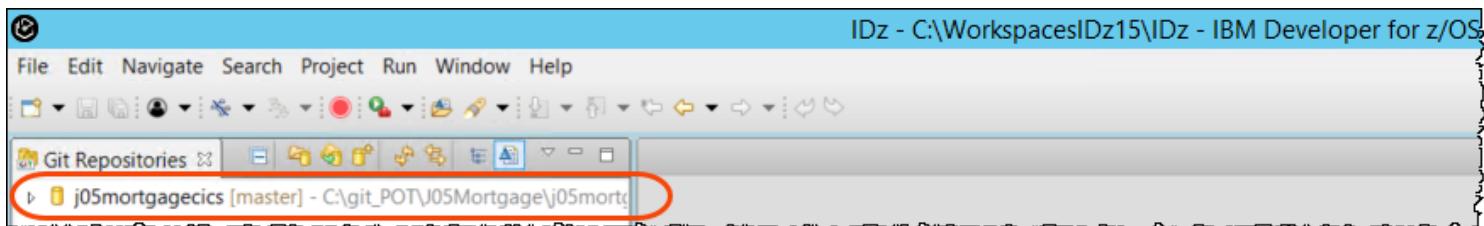
2.1.13 The *Directory* now should point to C:\git_POT\J05Mortgage\j05mortgagetcics

► Be sure that you selected **Import all existing Eclipse projects after clone finishes**

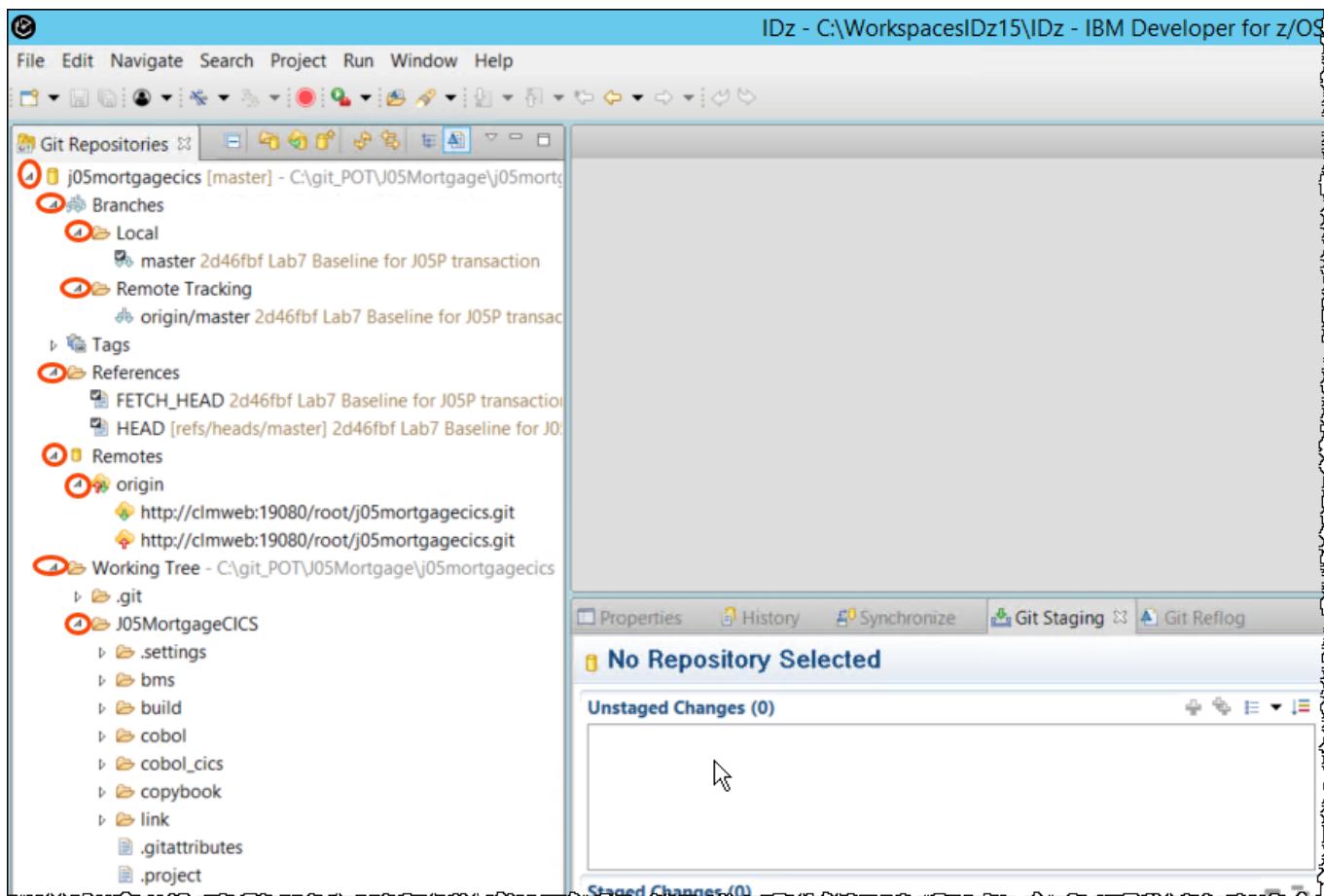
► Click **Finish**



2.1.14 The Mortgage Application will be cloned from the Remote master repository and will appear in the *Git Repositories* view.



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

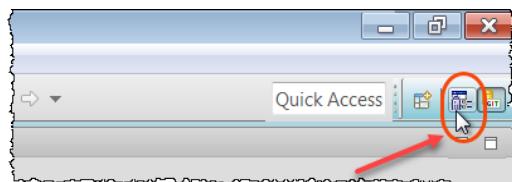


2.2 Verify the code cloned using z/OS projects perspective

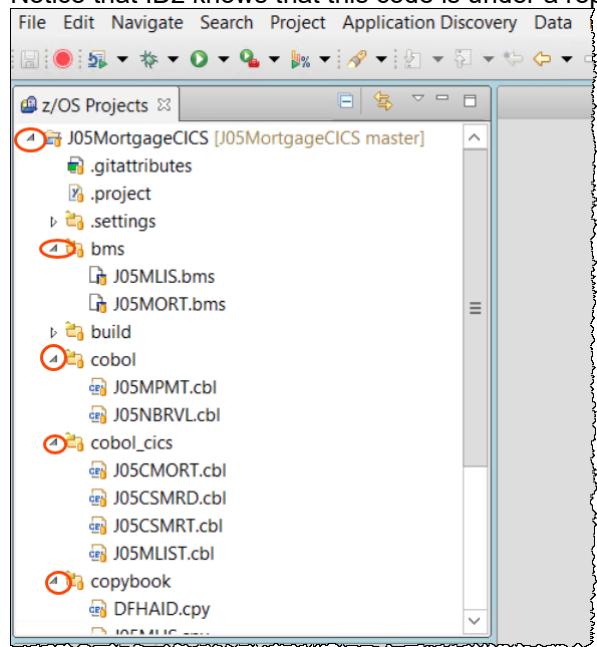
The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

Notice that you have cloned the project at your local workstation but later you will need to connect to z/OS to be able to compile and build your programs.

2.2.1 **Switch to the z/OS Projects perspective** clicking on icon on the top right corner



2.2.2 ► Expand the project **J05MortgageCICS** clicking on icon ▶ and see the source code loaded
 Notice that IDz knows that this code is under a repository and the yellow decorator on the icon 📁 indicates that.



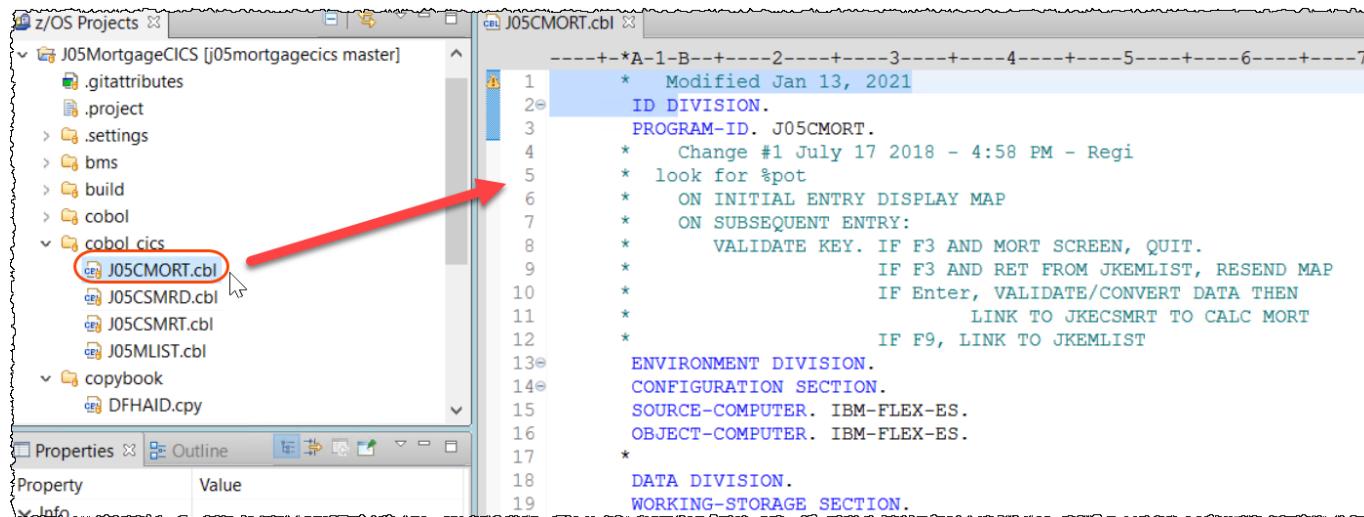
Section 3 –Modify the COBOL code using IDz.

Using IDz you will modify the COBOL to have a different message in a CICS dialog.
 You will replace the message “**INVALID KEY PRESSED**” when pressing F1 by another message:
“YYYYMMDD - INVALID KEY PRESSED”.

3.1 Edit and modify the code that send the message

The COBOL code to be modified is the program **J05CMORT** that is under the folder **cobol_cics**.

3.1.1 ► Using z/OS Projects view double click **J05CMORT** under **cobol_cics**
 This is the program that you will update



3.1.2 ►| Use **Ctrl + f** to find the “%pot” on line 137 (second Find hit). You will modify the line 141..

J05CMORT.cbl

```

131 * Process Enter Key to calculate the loan amount
132     PERFORM A100-PROCESS-MAP
133     END-IF
134     WHEN OTHER
135 * Invalid key
136     MOVE LOW-VALUES TO JKEMENUO
137 * %pot - below is the message to be changed -
138 * Replace YYYYMMDD Example: 20180712 *
139 *
140     MOVE 'INVALID KEY PRESSED.' TO MSGERRO
141     MOVE 'INVALID KEY PRESSED.' TO MSGERRO
142     SET SEND-DATAONLY TO TRUE
143     PERFORM A300-SEND-MAP
144     END-EVALUATE
145     EXEC CICS
146         RETURN TRANSID(EIBTRNID)
147         COMMAREA(W-COMMUNICATION-AREA)
148         LENGTH(W-COMAREA-LENGTH)
149     END-EXEC.
150

```

Find/Replace

Find: %pot (1)

Replace with:

Direction: Forward Scope: All

Options: Case sensitive (unchecked), Wrap search (checked), Whole word (unchecked), Incremental (unchecked), Regular expressions (unchecked)

Buttons: Find (2), Replace/Find, Replace, Replace All, Close

3.1.3 ►| Close the find dialog and modify the line 141

From

MOVE 'INVALID KEY PRESSED.' TO MSGERRO

To

MOVE 'YYYYMMDD - INVALID KEY PRESSED.' TO MSGERRO

Where **YYYYMMDD** is the today's date.. Example, I changed to **20190529..**

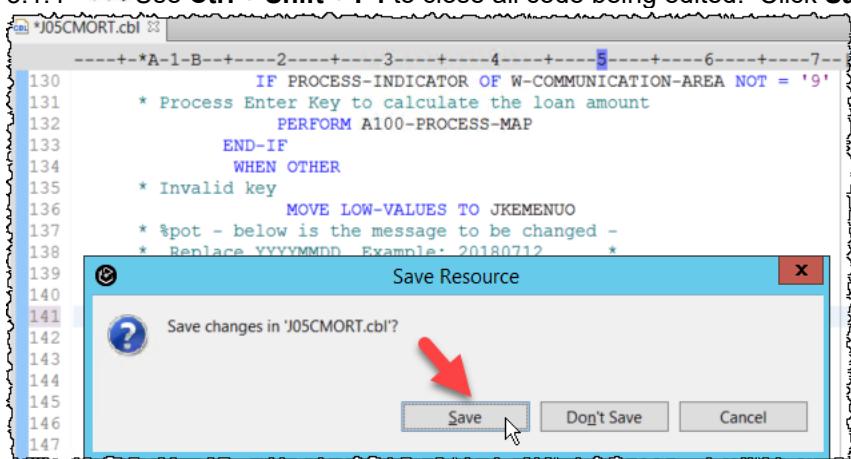
J05CMORT.cbl

```

129 WHEN EIBAID = DFHENTER
130     IF PROCESS-INDICATOR OF W-COMMUNICATION-AREA NOT = '9'
131 * Process Enter Key to calculate the loan amount
132     PERFORM A100-PROCESS-MAP
133     END-IF
134     WHEN OTHER
135 * Invalid key
136     MOVE LOW-VALUES TO JKEMENUO
137 * %pot - below is the message to be changed -
138 * Replace YYYYMMDD Example: 20180712 *
139 *
140     MOVE '20190529 - INVALID KEY PRESSED.' TO MSGERRO
141     MOVE '20190529 - INVALID KEY PRESSED.' TO MSGERRO
142     SET SEND-DATAONLY TO TRUE
143     PERFORM A300-SEND-MAP
144     END-EVALUATE
145     EXEC CICS
146         RETURN TRANSID(EIBTRNID)
147         COMMAREA(W-COMMUNICATION-AREA)

```

3.1.4 ►| Use **Ctrl + Shift + F4** to close all code being edited. Click **Save** to save the modified code.

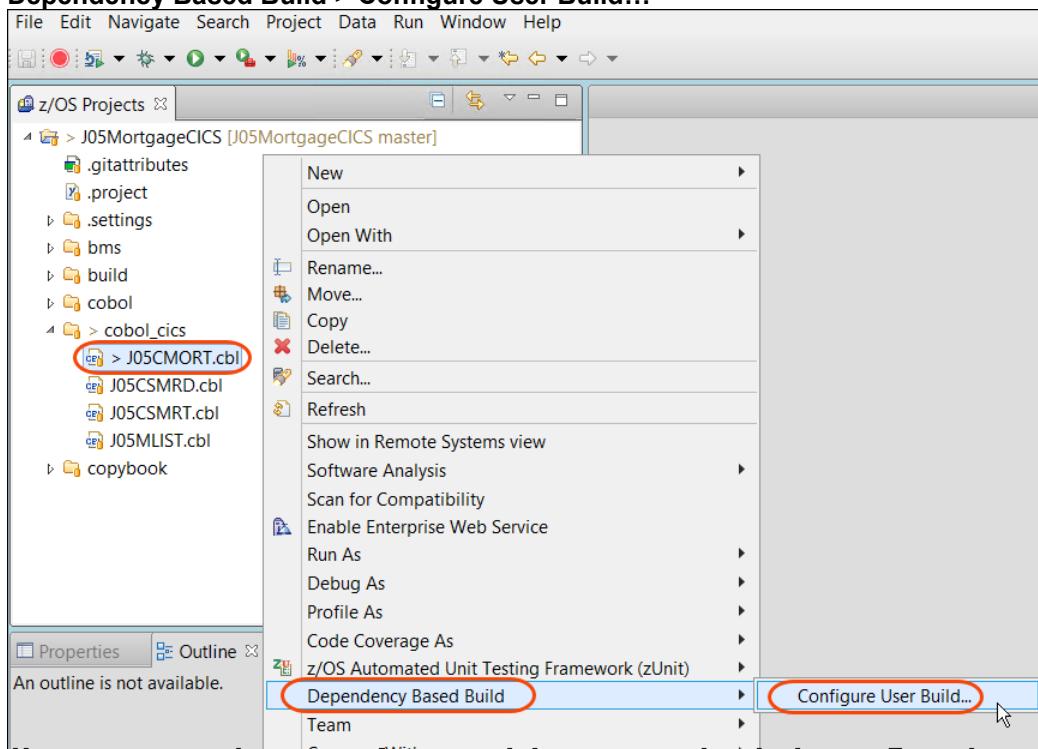


Section 4. Use IDz DBB User Build to compile/bind and perform personal tests.

You will compile and link the modified code using the DBB User Build function. When complete you will run the code using CICS for a personal test and verify that the change is correctly implemented.

4.1 Using Dependency Based Building option

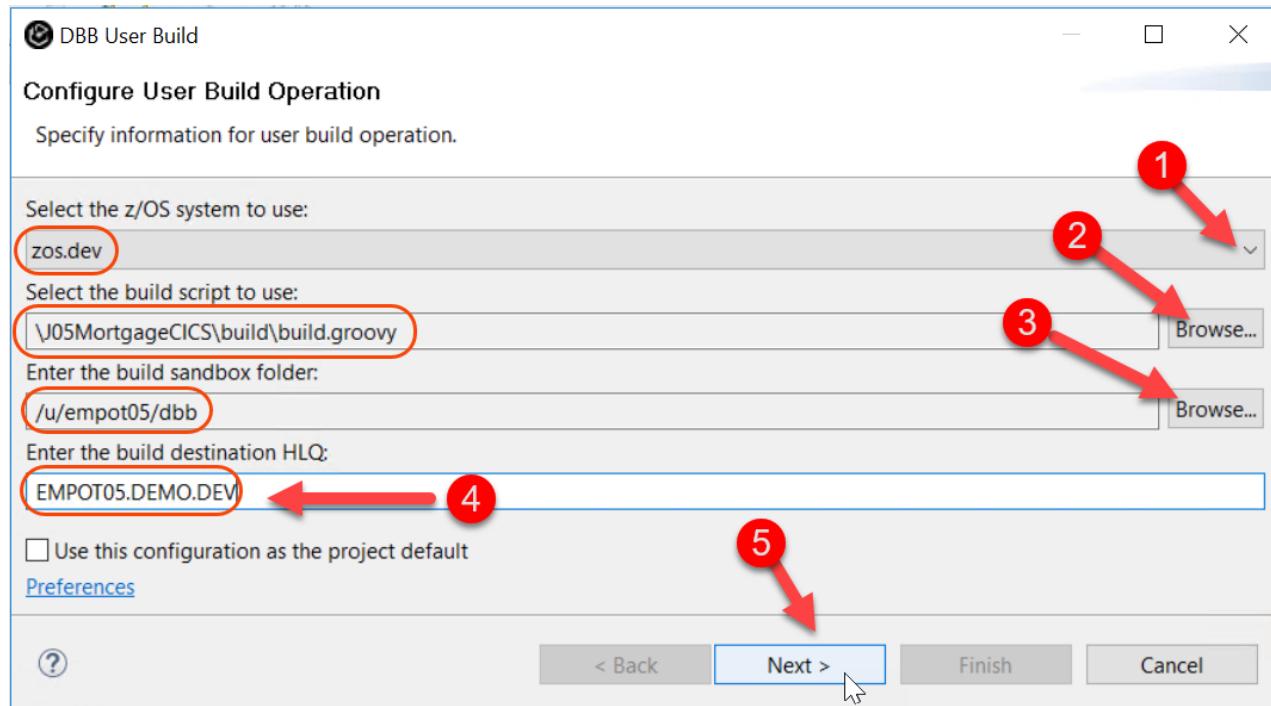
4.1.1 ►| Under **z/OS Projects** view right click **J05CMORT** and select **Dependency Based Build > Configure User Build...**



4.1.2 ► Be sure that those values are already assigned for the build, otherwise use the **Browse** button and select the values as below

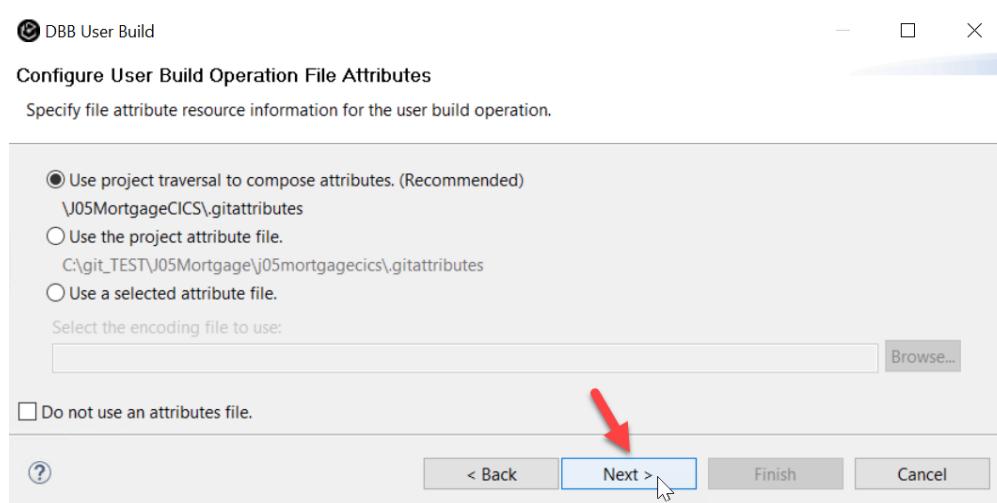
(Tip → Use the notepad and edit `c:\ADF_POT\LAB7\LAB7_Copy_Paste.txt` if want to copy/paste).

- 1 z/OS system: `zos.dev`
- 2 Build script : `\J05MortgageCICS\build\build.groovy` (use **Browse the local workspace**)
- 3 Build sandbox : `/u/empot05/dbb` (Use **Browse**, expand **MyHome**, click **dbb** and **OK**)
- 4 Build destination HLQ: `EMPOT05.DEMO.DEV` (must type)
- 5 ► Click **Next**

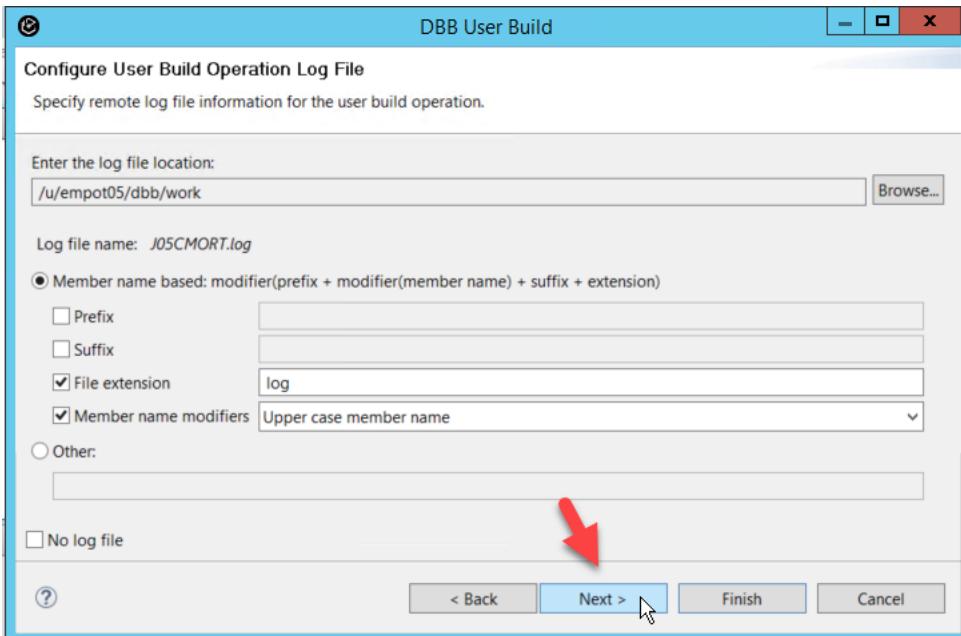


4.1.3 ► On *Configure User Build Operation File Attributes* click **Next**

The `.gitattributes` have the information to translate from *UTF-8* to *EBCIDIC* when moving the code to z/OS.

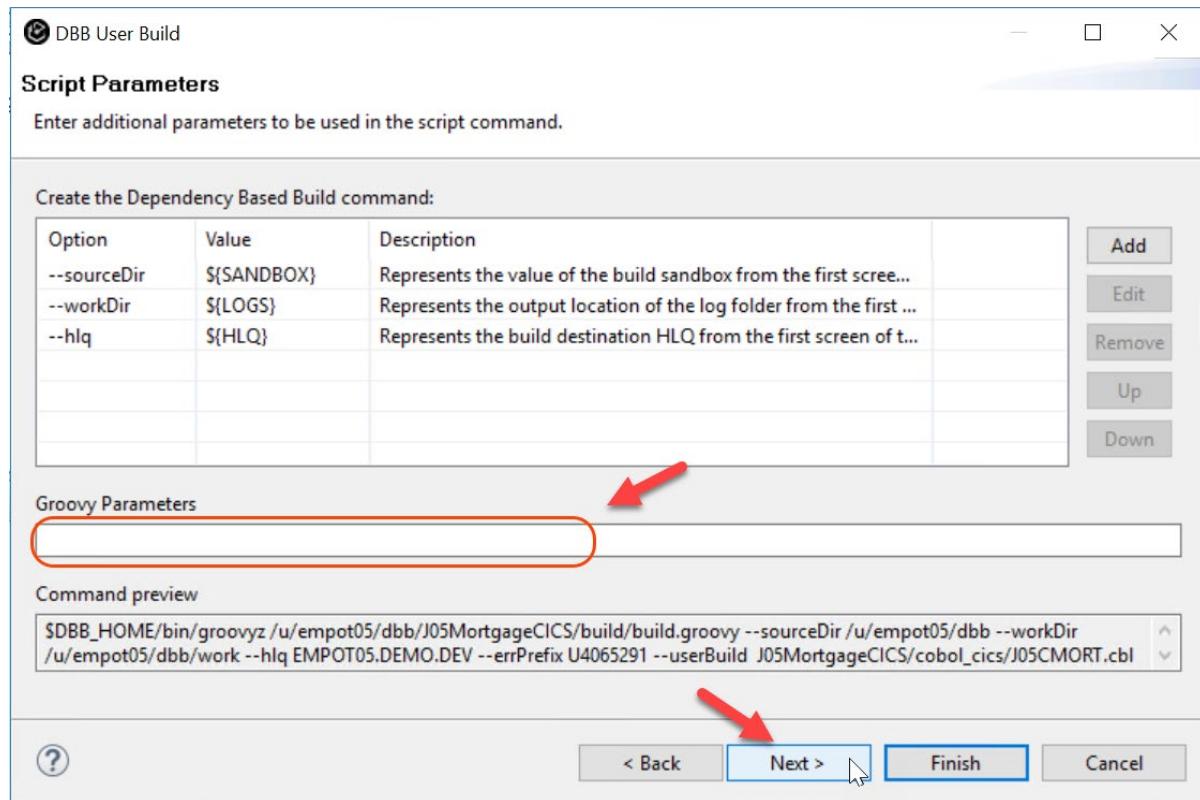


4.1.4 ►| On *Configure User Build Operation Log File* click **Next**
 You will use the default values.



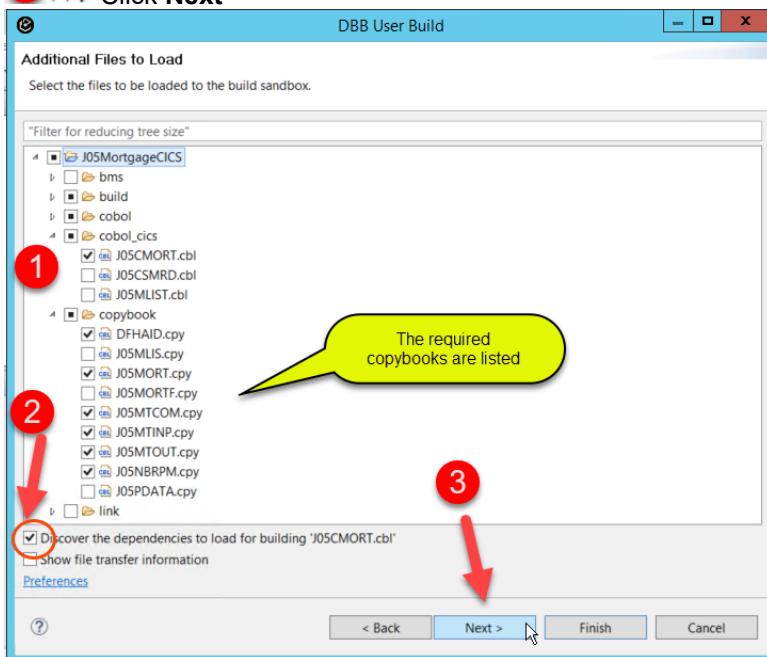
4.1.5 ►| Be sure that the field *Groovy Parameters* is empty and click **Next**

Notice the preview of the commands that will be execute at the z/OS by the DBB toolkit.

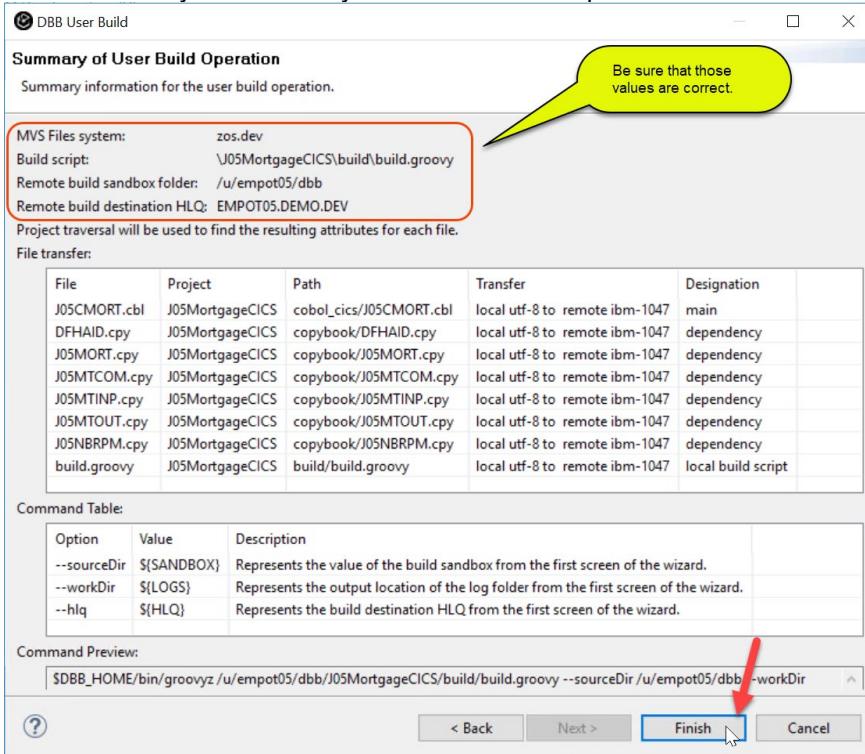


4.1.6 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

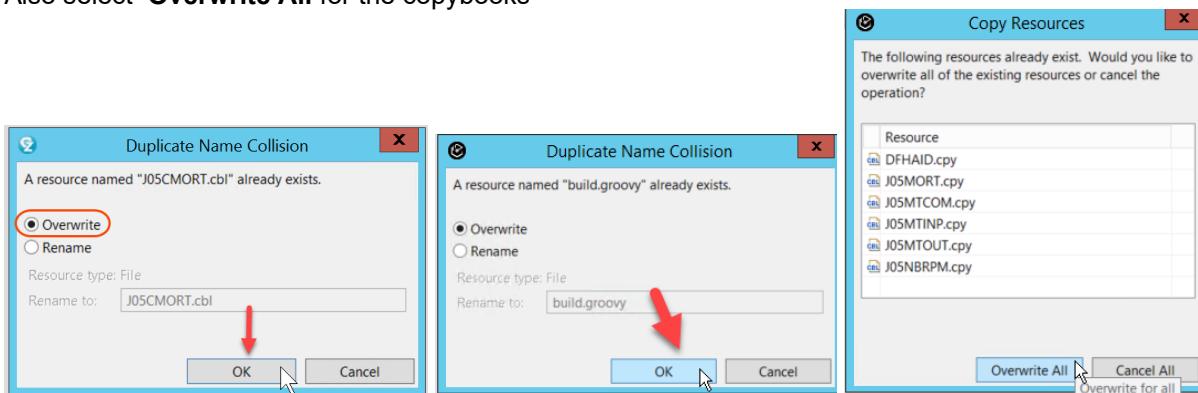
- 1 ►► Expand J05Mortgage CICS and the **cobol_cics** and **copybook** directories
- 2 ►► Since you will also need the related copybooks for a clean compile be sure that “**Discover the dependencies to load for building J05CMORT.cbl**“ is selected (automatically checked).
- 3 ►► Click Next



4.1.7 ►► Verify the summary of the User Build Operation and click **Finish**



4.1.8 ► If the dialogs below pops up select **Overwrite** and click **OK**
Also select **Overwrite All** for the copybooks

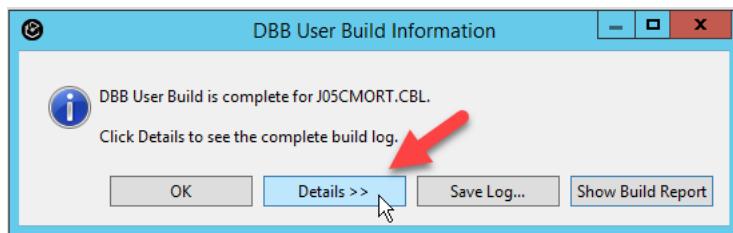


The DBB User build will be running and the messages will be shown at the Console view..

► Click on **Console** tab on lower right corner The execution will start, and this will take a while (2 Minutes) .

4.1.9 When complete you will have the message below:

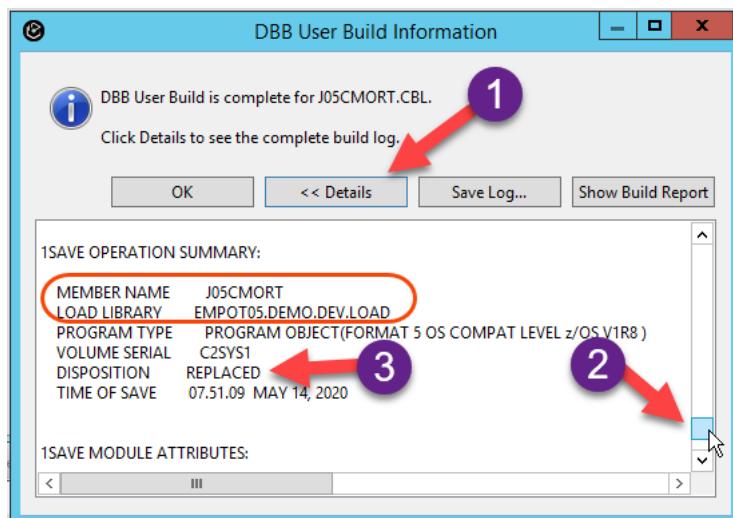
► Click **Details**



4.2 Verify the DBB User Build results

4.2.1 ► After clicking **Details >>** scroll down and verify that a new load module name **J05CMORT** was replaced on the dataset **EMPOT05.DEMO.DEV.LOAD**

► Click **OK** to close this dialog.



- 4.2.2 ► Verify at the Condole view that the Build State should show a **CLEAN** build.

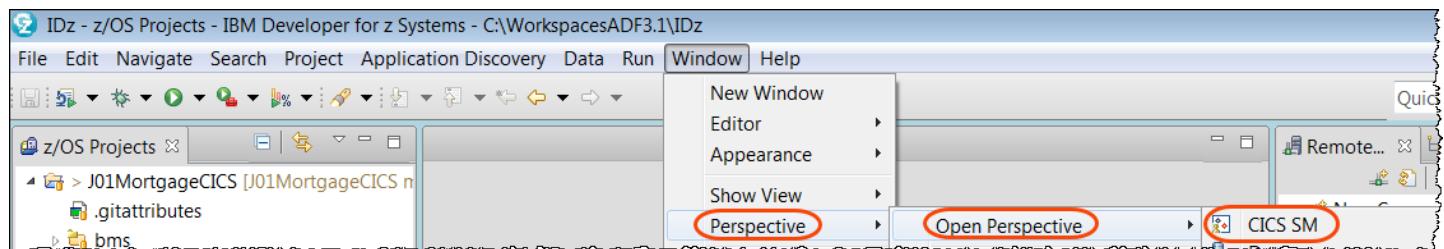
```
** Build State : CLEAN
** Total files processed : 1
** Total build time : 44.516 seconds
/u/empot05/dbb>
** Build finished
/u/empot05/dbb>
```

4.3 Issuing a CICS New copy

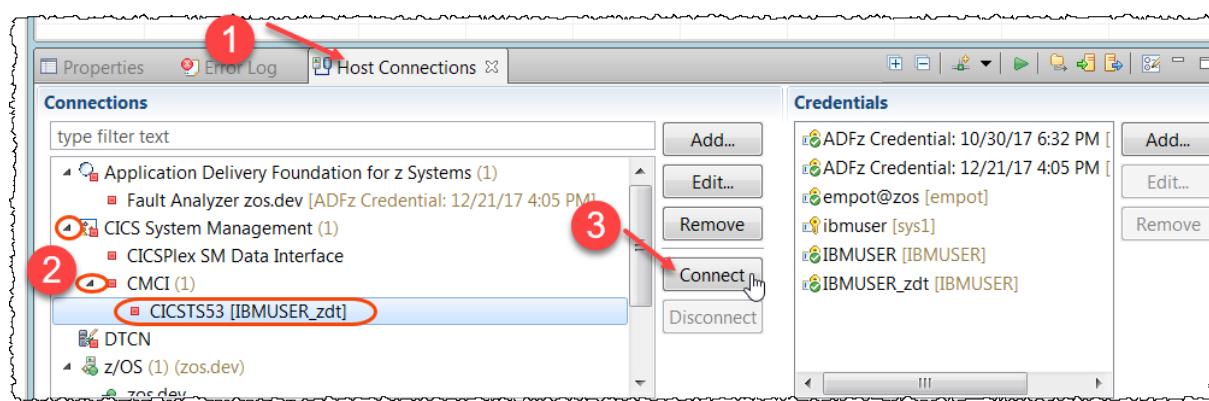
- 4.3.1 At this point you may test your new load module. But since this is a CICS system you will need to make a CICS NEWCOPY. Let's use IDz and CICS Explorer to do that.

Notice that the new copy could be done by the Groovy scripts, but I want to show CICS Explorer nice capabilities here..

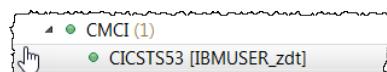
- Open the CICS SM perspective selecting Window > Perspective > Open Perspective > CICS SM



- 4.3.2 ► Click on tab Host Connections on the bottom, expand the CICS System Management, CMCI, select CICSTS53 and click on Connect



- 4.3.3 The red dot will turn green once is connected to CICS



4.3.4 ① ► On top left expand **CICSTS53** and click on **CICSTS53 (CICSTS53)**.

② ► Click on the **Programs** tab.

Since the filter is already made for "Name = J05*" " you should see the list below:

The screenshot shows the IBM Developer for z Systems interface. The title bar reads "IDz - CICS SM - IBM Developer for z Systems - C:\Workspaces\ADF3.1\IDz". The menu bar includes File, Edit, Search, Application Discovery, Data, Operations, Window, Help. The toolbar has various icons for file operations. The left pane is the "CICSplex Explorer" showing a tree structure with "CICSTS53 (1/1)" expanded, revealing "Systems" and "CICSTS53 (CICSTS53)". A red circle with number 1 is around "CICSTS53 (CICSTS53)". The right pane is titled "Regions Tasks Programs ISC/MRO Connections Terminals Local Files Local Transactions". The "Programs" tab is selected, indicated by a red circle with number 2. The sub-tab "Programs" is also highlighted. The main area displays a table titled "CNX0211I Context: CICSTS53. Resource: PROGRAM. 8 (filtered,sorted) records collected at May 29, 2019, 8:36:17 AM". The table columns are Region, Name, Status, Use Count, Concurrent Use C..., Language, Share Status, and CEDF Status. The data rows are as follows:

Region	Name	Status	Use Count	Concurrent Use C...	Language	Share Status	CEDF Status
CICSTS53	J05CMORT	ENABLED	3	0	COBOL	PRIVATE	CEDF
CICSTS53	J05CSMRD	ENABLED	0	0	COBOL	PRIVATE	CEDF
CICSTS53	J05CSMRT	ENABLED	1	0	COBOL	PRIVATE	CEDF
CICSTS53	J05MLIS	ENABLED	0	0	NOTAPPLIC	PRIVATE	NOTAPPLIC
CICSTS53	J05MLISD	ENABLED	0	0	COBOL	PRIVATE	CEDF
CICSTS53	J05MLIST	ENABLED	0	0	COBOL	PRIVATE	CEDF
CICSTS53	J05MORT	ENABLED	3	0	NOTAPPLIC	PRIVATE	NOTAPPLIC
CICSTS53	J05MPMT	ENABLED	1	0	COBOL	PRIVATE	CEDF

4.3.5 ► Right click on **J05CMORT** and select **New Copy** as the example below

The screenshot shows a context menu for the program "J05CMORT" in the "Programs" table. The menu options are: Open, Phase In, New Copy (highlighted with a red circle), Add Quick Filter, Copy (Ctrl+C), Discard, Delete, and Enable.

4.3.6 ► Select OK

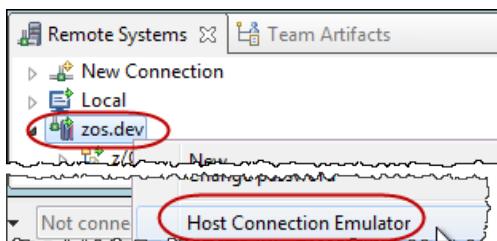
The screenshot shows the "Perform Operation" dialog box. The title is "Perform NEWCOPY Operation". It states "NEWCOPY operation will be performed on 1 item in the execution queue". The action summary shows "Name: NEWCOPY" and "Parameters: J05CMORT". At the bottom are "OK" and "Cancel" buttons, with a red arrow pointing to the "OK" button.

4.3.7 Let's start the 3270 emulation again ..

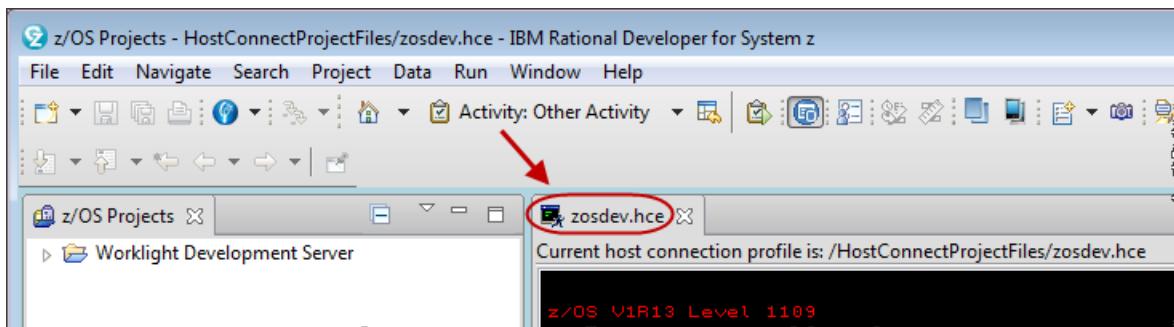
► Go to the **z/OS Projects** perspective

The screenshot shows the Eclipse IDE interface. The title bar says "Eclipse IDE for Enterprise Java Developers". The quick access bar at the top has several icons. The "z/OS Projects" icon is highlighted with a red circle. Below the quick access bar are tabs for "Local Files" and "Local Transactions".

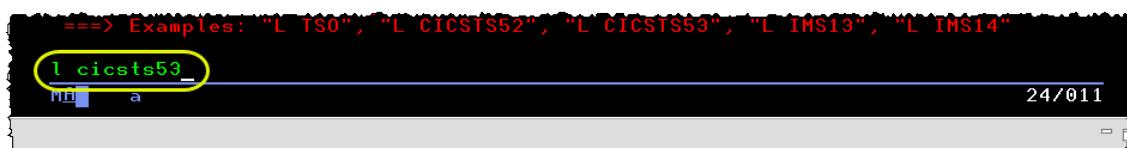
4.3.8 ► Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator**.



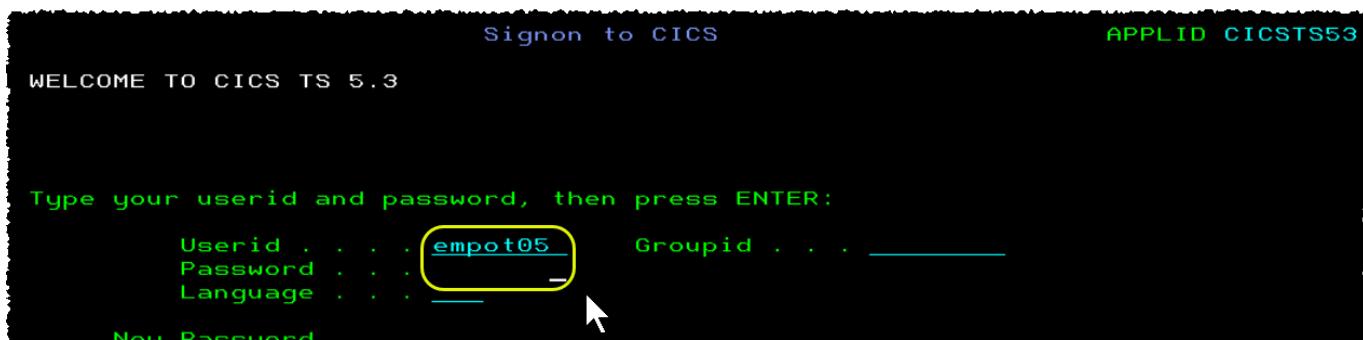
4.3.9 ► Since you will need more space, **double-click** on the **zos.dev.hce** title



4.3.10 ► Type **I cicsts53**. and press **Enter**.



4.3.11 ► Logon using **empot05** and password **empot05** and press **Enter**.



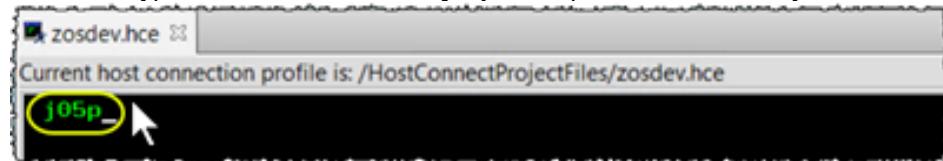
4.3.12 The sign-on message is displayed



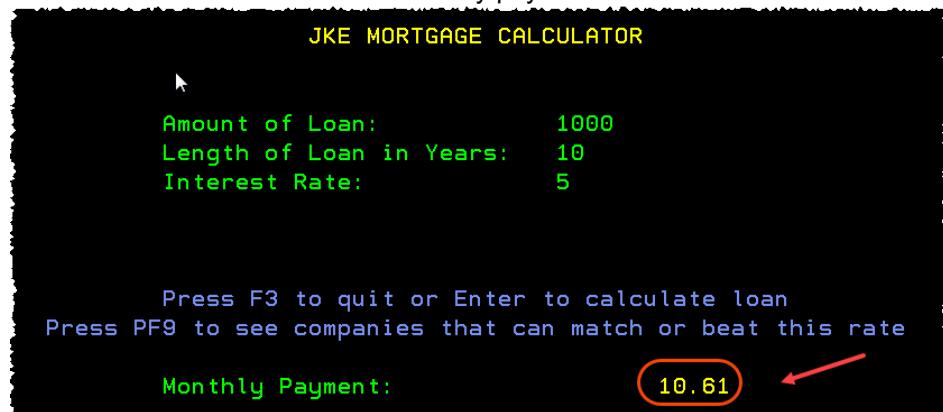
4.4 Running *j05p* CICS transaction

You should now be on the z/OS CICS region named *CICSTS53*. This is the CICS instance where you made the program changes.

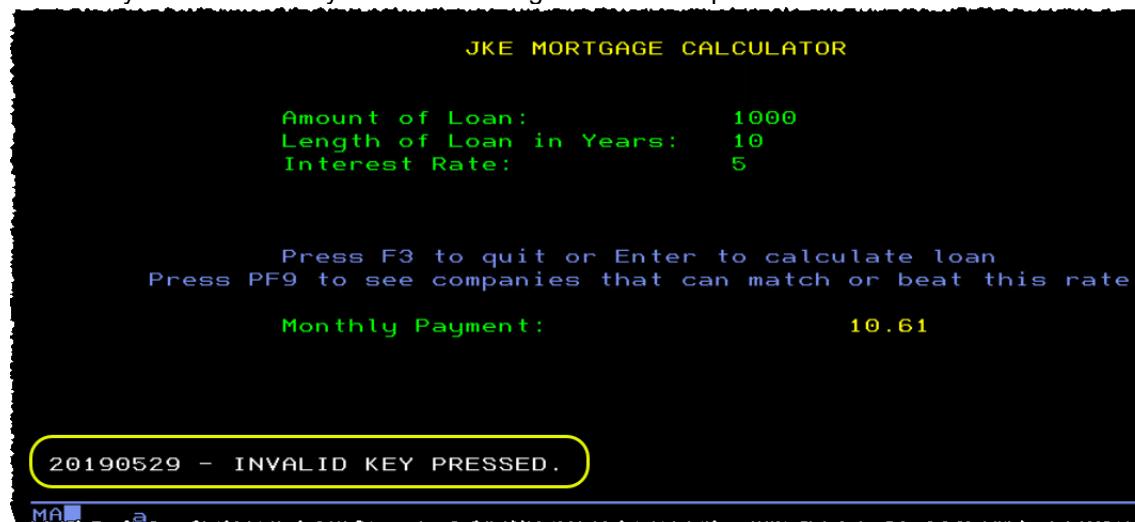
- 4.4.1 ► Type the CICS transaction **j05p** and press the **Enter** key.



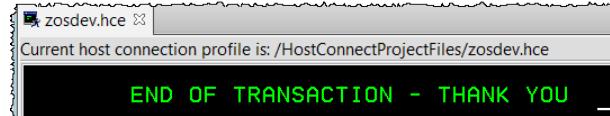
- 4.4.2 ► Press **enter** to see the monthly payment for the default data .



- 4.4.3 ► Press **F1** key and verify the message displayed
This was your mission. As you see the change has been implemented.



- 4.4.4 ► Press **F3** to end the CICS transaction.



- 4.4.5 ► Use **Ctrl + Shift + F4** to close the terminal emulation.

Section 5. Push and Commit the changed code to Git.

Using IDz you will commit the changes you made to Git and later perform the final building and deploy using Jenkins and UCD.

5.1 Using IDz with the Git plugin to compare the change versus original

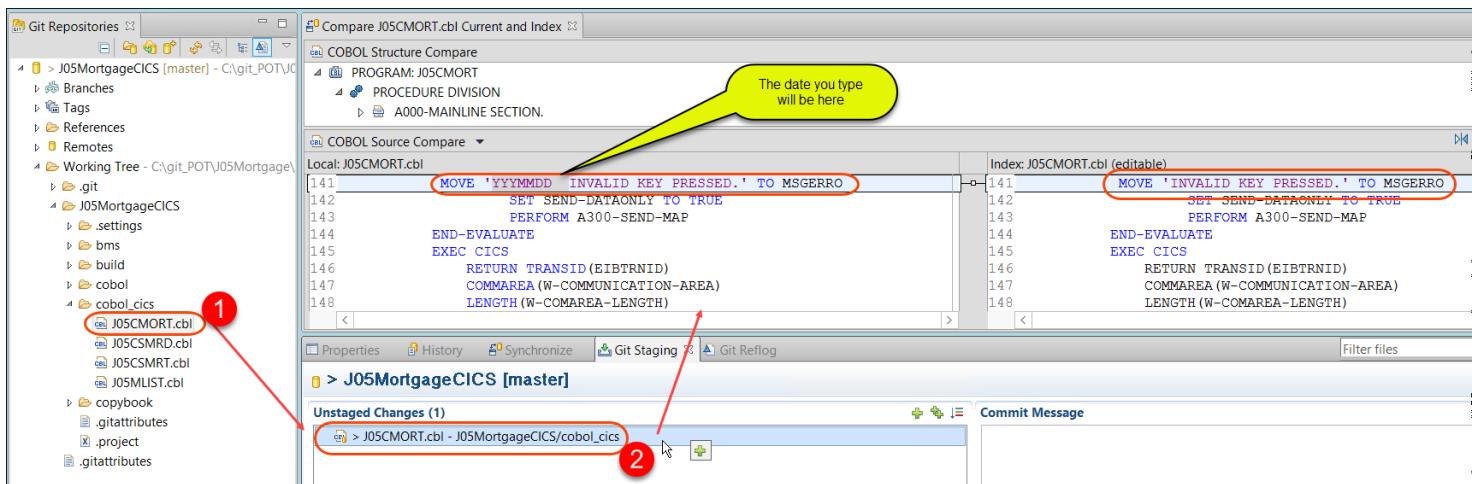
5.1.1 ► Switch to the **Git Perspective** selecting the icon on top and right



5.1.2 ► ① Expand the nodes and click on the program **J05CMORT.cbl** that you have modified. Notice that the changed program is listed in the *Git Staging* view...

► ② Double click on the **J05CMORT.cbl** that is listed under **Unstaged Changes** and noticed the comparison among the program that you updated versus the one that is in the Git repository.

You will need to commit the changes to the Git repository to make a final build later.

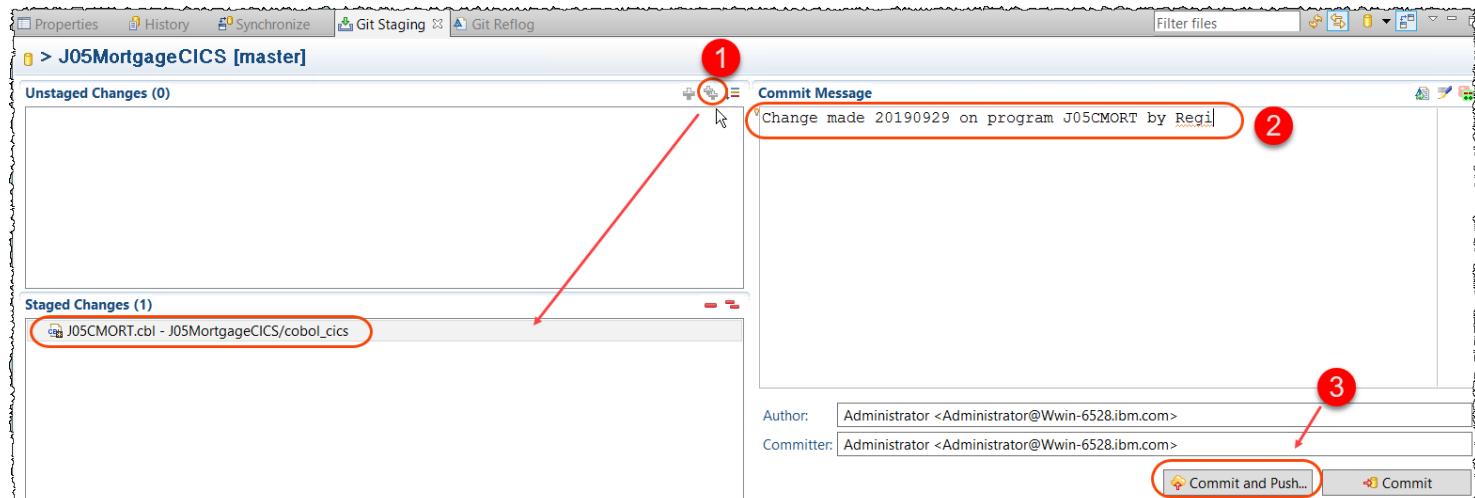


5.1.3 ► Use **Ctrl + Shift + F4** to close the editors.

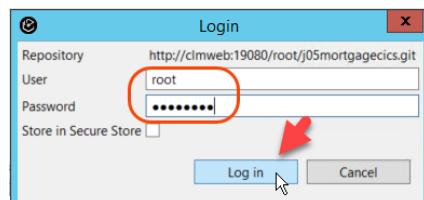
5.2 Push and Commit the changes to Git

5.2.1 ► Using Git Staging view ,

- 1 Click on the icon 
- 2 Write a commit message like: **Change made yyyymmdd on program J05CMORT by your name**
- 3 Click **Commit and Push ...**



5.2.2 ► If it prompts for login credentials use **root** and password **zdtlinux** and click **Log in**



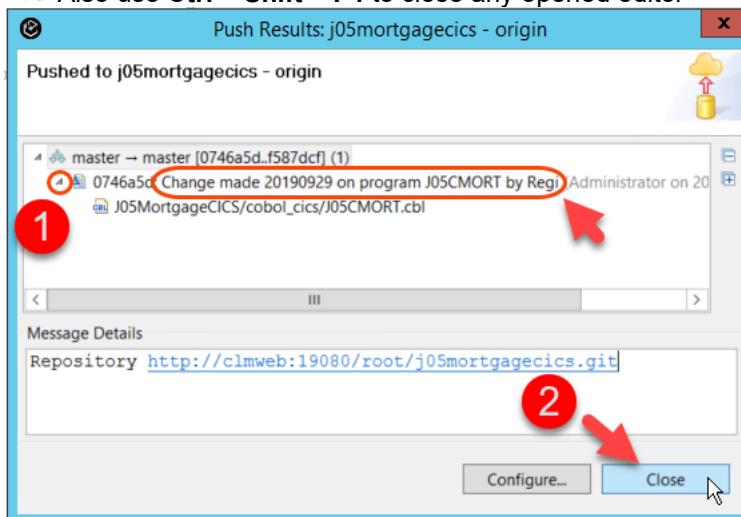
5.2.3 To see the changes you committed to Git:

- 1 Double click on **Git hash string** and 2 select **Diff** tab

The screenshot shows two windows. The left window is titled 'Commit 3e24aa9d866d27ec96' and displays a diff between 'master' and 'master'. A red circle highlights the 'Diff' tab at the bottom. The right window is titled 'Push Results: j05mortgagecics - origin' and shows a log entry for the commit. A red circle highlights the commit message: '3e24aa9: Change made 20201202on program J05CMORT by Regi (Administrator on 2020-12-02)'. Both windows have red arrows pointing to their respective circled areas.

5.2.4 ► Verify that the commit was successful and click **Close**

► Also use **Ctrl + Shift + F4** to close any opened editor



5.2.5 ► To see this changes in **GitLab** use the **Firefox browser** and do a **Refresh (F5)** (details how to get the link at 2.1.1).

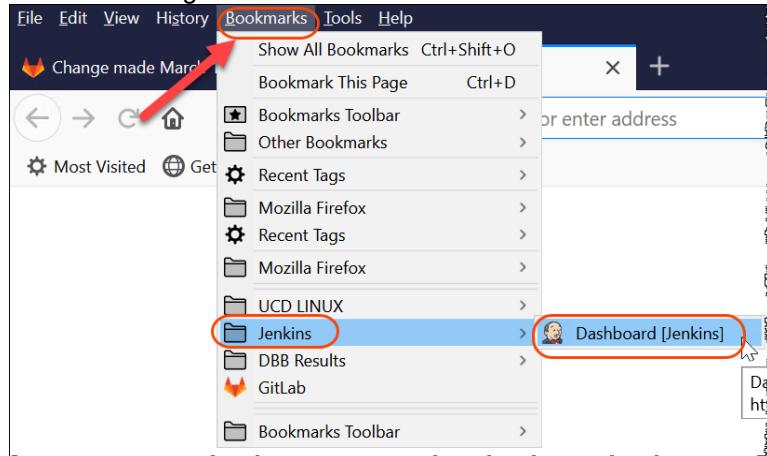
5.2.6 ► You also can click on **Change made yyyyymmdd on program J05CMORT by xxxx** to see the changes

Section 6. Use Jenkins with Git plugin to build all the modified code committed to Git.

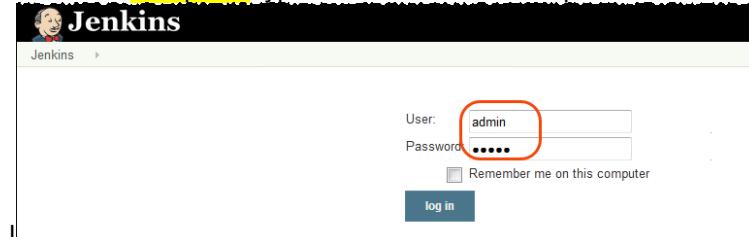
At this point the changed code is delivered into the Git repository that is on Linux. You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD. Note that this step makes a build of all code committed to Git, while on Section 4 only the selected COBOL program was built.

6.1 Logon to Jenkins using a Web Browser and start the z/OS agent

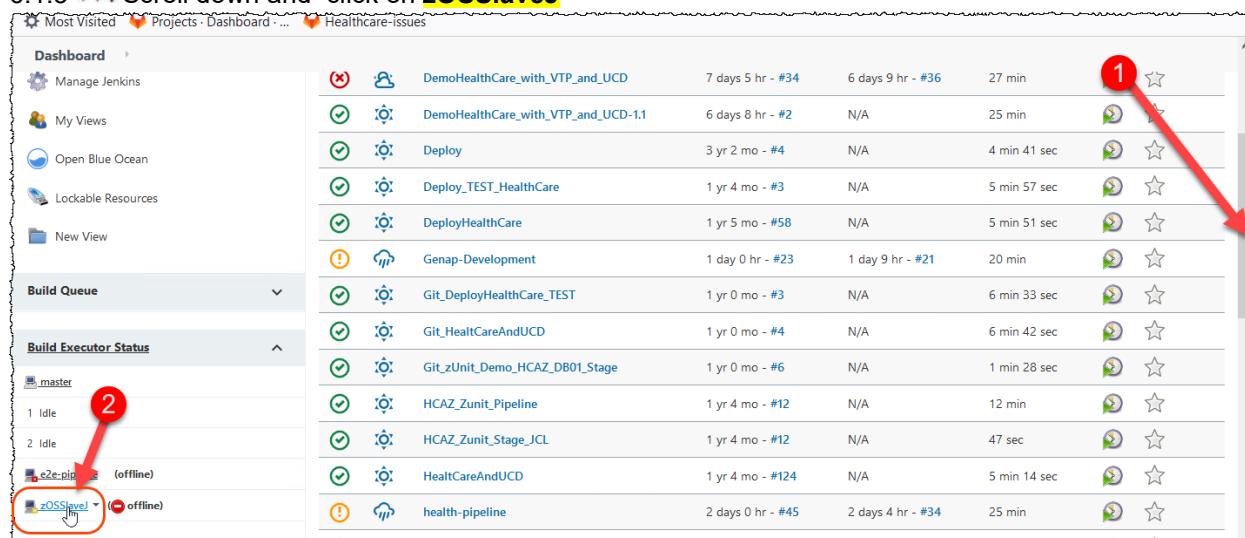
6.1.1 Using the Bookmarks click on Jenkins > Dashboard



6.1.4 If required type the user admin and password admin (lowercase) and click log in.



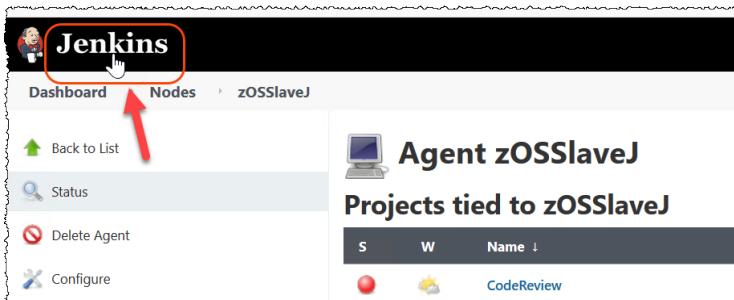
6.1.5 Scroll down and click on zOSSlaveJ



- 6.1.6 ► On the right corner click on the **Bring this node back online** “blue button”. This will bring the Jenkins agent online again.



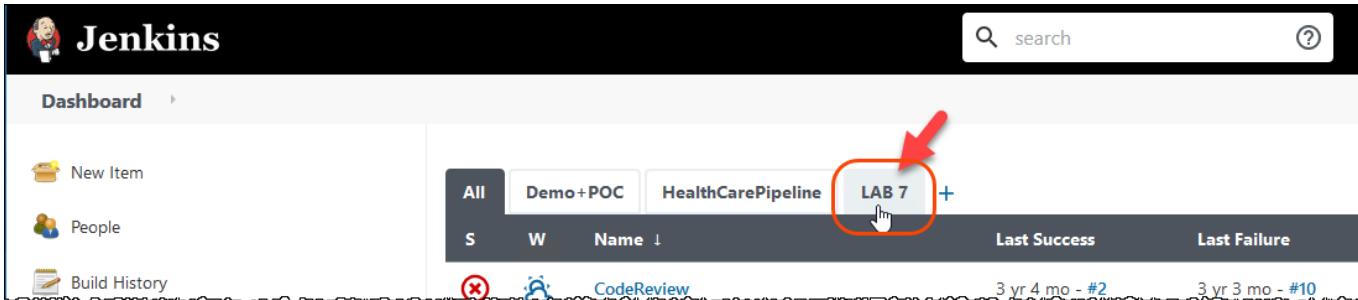
- 6.1.6 ► Click on Jenkins on top left:



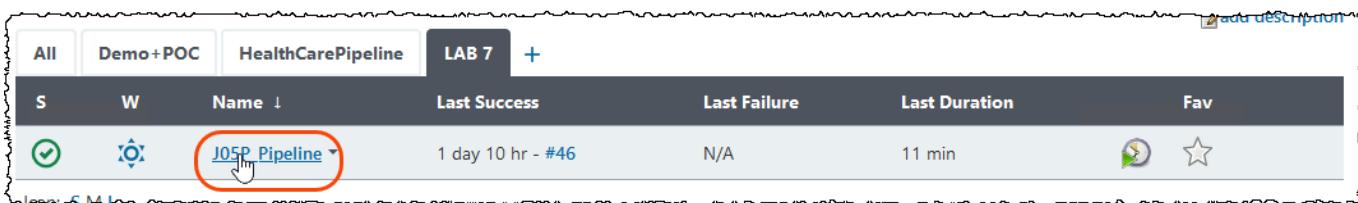
6.2 Starting the Jenkins Pipeline

- 6.2.1 The homepage lists all of the Jenkins jobs and pipelines.

- Click LAB 7 view



- Click on the hyperlink for the project **J05P_Pipeline** that represents your CICS application.



- 6.2.2 This opens the **J05P_Pipeline** view. This pipeline has been designed with two stages:

- 1 – **Stage 1** builds ONLY the changed code from Git. This is an example of a **Build pipeline**.
- 2 – **Stage 2** deploys the changes into CICS Development Region (Dev) using UrbanCode Deploy (UCD). This is an example of a **Delivery pipeline**.

This pipeline is a simplistic form of an actual delivery pipeline. Any other activity such as, test cases, code reviews, batch job runs, rexx scripts, table queries and other manual tasks that are currently being carried out in a shop before code is moved to QA (or any other region) can be built into a pipeline.

► Click on **Build Now** on the left.

IMPORTANT : This Jenkins Build will take at least **14 Minutes** since the cloud instances are slow compared to a real environment.

If you are running late and don't want to wait 14 minutes, you can see a build done already (example **Building #36**) or just read the steps described here.

6.2.3 ① Notice that the new build has appeared under **Build History** indicating the currently started Pipeline.

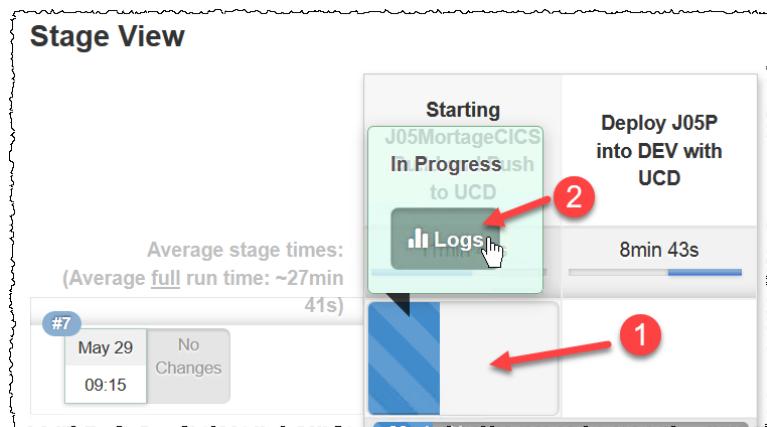
② The **Stage View** also shows a new line with an in-progress bar.

6.3 Checking the results

6.3.1 Since this z/OS is running in a small processor in the cloud, this activity may take 10 or more minutes, but you don't need to wait to be completed to check the results.

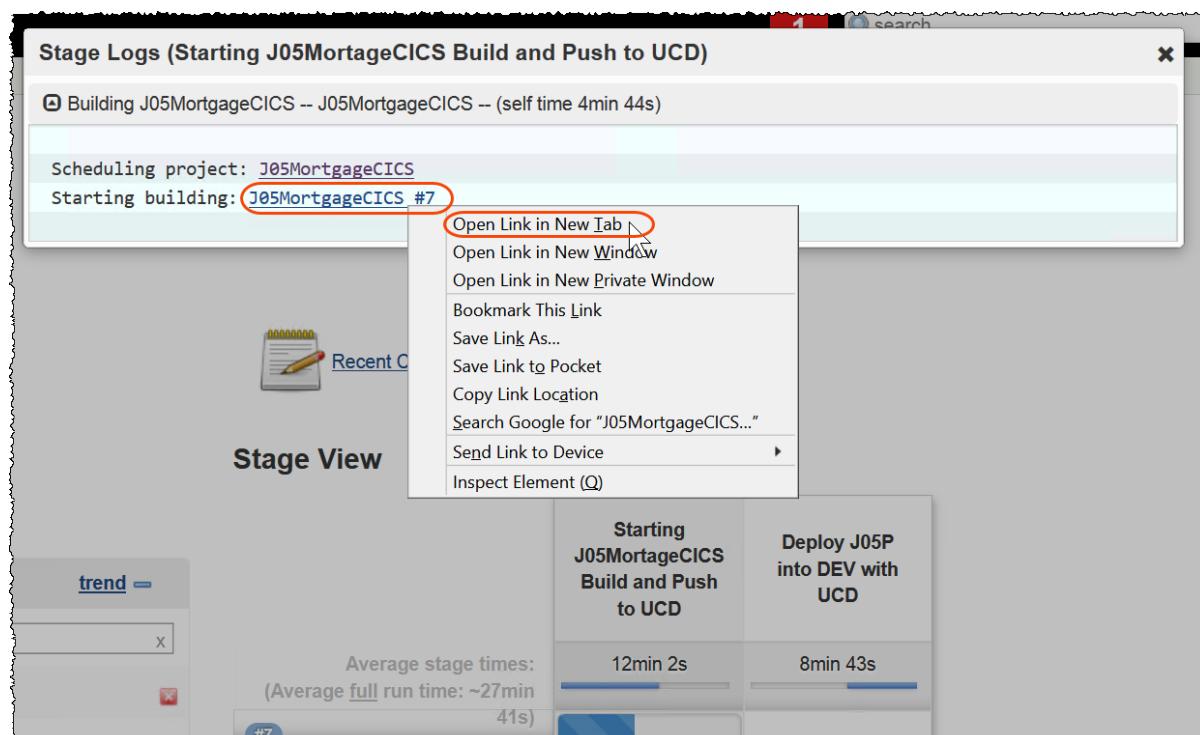
You can follow what is going on as described below ..

► Move the cursor and click on the blue area ① and then click on Logs ②.



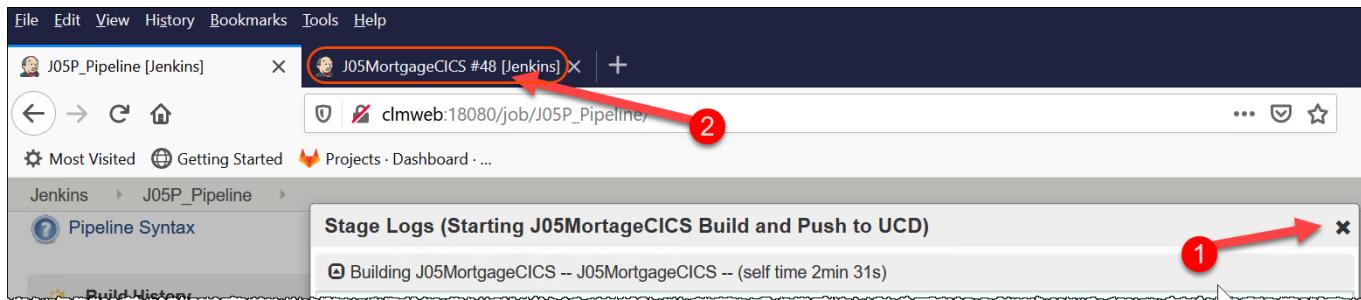
6.3.2 The dialog Stage Logs will open.

► Right click on J05MortgageCICS #nn and select Open Link in New Tab



6.3.3 This opens a new Jenkins tab as shown below.

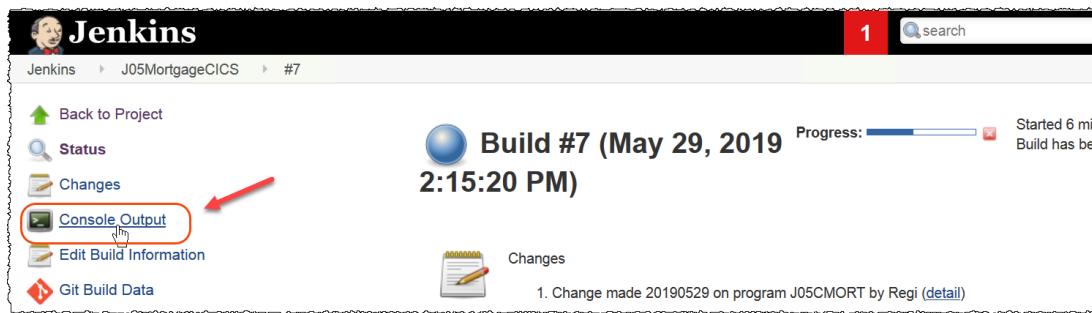
▶ Close Stage Logs dialog and click on the newly opened tab (**J05MortgageCICS#nn [Jenkins]**)



6.3.4 This opens the current running task for the first step of the J05Mortgage Application Build.

▶ Click on the **Console Output** to view the log.

Tip: You may use **Ctrl +** to make bigger fonts and **Ctrl –** to make it smaller



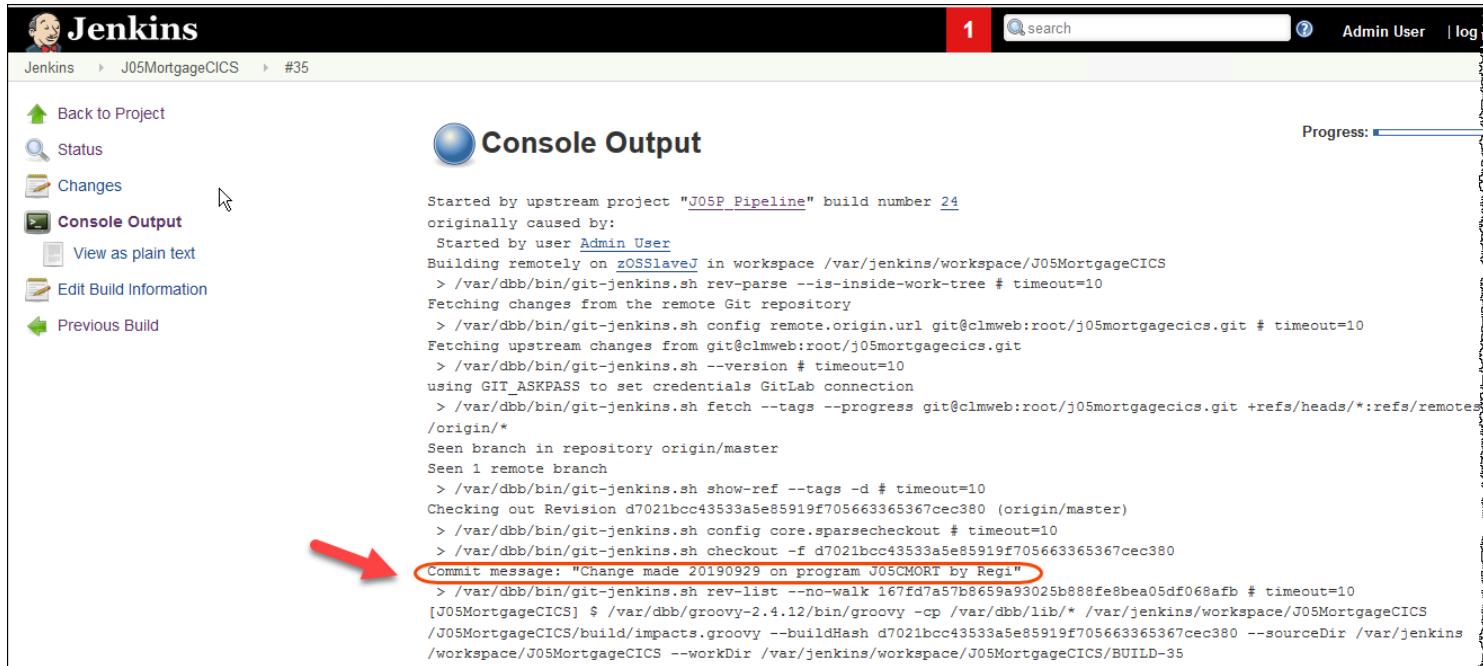
6.3.5 The Console Output log of first pipeline shows the various Groovy scripts being executed.

You need to understand what is going on. Below some explanation.

IMPORTANT → The Jenkins agent may take a while to start on z/OS.. BE PATIENT..

Notice that the comment that you added when you commit to Git is displayed.

This is what we call “smart building” since ONLY the modified code will be built.



Your commit message is not what you expected?

Jenkins agent uses SSH to connect to GitLab on Linux. Since you are using the z/OS on cloud this SSH connection may fail and the code committed is not being considered.

This happens when the "known_hosts" file located at "/u/ibmuser/.ssh" has an incorrect value. Each zOS instance must have a unique value and you need to generate a new one.

You will need to delete known_hosts file and create a new one to fix that issue.

The sequence to be done is:

1. Use the terminal emulation and type **L TSO**.
2. Logon as **IBMUSER** and password **SYS1**.
3. Go to OMVS using the option **6** and typing **OMVS**.
4. Execute the script typing **sshc.sh** if prompt for host authentication answer yes and press enter 3 times for the password.

If you cannot find this script, execute the statements below (this is the content of the script provided)

```
#This script will add Linux ip to known_hosts
echo "-----"
echo "--> This script will add Linux ip to known_hosts"
echo "--> Msg authenticity of host 'clmweb' can't be established will be displayed"
echo "--> When prompt yes/no answer yes"
echo "--> When prompt for password press enter 3 times"
echo "-----"
cd /u/ibmuser/.ssh
rm known_hosts
ssh clmweb
```

5. Exit OMVS and LOGOFF the TSO.



6.4 Understand the results

This task performs five sub-steps.

► Scroll down and locate the highlighted portions shown below from the build console output to know more about the build

6.4.1 Check out of code from Git Repositories based on the latest commit

```
Building remotely on zOSSlaveJ in workspace /var/jenkins/workspace/J05MortgageCICS
> /var/dbb/bin/git-jenkins.sh rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> /var/dbb/bin/git-jenkins.sh config remote.origin.url git@clmweb:root/j05mortgagecics.git # timeout=10
Fetching upstream changes from git@clmweb:root/j05mortgagecics.git
> /var/dbb/bin/git-jenkins.sh --version # timeout=10
using GIT_ASKPASS to set credentials GitLab connection
> /var/dbb/bin/git-jenkins.sh fetch --tags --progress git@clmweb:root/j05mortgagecics.git +refs/heads/*:refs/remotes/origin/*
Seen branch in repository origin/master
Seen 1 remote branch
> /var/dbb/bin/git-jenkins.sh show-ref --tags -d # timeout=10
Checking out Revision d7021bcc43533a5e85919f705663365367cec380 (origin/master) (arrow pointing here)
> /var/dbb/bin/git-jenkins.sh config core.sparsecheckout # timeout=10
> /var/dbb/bin/git-jenkins.sh checkout -f d7021bcc43533a5e85919f705663365367cec380
Commit message: "Change made 20190929 on program J05CMORT by Regi"
> /var/dbb/bin/git-jenkins.sh rev-list --no-walk 167fd7a57b8659a93025b888fe8bea05df068afb # timeout=10
[J05MortgageCICS] $ /var/dbb/groovy-2.4.12/bin/groovy -cp /var/dbb/lib/* /var/jenkins/workspace/J05MortgageCICS
```

6.4.2 **An Impact Analysis** to identify if any programs have been changed since the last build using DBB Impact scripts. The DBB application server that is installed on Linux keeps track of all builds.

```
** Impact analysis start at 20201202.080335.003
** Searching for last successful build commit hash for build group J05MortgageCICS
Last successful build commit hash located. label : build.20201202.071030.010 , buildHash :
167fd7a57b8659a93025b888fe8bea05df068afb
** Executing Git command: git diff --name-only 167fd7a57b8659a93025b888fe8bea05df068afb
d7021bcc43533a5e85919f705663365367cec380
Number of changed files detected since build build.20201202.071030.010 : 1
J05MortgageCICS/cobol_cics/J05CMORT.cbl

** Scan the changed file list to collect the latest dependency data
Scanning changed file J05MortgageCICS/cobol_cics/J05CMORT.cbl
** Store the dependency data in repository collection 'J05MortgageCICS'
HTTP/1.1 200 OK
** Creating build list by resolving impacted programs/files for changed files
Found build script mapping for J05MortgageCICS/cobol_cics/J05CMORT.cbl. Adding to build list.
** Writing buildlist to /var/jenkins/workspace/J05MortgageCICS/BUILD-35/buildlist.txt
** Impact analysis finished at Wed Dec 02 20:04:05 GMT 2020
** Total # build files calculated = 1
** Total analysis time : 30.138 seconds
[J05MortgageCICS] $ /var/dbb/groovy-2.4.12/bin/groovy -cp /var/dbb/lib/* /var/jenkins/workspace/J05MortgageCICS
/J05MortgageCICS/build/build.groovy --buildHash d7021bcc43533a5e85919f705663365367cec380 --sourceDir /var/jenkins
/workspace/J05MortgageCICS --workDir /var/jenkins/workspace/J05MortgageCICS/BUILD-35 --hlq EMPOT05.DEMO.DEV /var/jenkins
/workspace/J05MortgageCICS/BUILD-35/buildList.txt
```

6.4.3 **A dependency scan** to pick out all the dependencies of the changed programs, The DBB application server that is installed on Linux keeps track of all dependencies.

```
** Scan the changed file list to collect the latest dependency data
Scanning changed file J05MortgageCICS/cobol_cics/J05CMORT.cbl
** Store the dependency data in repository collection 'J05MortgageCICS'
HTTP/1.1 200 OK
** Creating build list by resolving impacted programs/files for changed files
Found build script mapping for J05MortgageCICS/cobol_cics/J05CMORT.cbl. Adding to bu
** Writing buildlist to /var/jenkins/workspace/J05MortgageCICS/BUILD-35/buildlist.txt
** Impact analysis finished at Wed Dec 02 20:04:05 GMT 2020
** Total # build files calculated = 1
** Total analysis time : 30.138 seconds
[J05MortgageCICS] $ /var/dbb/groovy-2.4.12/bin/groovy -cp /var/dbb/lib/* /var/jenkins/workspace/J05MortgageCICS
```



This collection is on the DBB App Server

6.4.4 **A compile/link-edit** of the changed programs

```
** Invoking build scripts according to build order: Compile, LinkEdit, CobolCompile
* Building J05MortgageCICS/cobol_cics/J05CMORT.cbl using CobolCompile.groovy script
Copying /var/jenkins/workspace/J05MortgageCICS/J05MortgageCICS/cobol_cics/J05CMORT.cbl to
EMPOT05.DEMO.DEV.COBOL(J05CMORT)
Resolving dependencies for file J05MortgageCICS/cobol_cics/J05CMORT.cbl and copying to EMPOT05.DEMO.DEV.COPYBOOK
Compiling and link editing program J05MortgageCICS/cobol_cics/J05CMORT.cbl
** Build finished at Wed Dec 02 20:06:21 GMT 2020
** Build State : CLEAN
** Total files processed : 1
** Total build time : 1 minutes, 23.097 seconds
```

6.4.5 **Creation of a deployable binary version** of code that can be used by *UrbanCode Deploy* to deploy to the necessary environments. Here Jenkins is using the UCD plugin to store an executable version at UCD server for later deploy.

```

** Create version start at 20201202.080702.007
** Properties at startup:
  component -> J05MortgagePOT
  startTime -> 20201202.080702.007
  workDir -> /var/jenkins/workspace/J05MortgageCICS/BUILD-35
  buztoolPath -> /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh
** Read build report data from /var/jenkins/workspace/J05MortgageCICS/BUILD-35/BuildReport.json
** Find deployable outputs in the build report
  EMPOT05.DEMO.DEV.LOAD(J05CMORT), LOAD
** Generate UCD ship list file
** Write ship list file to /var/jenkins/workspace/J05MortgageCICS/BUILD-35/shiplist.xml
** Create version by running UCD buztool
** Ignore > Error adding properties to version in UrbanCode Deploy server
/etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh createzosversion -c J05MortgagePOT -s /var/jenkins/workspace/J05MortgageCICS/BUILD-35/shiplist.xml -o /var/jenkins/workspace/J05MortgageCICS/BUILD-35/buztool.output
zos toolkit config : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,20170906-2200)
zos toolkit binary : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,20170906-2200)
zos toolkit data set : BUZ626 (6.2.6,20170907-0249)
Reading parameters:
....Command : createzosversion
....Component : J05MortgagePOT
....Generate version name : 20201202-200731
....Shiplist file : /var/jenkins/workspace/J05MortgageCICS/BUILD-35/shiplist.xml
....Output File:/var/jenkins/workspace/J05MortgageCICS/BUILD-35/buztool.output
Verifying version
....Repository location : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20201202-200731
Pre-processing shiplist:
....Shiplist after processing :/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20201202-200731/shiplist.xml

```



Ignore the error message below. This does not impact the UCD version creation..

```

....Shiplist file : /var/jenkins/workspace/J05MortgageCICS/BUILD-35/shiplist.xml
....Output File:/var/jenkins/workspace/J05MortgageCICS/BUILD-35/buztool.output
Verifying version
....Repository location : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20201202-200731
Pre-processing shiplist:
....Shiplist after processing :/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20201202-200731
/shiplist.xml
Packaging data sets:
....Location to store zip : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20201202-200731
....Zip name : package.zip
....EMPOT05.DEMO.DEV.LOAD.bin
....Elapsed time for data set package or deploy operation : 5.073116
Post-processing package:
PackageManifest file post-processing completed.
Create version and store package:
....Error adding properties to version in UrbanCode Deploy server

```



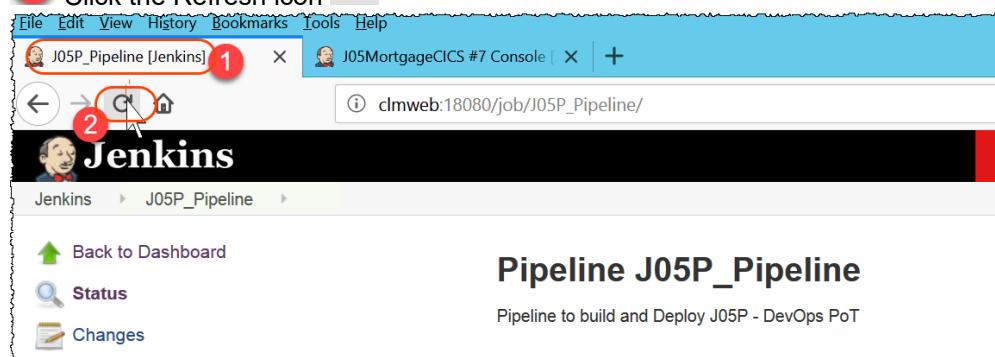

Section 7. Use Jenkins and UCD plugin to deploy results and test the CICS transaction again

You will verify the results of the second stage (deploy using UCD) .

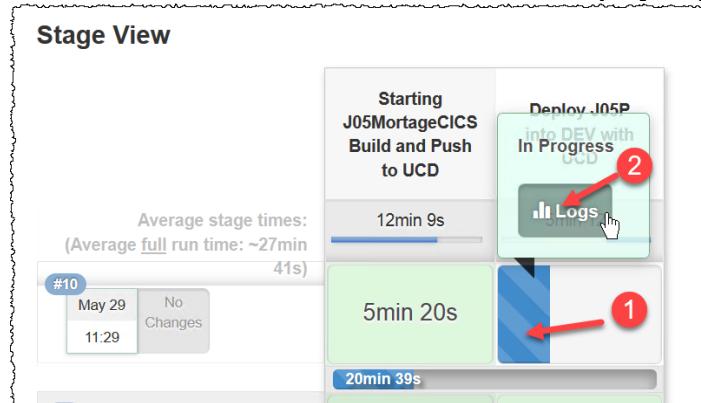
7.1 Checking the deploy results (second stage)

- 7.1.1 ► 1 Click on the previous browser tab **J05P_Pipeline[Jenkins]**

- 2 Click the Refresh icon 

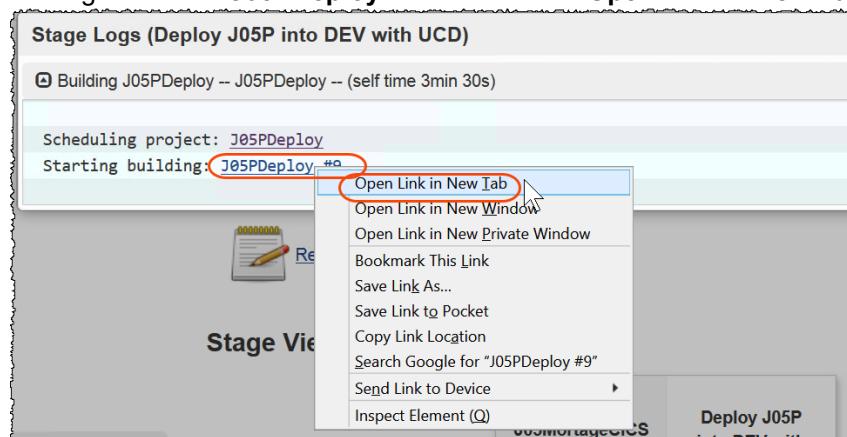


- 7.1.2 ► Click as show in the **blue area** of the Deploy stage 1 and then click on **Logs** 2.



- 7.1.3 The dialog Stage Logs will open.

- Right click on **J05PDeploy #nn** and select **Open Link in New Tab**



7.1.4  Close Stage Logs and click on that J05PDeploy #nn



7.1.5  Click on the Console Output to view the logs.

7.1.6 The Console Output log of second pipeline shows that the program that you delivered will be deployed to CICS Dev environment using UCD..

 You don't need to wait for the deploy finished. Go ahead and check UCD in action (step 7.2.1 below).

7.2 Checking UrbanCode Deploy logs

7.2.1  Start UCD console using the Bookmarks below

7.2.2 ► If you are not logged as **admin**, **Sign Out** and **Login** again using **admin** and password **admin** and click **Login**



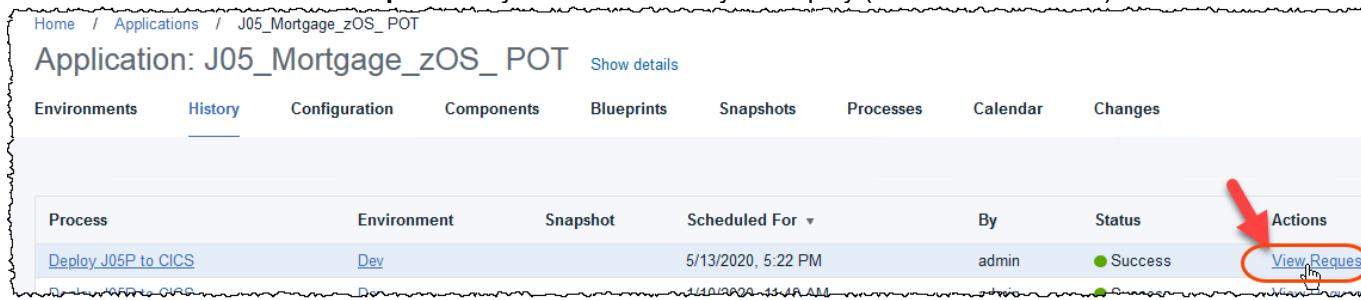
7.2.3 ► Click on **Applications** and find your application **J05_Mortgage_zOS_POT** and click on it

► Depending on your fonts sizes, you may need to use the bottom to scroll to next page..

<input type="checkbox"/>	Name	Template	Description
<input type="checkbox"/>	J04 Mortgage zOS and Worklight		J04 CICS + JKE Mobile
<input type="checkbox"/>	J04_Mortgage_zOS_POT		Deploy J04P for DevOps PoT
<input type="checkbox"/>	J05 Mortgage zOS and Worklight		J05 CICS + JKE Mobile
<input type="checkbox"/>	J05_Mortgage_zOS_POT		Deploy J05P for DevOps PoT
<input type="checkbox"/>	J06 Mortgage zOS and Worklight		J06 CICS + JKE Mobile

7.2.4 ► Click on the **History** tab

7.2.5  Click on the **View Request** entry that is related to your deploy (check the date/time)



Process	Environment	Snapshot	Scheduled For	By	Status	Actions
Deploy_J05P to CICS	Dev		5/13/2020, 5:22 PM	admin	Success	View Request
Deploy_J05P to CICS	Dev		5/13/2020, 5:22 PM	admin	Success	View Request

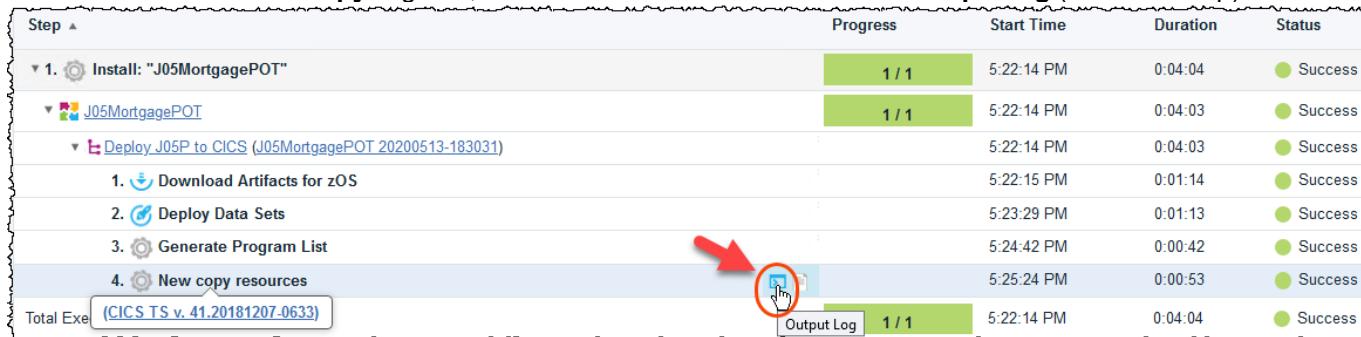
7.2.6  Click **Expand All** and expand the node as show in the #2 below

Once the step is green means that it is completed.



Step	Progress	Start Time	Duration	Status
1. Install: "J05MortgagePOT"	1 / 1	5:22:14 PM	0:04:04	Success
J05MortgagePOT	1 / 1	5:22:14 PM	0:04:03	Success
2. Deploy J05P to CICS (J05MortgagePOT 20200513-183031)		5:22:14 PM	0:04:03	Success
1. Download Artifacts for zOS		5:22:15 PM	0:01:14	Success
2. Deploy Data Sets		5:23:29 PM	0:01:13	Success
3. Generate Program List		5:24:42 PM	0:00:42	Success
4. New copy resources		5:25:24 PM	0:00:53	Success
Total Execution	1 / 1	5:22:14 PM	0:04:04	Success

7.2.7  Once the **New copy** is green, move the mouse as below and click **output log** (last UCD step)



Step	Progress	Start Time	Duration	Status
1. Install: "J05MortgagePOT"	1 / 1	5:22:14 PM	0:04:04	Success
J05MortgagePOT	1 / 1	5:22:14 PM	0:04:03	Success
Deploy J05P to CICS (J05MortgagePOT 20200513-183031)		5:22:14 PM	0:04:03	Success
1. Download Artifacts for zOS		5:22:15 PM	0:01:14	Success
2. Deploy Data Sets		5:23:29 PM	0:01:13	Success
3. Generate Program List		5:24:42 PM	0:00:42	Success
4. New copy resources		5:25:24 PM	0:00:53	Success
Total Exec (CICS TS v. 4.1.20181207-0633)	1 / 1	5:22:14 PM	0:04:04	Success

7.2.8 Notice that **J05CMORT** program was deployed to CICS Dev environment. and a CICS NEWCOPY was succeeded.

```

26 AGENT_HOME=/etc/ibm-ucd/v6.2.6/dtsc-agent
27 AH_AUTH_TOKEN=*****
28 AH_WEB_URL=https://ucdserver:18443
29 AUTH_TOKEN=*****
30 DS_AUTH_TOKEN=*****
31 DS_SYSTEM_ENCODING=IBM-1047
32 JAVA_OPTS=-Dfile.encoding=IBM-1047 -Dconsole.encoding=IBM-1047
33 PLUGIN_HOME=/etc/ibm-ucd/v6.2.6/dtsc-agent/var/plugins/com.ibm.ucd.plugin.cics_38_58e584b978e3fac6bb5320aa600d574e178448fc6
34 UD_DIALOGUE_ID=70e7cf54-db57-4993-a7f9-310c09585fc1
35 WE_ACTIVITY_ID=19982f24-80a5-4b63-a1ab-acbe4b5a1b8e
36
37 2019/05/29 16:39:38.977 GMT BUZCP0006I Connected to "10.1.1.2:1490". CICS TS version: 050300.
38 2019/05/29 16:39:43.618 GMT BUZCP0037I Perform NEWCOPY Operation.
39 2019/05/29 16:39:44.384 GMT BUZCP0024I NEWCOPY PROGRAM "J05CMORT" succeeded.
40 2019/05/29 16:39:44.517 GMT BUZCP0029I Summary: 1 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.
41

```

7.2.9 You could verify the binaries generated by the build (LOADLIB) at the UCD Code station. You can have versions of executables stored under UCD.

▶ Close the Output Log and click on Components

The screenshot shows the UrbanCode Deploy interface with the 'Components' tab highlighted (circled in red). Other tabs include 'Dashboard', 'Applications', 'Configuration', and 'Processes'. Below the tabs, the URL is 'Home / Components' and there are 'Components' and 'Templates' links.

7.2.10 ▶ Scroll down and click on J05MortgagePOT

The screenshot shows the 'Components' page with a component named 'J05MortgagePOT' selected (circled in red). It displays the version '20201202-200731', status 'Used in DevOps Pot and Jenkins Lab', and timestamp '7/17/2018, 4:13 PM'.

7.2.11 ▶ Click on Versions and the version created by your Jenkins run (it is a timestamp value)

The screenshot shows the 'Versions' page. A red circle labeled '1' highlights the 'Versions' tab. A red circle labeled '2' highlights the timestamp '20201202-200731' in the list of versions.

Version	Statuses	Type	Created By	Date	Is Importin
Filter	Statuses	Any			
20201202-200731		Incremental	admin	12/2/2020, 3:07 PM	false

7.2.12 ▶ Expand EMPTO5.DEMO.DEV.LOAD and you will see the load module created by the build on Jenkins pipeline. This is the load module that will be deployed.

The screenshot shows the 'Artifacts' page. A red circle labeled '1' highlights the 'Expand All' link. A red circle labeled '2' highlights the expanded entry for 'EMPTO5.DEMO.DEV.LOAD [PDS.ADD]'. The entry shows a file named 'J05CMORT' with type 'LOAD' and properties 'buildcommand=IEWBLINK' and 'builddoptions=MAP,RENT,COMPAT(PM5)'.

7.2.13 ▶ Minimize the browser if you will do the last optional step (#8) or close it.

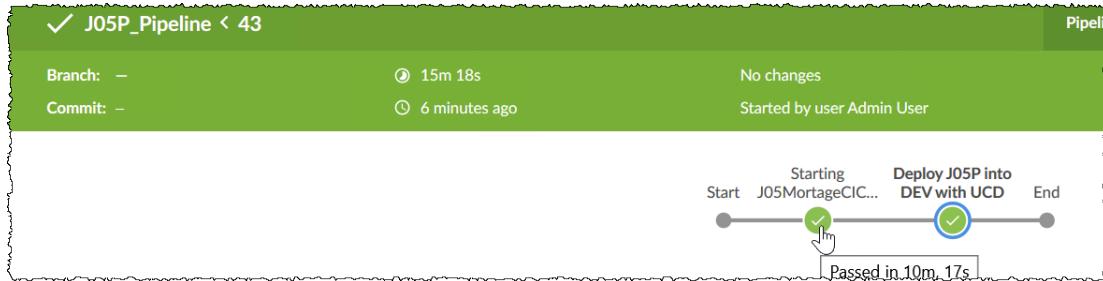
7.3 Using Jenkins Blue Ocean Plugin

This is a nice plugin. We could have used this on the Jenkins activities..

7.3.1 ▶ On left under Build History left click on your build number and select "Open Blue Ocean"

The screenshot shows the Jenkins Blue Ocean interface. A red circle labeled '1' points to the build number '#43' in the build history list. A red circle labeled '2' points to the 'Open Blue Ocean' button at the bottom of the left sidebar.

7.3.2 It shows the pipeline in another way. This plugin is handy when using “Jenkinsfile”



7.3 Testing the CICS code deployed to Dev environment

7.3.1 Double click on the **3270 terminal emulator** that is on the Windows desktop



7.3.2 Type **I cicsts53.** (I as logon) and press **Enter** key. (**Ctrl** in some keyboards).

```

Session A - [24 x 80]
File Edit Settings View Communication Actions Window Help
Print Copy Paste Send Recv Display Color Map Record Stop Play Out Support Index
z/OS V2R2 PUT1606 / RSU1607 IP Address = 10.148.150.8
VTAM Terminal = SC0TCP03
Application Developer System
      // 0000000  SSSSS
      zz // 00 00 SS
      zz // 00 00 SSSS
      zz // 00 00   SS
      zzzzzz // 0000000  SSSS
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"
1 cicsts53
24/01/11
Connected to remote server/host zos-dev using lu/pool SC0TCP03 and port 23

```

7.3.3 Logon using your z/OS user id and password (**empot05/empot05**) and press **Enter**

```

WELCOME TO CICS TS 5.3

Type your userid and password, then press ENTER:
Userid . . . empot05 Groupid . . .
Password . . . empot05_
Language . . .
New Password . . .


```

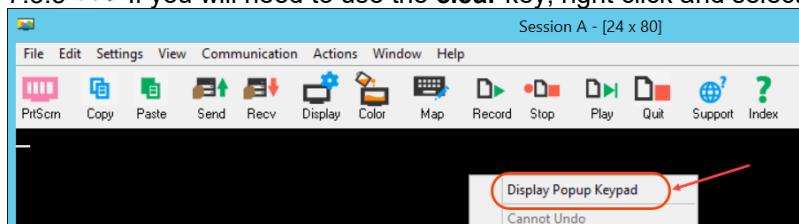
7.3.4 The sign-on message is displayed

```

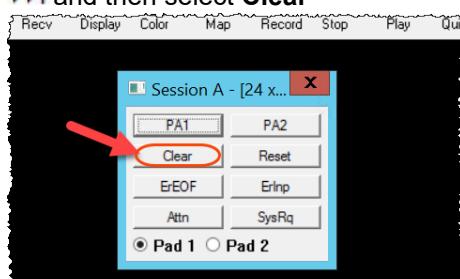
DFHCE3549 Sign-on is complete (Language ENU).
MAP a


```

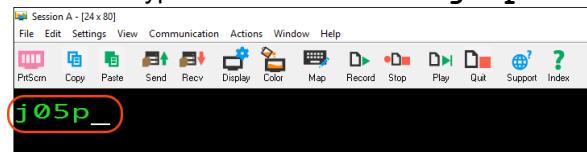
7.3.5 ► If you will need to use the **clear** key, right click and select **Display Popup Keypad**



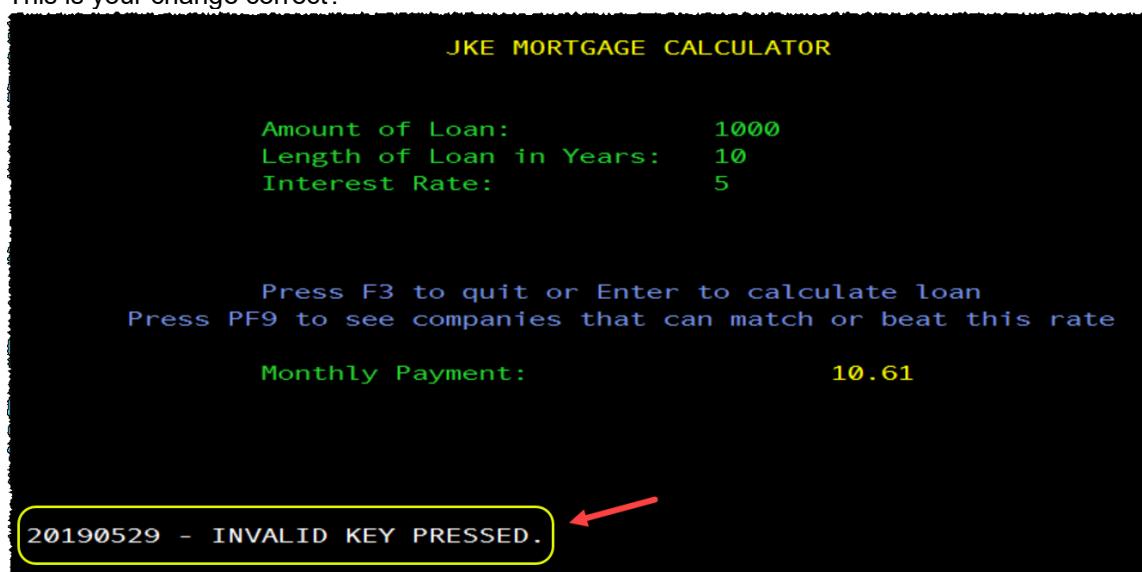
► and then select **Clear**



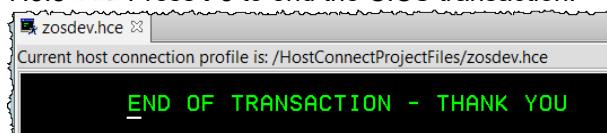
7.3.6 ► Type the CICS transaction **j05p** and press the **Enter** key.



7.3.7 ► Press **Enter** and then the **F1** key
and verify the message displayed "YYYYMMDD - INVALID KEY PRESSED".
This is your change correct?



7.3.8 ► Press **F3** to end the CICS transaction.



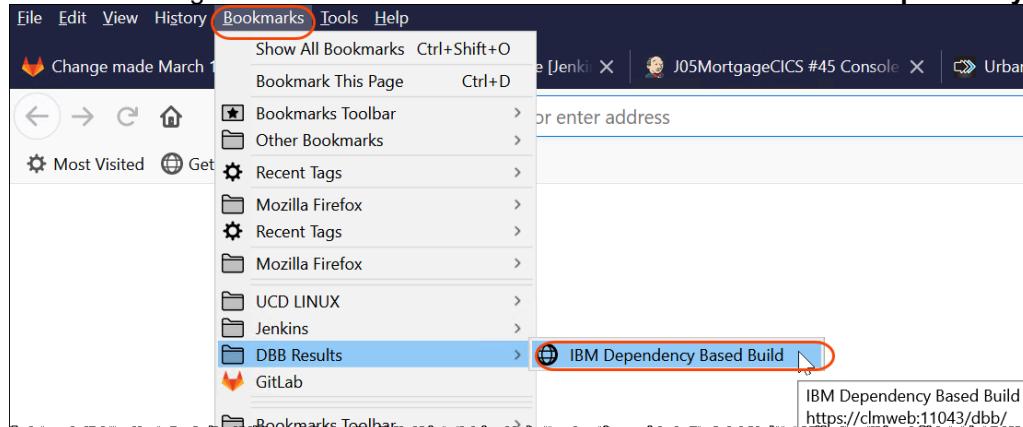
7.3.9 ► Close the terminal emulator.

Section 8.(Optional) Understanding DBB Build Reports

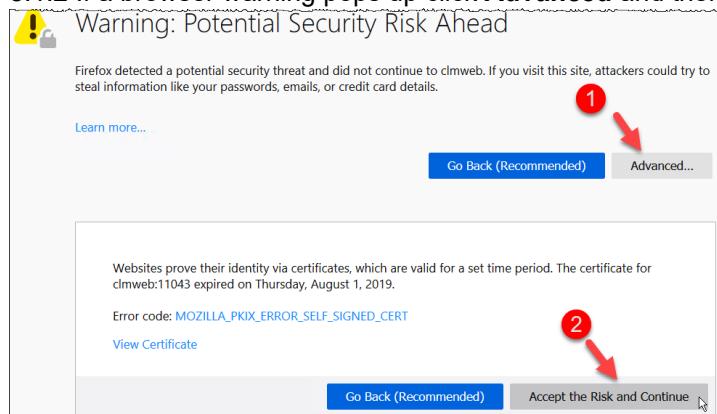
DBB has an application server (our is running on Linux) that capture the various build events. Here you will take a quick look of those reports.

8.1 Login to the DBB server using the browser

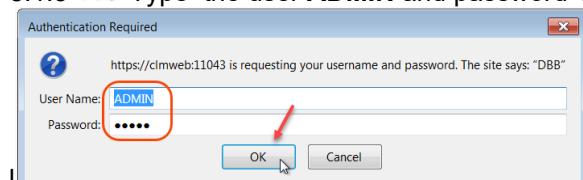
8.1.1 Using the Browser Bookmarks click on DBB Results > IBM Dependency Based Build



8.1.2 If a browser warning pops up click Advanced and then Accept the Risk and Continue.



8.1.3 Type the user ADMIN and password ADMIN (uppercase) and click OK.



DBB Collections

In order to use the build dependency information collected by the *DependencyScanner* for dependency resolution and impact analysis, all scanned source files (both programs and dependency files) will need their resulting logical files stored in the DBB repository database as part of a dependency **Collection**.

A collection is a repository container for logical files. The scope of a collection is determined by the user but generally a collection contains all the logical files of a Git branch. This way the logical files in a collection can use the scanned file's relative path from the *sourceDir* as a unique identifier in the collection.

Collections themselves can have any name but it is recommended to use the name of the Git branch of the source files being scanned

Collection names must be unique in the DBB repository database and an error will occur when trying to create a collection that already exists. A good practice is to first check if the collection with that name already exists before creating it.

8.2 Understand the DBB Collections

8.2.1 Click the Collections hyperlink and the collection name J05MortgageCICS

IBM

[Collections](#) [Build Results](#)

DBB Collections

- [J01MortgageCICS](#)
- [HealthCareApp](#)
- [J03MortgageCICS](#)
- [J02MortgageCICS](#)
- [J04MortgageCICS](#)
- [J05MortgageCICS](#)
- [J06MortgageCICS](#)
- [J07MortgageCICS](#)
- [J08MortgageCICS](#)
- [J09MortgageCICS](#)

8.2.2 This opens the DBB Collection details..

▶| Scroll down and click the link for the program J05CMORT.

Iname	file	language	link
J05MLIS	J05MortgageCICS/bms/J05MLIS.bms	ASM	link
J05MORT	J05MortgageCICS/bms/J05MORT.bms	ASM	link
J05CSMRT	J05MortgageCICS/cobol/J05CSMRT.cbl	COB	link
J05MPMT	J05MortgageCICS/cobol/J05MPMT.cbl	COB	link
J05NBRVL	J05MortgageCICS/cobol/J05NBRVL.cbl	COB	link
J05CMORT	J05MortgageCICS/cobol_cics/J05CMORT.cbl	COB	link
J05CSMRD	J05MortgageCICS/cobol_cics/J05CSMRD.cbl	COB	link

This will show the COPY book dependencies for this program.

J05CMORT	J05MortgageCICS/cobol_cics/J05CMORT.cbl		
COB			
true			
false			
false			
Iname	category	library	link
DFHAID	COPY	SYSLIB	link
J05MTCOM	COPY	SYSLIB	link
J05NBRPM	COPY	SYSLIB	link
J05MORT	COPY	SYSLIB	link

8.3 Understand the Build Results

8.3.1 ► Click on the **Build Results** hyperlink, your collection name **J05MortgageCICS** and the build that you have just run (latest)

IBM Dependency Based Build

IBM

Collections **Build Results**

DBB Build Results

- J01MortgageCICS
- HealthCareApp
- J03MortgageCICS
- J02MortgageCICS
- J04MortgageCICS
- **J05MortgageCICS**
 - [build.20180807.083740.037](#)
 - [build.20180808.101018.010](#)
 - [build.20180809.033736.037](#)
 - [build.20180809.034610.046](#)
 - [build.20180810.044700.047](#)
 - [build.20180810.063341.033](#)
 - [build.20180829.031408.014](#)
 - [build.20190529.022144.021](#)
 - **[build.20190529.043258.032](#)**
- J06MortgageCICS

8.3.2 ► This opens the *DBB Build Results*.. Click the Build Report **view** to get details..

Field	Value
id	899
group	J05MortgageCICS
label	build.20190529.043258.032
Build Report	view ← view
buildReportData	
state	2 (COMPLETE)
status	0 (CLEAN)
owner	ADMIN
permission	664
created	2019-05-29T16:32:54.026Z

8.3.3 ► This opens the *Build Report*.. Notice the load module created on this build
You also may explore other hyperlinks. Click **Show dependencies**.

Build Report

Toolkit Version:

Version:	1.0.1
Build:	83.
Date:	31-May-2018 17:36:24

Build Summary

Number of files being built: 1

	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	J05MortgageCICS/cobol_cics/J05CMORT.cbl Show Dependencies	IGYCRCTL IEWBLINK	4 0	EMPOT05.DEMO.DEV.COBOL(J05CMORT)	EMPOT05.DEMO.DEV.LOAD(J05CMORT)	LOAD	J05CMORT.log

8.3.4 ► This opens the *Build Summary Report*.. Notice the files that are required for this build (dependencies)

Build Summary

Number of files being built: 1

	File	Commands	RC	Data Sets	Outputs	Logs
1	J05MortgageCICS/cobol_cics/J05CMORT.cbl <ul style="list-style-type: none"> • J05MortgageCICS/copybook/DFHAID.cpy COPY • J05MortgageCICS/copybook/J05MTINP.cpy COPY • J05MortgageCICS/copybook/J05MTOUT.cpy COPY • J05MortgageCICS/copybook/J05MTCOM.cpy COPY • J05MortgageCICS/copybook/J05NBRPM.cpy COPY • J05MortgageCICS/copybook/J05MORT.cpy COPY Hide Dependencies	IGYCRCTL IEWBLINK	4 0	EMPOT05.DEMO.DEV.COBOL(J05CMORT)	EMPOT05.DEMO.DEV.LOAD(J05CMORT)	J05CMORT.log

8.3.5 ► Close the browser.

Congratulations! You have completed the Lab 7. .

LAB 8 – (OPTIONAL) Using Application Performance Analyzer (APA) (60 minutes)

Modified June 01 2021 by Regi,, (Reviewed by Wilbert Kho)

This lab will take you through the steps of using [Application Performance Analyzer \(APA\)](#) integrated with [Application Delivery Foundation for z \(ADFz\)](#).

On this lab you will measure an existing COBOL/DB2 program running in batch.

Overview of development tasks

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

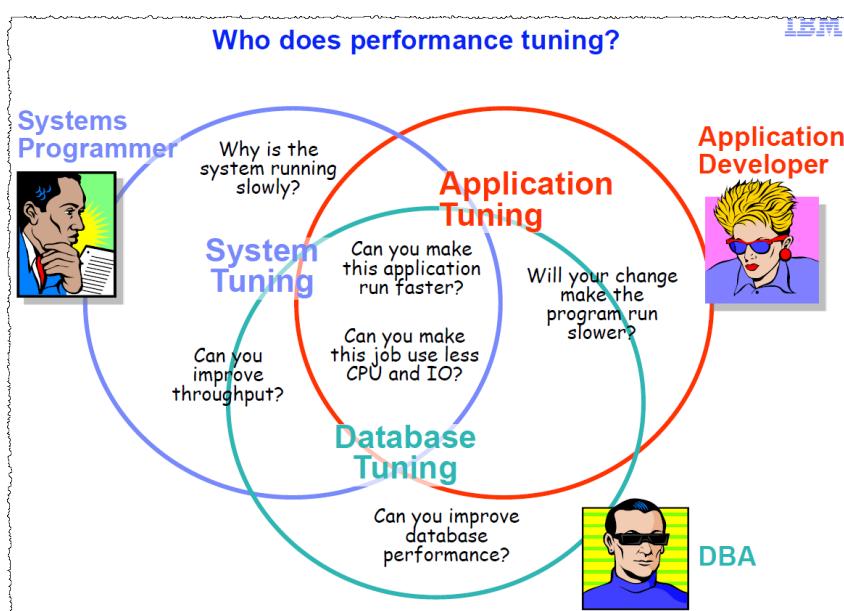
1. **Create a new observation request for a job that is not running yet**
→ Using ADFz you will create an APA observation request.
2. **Run a sample batch job to collect performance data**
→ You will submit a JCL that will execute a COBOL/DB2 batch program and will collect performance data.
3. **Review some of the reports created.**
→ You will analyze some of the reports created.

What is APA (Application Performance Analyzer) ?

IBM® Application Performance Analyzer for z/OS® measures and reports how applications use available resources. This tool helps you identify system constraints and improve application performance. The product's key functions allow users to maximize the performance of your existing applications and improve response time of online transactions and batch turnaround times.



The tool aids application design, development and maintenance cycles. It helps evaluate applications in the design phase, measure the impact of increased data volume or changes in business requirements, and generate historical reports to analyze performance trends.

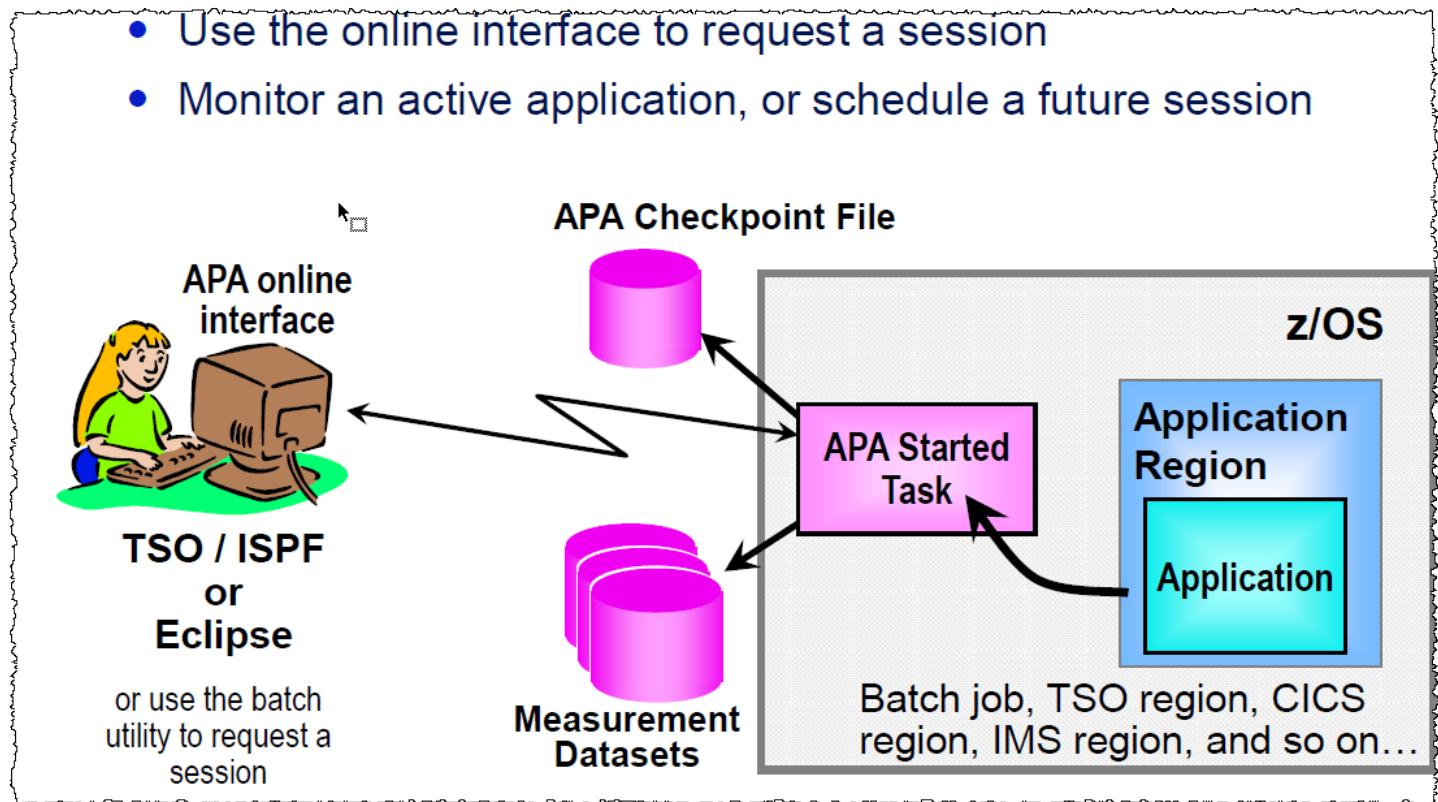


Section 1. Create a new observation request for a job that is not running yet

For this lab you will use **ibmuser** user id

You will create an **Application Analyzer Observation** for an existing program to be executed via JCL submission.

- Use the online interface to request a session
- Monitor an active application, or schedule a future session



1.1 Connect to the ADFz Common Components

There is a “Common components server” on the host system for Problem Determination Tools for z/OS. You must connect to it to access Application Performance Analyzer.

1.1.1 Start IBM Developer for z Systems version 15

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

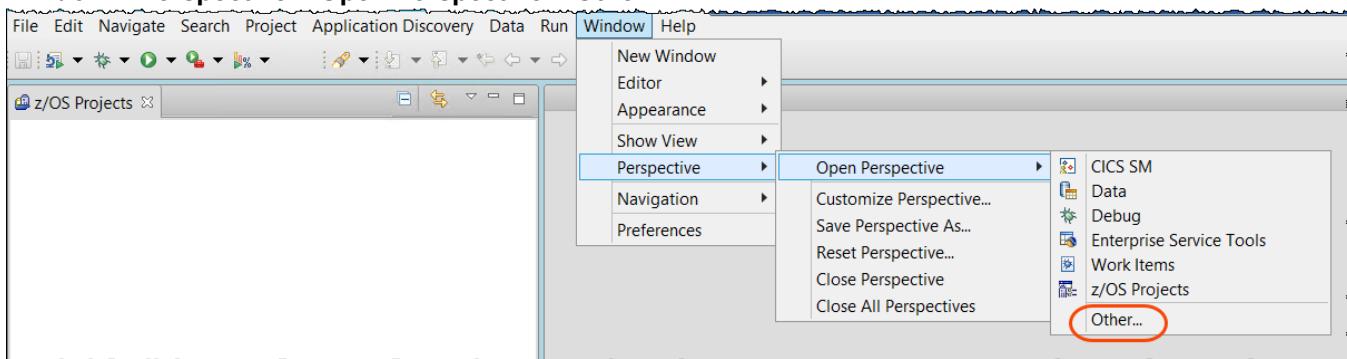
►► Verify that the message indicates that it is Version 15.0.1

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.

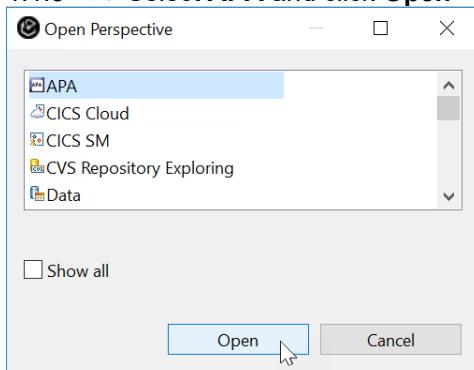


1.1.3 On other labs you may be used a different userid . Your new userid now will be **ibmuser**.

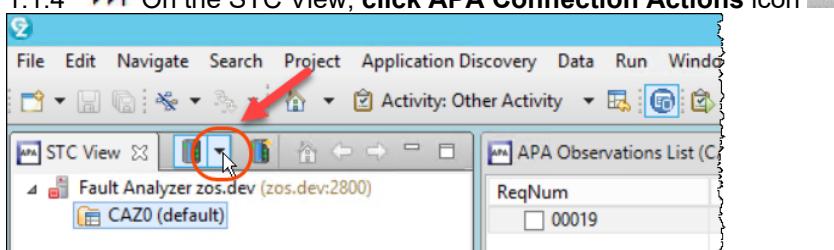
1.1.2 ►| Open the **APA** perspective by selecting
Window > Perspective > Open Perspective > Other..



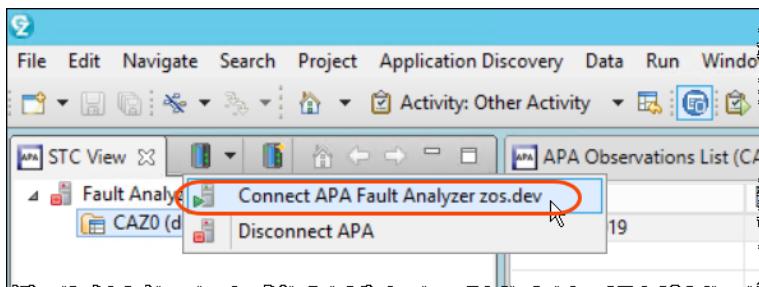
1.1.3 ►| Select **APA** and click **Open**



1.1.4 ►| On the STC View, click **APA Connection Actions** icon



1.1.5 ►| Then select **Connect APA Fault Analyzer zos.dev**



1.1.6 ►| If required use **IBMUSER** and password **SYS1** to sign in and click **OK**



You will see some few pop-ups and this view will lose the focus once the connection succeeded.

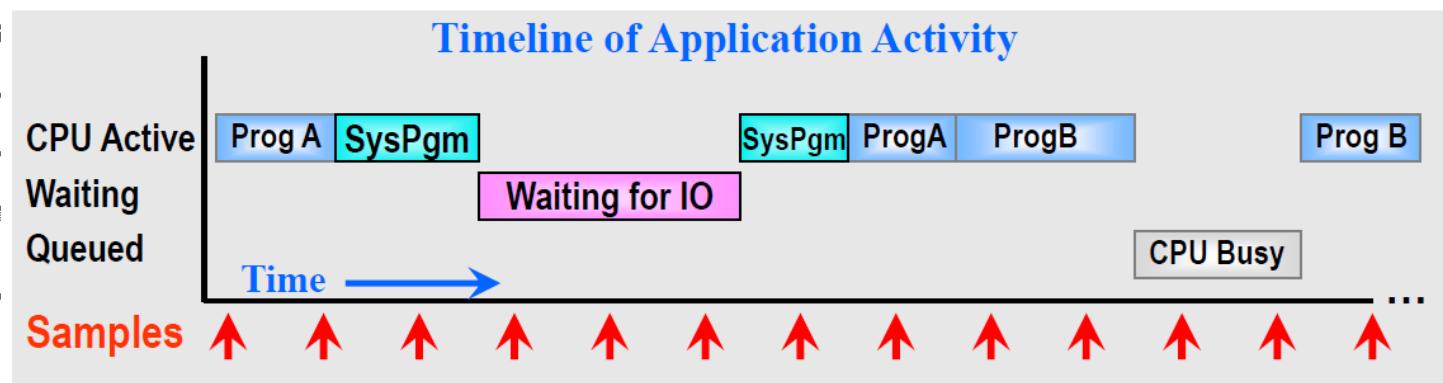
1.2 Begin a new APA observation session

You will now prepare APA to collect information from the job that you will submit latter.

You will collect 5000 samples in a duration of 1 minute (or less)

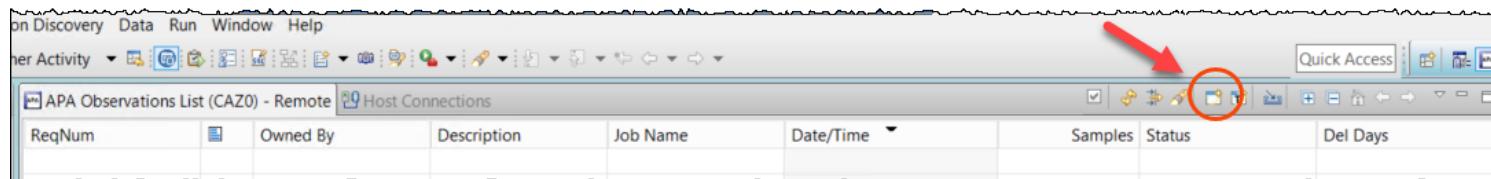
■ APA uses "Sampling"

- APA collects information at fixed intervals
- During each "sample", APA determines what the application is doing and why



- Collected data are stored in a measurement data set
- Samples are aggregated later, when you view APA reports

1.2.1 ► Click on the **New Observation** icon .

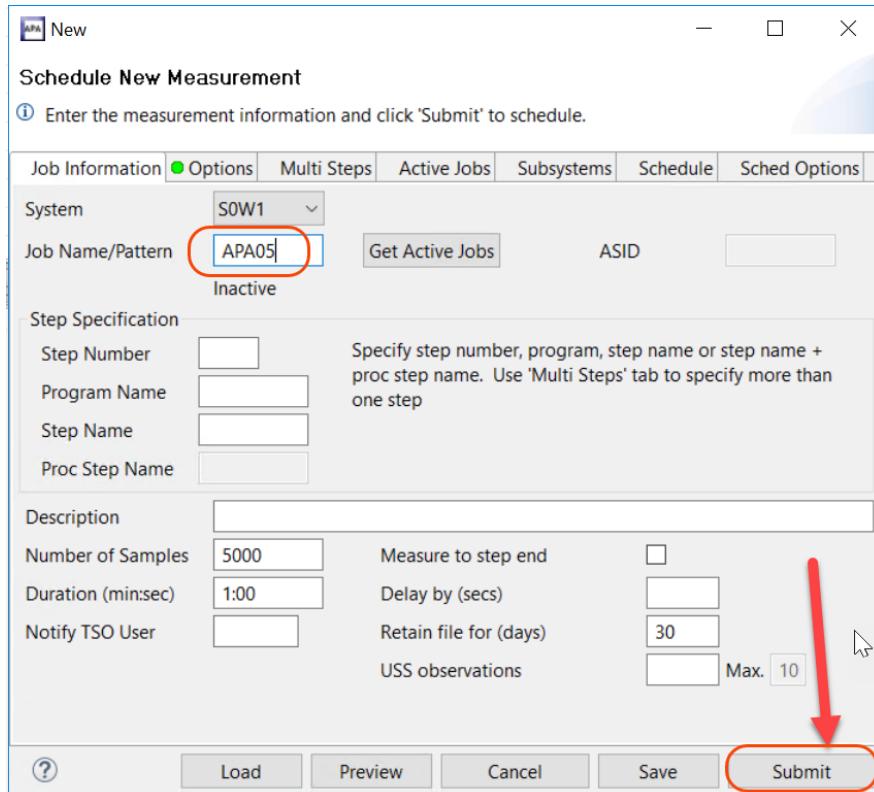


This will open the Schedule New Measurement dialog.

1.2.2 ► On *Job Name/Pattern* type **APA05**.

You will keep the number of samples as 5000 and duration 1 minute (defaults)

► Click **Submit**.



1.2.3 Your new observation request has been added and APA is now waiting for your job to begin (*Sched*)

APA Observations List (CAZ0) - Remote									
Reqnum	Owning Host	Owned By	Description	Job Name	Date/Time	Samples	Status	Del Days	System
00020		EMPOT01		APA05	Apr-28-2021 12:47	5,000	Sched	30	S0W1

What have you done so far?



You created an APA observation request for a job that you will submit. This observation will collect up to 5000 samples in a duration of 1 minute.

Section 2 – Run a sample batch job to collect performance data

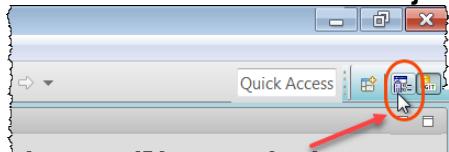
You will submit a COBOL/DB2 batch job named **APA05** and let APA collect performance data.

2.1 Connect to z/OS

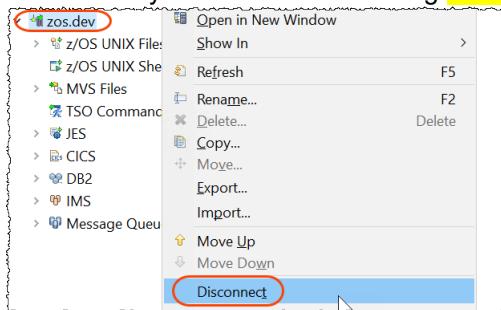
To submit a job, you first need to connect to z/OS using your assigned user id.

If you are already connected to z/OS using userid **ibmuser**, skip this and go to step 2.2.

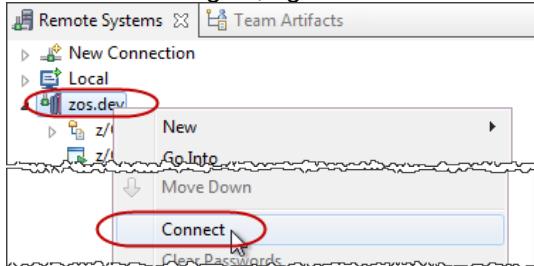
- 2.1.1 ► Switch to the **z/OS Projects** perspective clicking on icon  on the top right corner



- 2.1.2 ► If you are connected using other userid than **ibmuser** you must disconnect first.



- To connect again, right click on **zos.dev** and select **Connect**

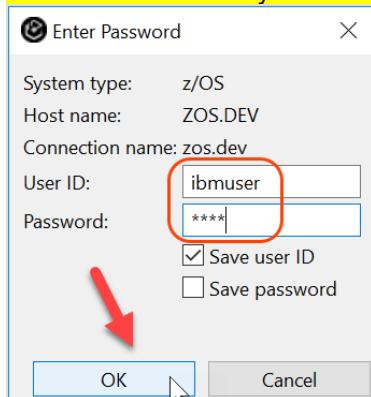


- 2.1.3 ► Type **ibmuser** as userid and **sys1** as password.

The userid and password can be any case; don't worry about having it in UPPER case.

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

Notice → On this lab your userid will be **ibmuser (not **empot01** or **empot05** used in other labs)**

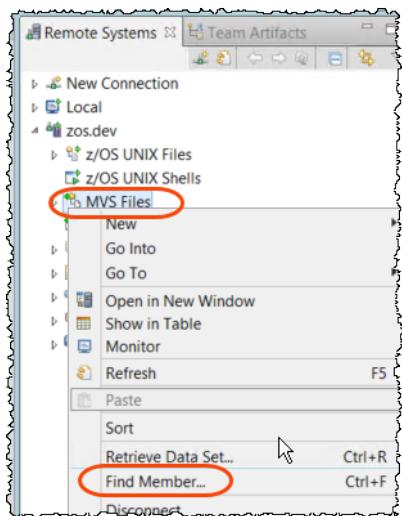


2.2 Find the JCL to be submitted

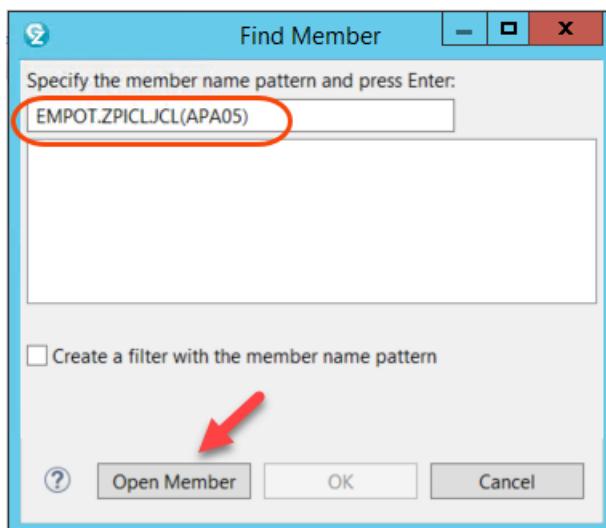
You will find a JCL that runs a COBOL/DB2 batch program.

There are multiple ways to find a job on z/OS. If you don't know in which PDS is your job you may use this technique below.

- 2.2.1 ► Right click on **MVS Files** and select **Find Member....**



- 2.2.2 ► Type **EMPOT.ZPICL.JCL(APA05)** and click **Open Member**.



2.3 Submit the JCL to execute the COBOL/DB2 batch program

The member **APA05.jcl** will open and you must submit for execute on z/OS.

A screenshot of the Eclipse IDE interface. On the left, the code editor window shows the JCL member **APA05.jcl**. The code content is as follows:

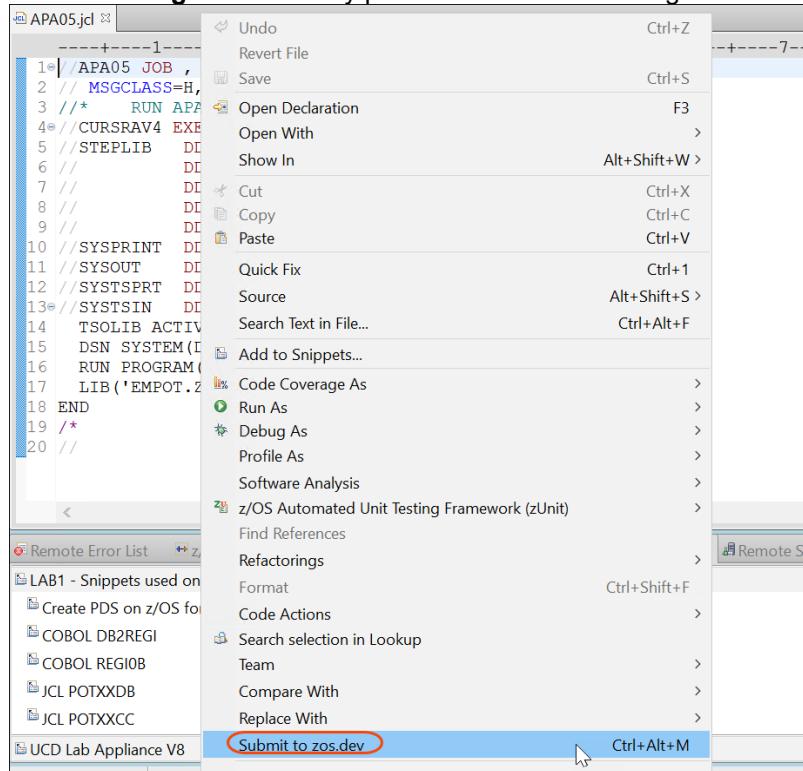
```

1 //> APA05 JOB
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 /* RUN APA05 that is compiled with NO OPTIMIZER
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=IBMUSER.POT.LOAD,DISP=SHR
6 // DD DSN=EMPT.ZPICL.LOAD,DISP=SHR
7 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR
8 // DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
9 // DD DISP=SHR,DSN=FEK910.SFEKAUTH
10 //SYSPRINT DD SYSOUT=*
11 //SYSOUT DD SYSOUT=*
12 //SYSTSPRT DD SYSOUT=*
13 //SYTSIN DD *
14 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
15 DSN SYSTEM(DBDBG)
16 RUN PROGRAM(APA05) PLAN(APA05) -
17 LIB('EMPT.ZPICL.LOAD')
18 END
19 /*
20 //

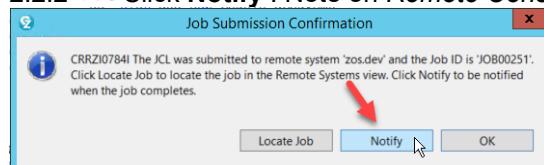
```

On the right, the **Remote Systems** view shows a tree structure of JCL members under the node **EMPT.ZPICL.JCL**, including **APA05.jcl** and many other members like **APA06.jcl**, **APA07.jcl**, etc.

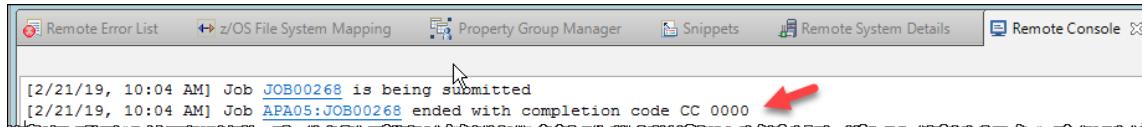
2.2.1 ➡ Right click in any place of the member being edited and select **Submit to zos.dev**



2.2.2 ➡ Click **Notify**. Note on *Remote Console* view that the job is being submitted



2.2.3 ► On the bottom you will see the *Remote Console* view. The completion code must be 000. If this is not the case call the instructor.



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

What have you done so far?



You created an APA observation request for a job that you have submitted.
The observation collected less than 5000 samples since it run in less than 1 minute.

What information does Application Performance Analyzer provide?

- APA reports detailed information about:

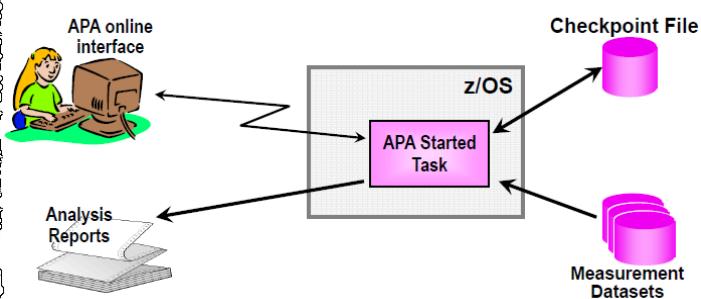
<u>Category:</u>	<u>Sub-categories (partial list) :</u>
CPU usage	by Module, by Statement, by Timeline, ...
WAIT time	by Module, Statement Attribution, ...
DB2	various Metrics by SQL stmt, Plan, Thread, ...
CICS	various Metrics by Txn and by Task, ...
IMS calls	by CPU Time, Call Service Time
DASD usage	by Device, by File, by Timeline, ...
HFS usage	by Device, by File, by Timeline, ...
MQ	by Queue, by Request, by Transaction ...
Storage	by Device, by Task, by Timeline, ...
And more	

Section 3 – Review some of the reports created.

APA produces analysis reports. On this section you will explore some of the reports produced. Notice that you could print, create a PDF or export in XML format.

APA produces analysis reports

- View and print analysis reports
 - View sample file data from the APA online interface
 - Or print performance reports to SYSOUT or to a PDF file
 - Sample file data can also be written in XML format



3.1 Download the most recent APA observation reports

The reports can be seen using the APA perspective.

- 3.1.1 On top right corner click the icon to switch to the APA perspective



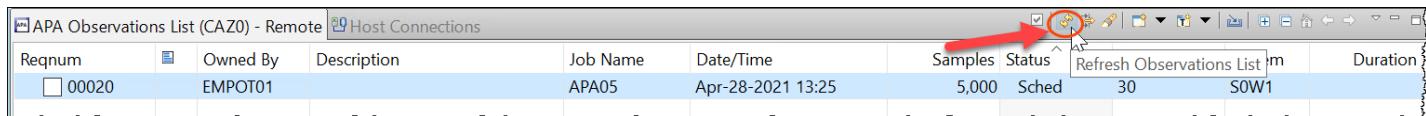
Observation Reports List Overview

The reports list is a 2-level tree-view. The first level (parent) rows represent the report category while the second-level (child) rows are for the individual reports.

The category rows are for informational/organizational purposes only and result in no action if clicked (other than if the category is expanded). A sticky note icon is displayed on each report row which has a sticky note.

For each report row that is selected (checkbox checked), a menu is available with a list of actions. Reference the Menu section of the Observations Reports List for details. If a report row is double-clicked a new Report View is opened. .

- 3.1.2 Click on the icon to refresh the APA Observation List view



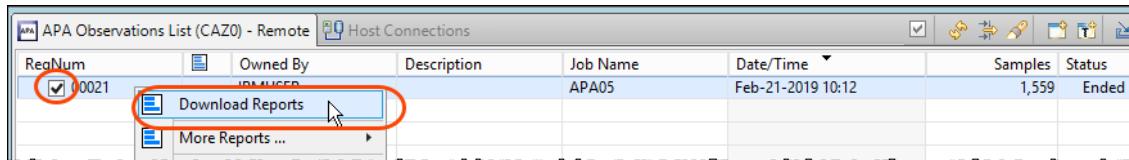
Notice that the Status change to **Ended** and also the *Details* view depicts information about the APA measurement. The name of the sample file ('APA.IBMUSER.xxx') shown below, circled in red, is displayed, which was automatically created by APA to hold the results of the observation session. This file is tied to the observation request. **On your lab probably the owner will be EMPOT01 instead of empot05 shown on the screen capture..**

Reqnum	Owned By	Description	Job Name	Date/Time	Samples	Status	Del Days	System
00018	EMPOT05	APA05	Mar-16-2021 11:55	1,438	Ended	30	S0W1	

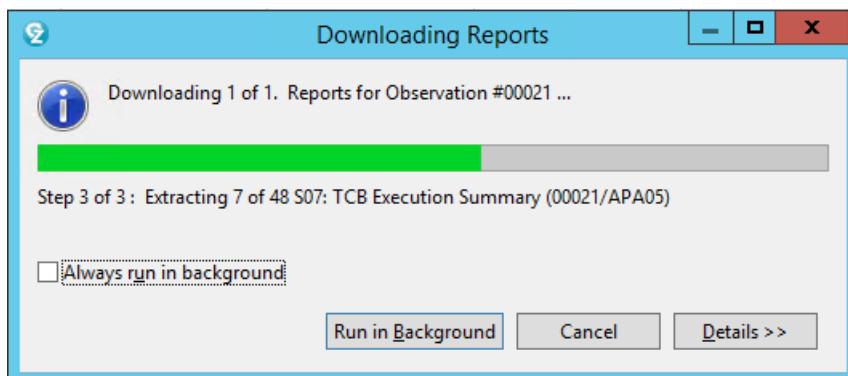
General		Measurement Criteria	
Request Number	00018	Select by Job Name	APA05
Request Description	No Description Entered	Select by Sys Name	S0W1
Request Status	Ended	Sample Interval	12,000 microseconds
Owner Id	EMPOT05	Duration	60 seconds
Time of Request	Tuesday Mar 16 2021 11:51:33.98	Measurement Information	
Session Start Time	Tuesday Mar 16 2021 11:55:24.23	Sample File DSN	APA.EMPOT05.R00018.APA05.SF
Session End Time	Tuesday Mar 16 2021 11:55:41.72	Samples Requested	5,000
Session Duration	0 minutes, 17.49 seconds	Samples Done	1,438
Session Delete Date	Thursday Apr 15 2021 11:55:47.89	ASID	001D
		Job ID	JOB00265

Notice that around 1,500 samples were collected instead of the 5,000 requested.
The reason is that the job duration was less than one minute requested (your value may be different).

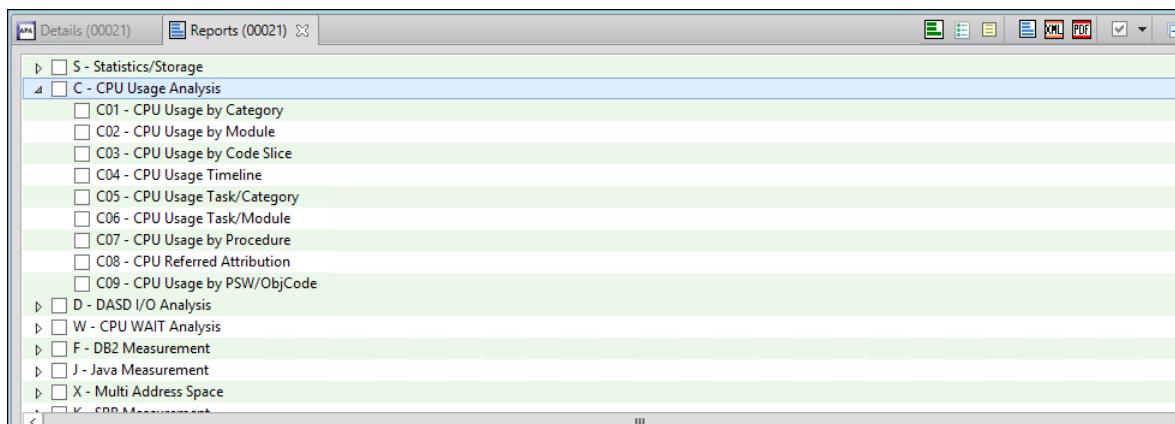
3.1.3 ➔ Click on the box (on left) to select the observation, **Right click** on observation and select **Download Reports**. This will download the created reports to your workspace.



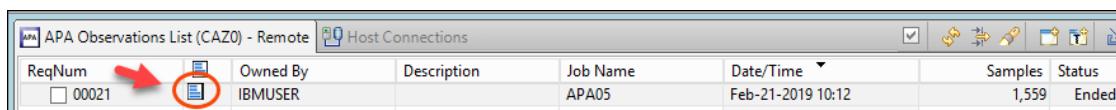
This dialog below may or may not be displayed. Notice that this operation may take a while, depending on your system, network, etc..



When completed you will have the *Reports* view populated.



Also notice that a sticky note is created on *APA Observation List* view.
At any point you can analyze any of the downloaded reports.



3.2 Analyzing the APA Measurement Profile report (S01)

The reports can be seen using the APA perspective. Each report category can be expanded. Let's start with the Statistics/Storage reports.

COBOL/DB2 application used on this example

The COBOL/DB2 program source code used here is on the dataset **EMPOT.ZPICL.COBOL(APA05)**.

You may have access if interested on see the source code
This program is compiled with **NOOPTIMIZE** option. It executes OPEN/FETCH/CLOSE on a small DB2 table 300 times:

```

000-LOOP-OPEN-FECH-CLOSE.
  MOVE 1 to IND.
  PERFORM 000-MAINLINE-RTN THRU 350-EXIT
          UNTIL IND = 300.

...
000-MAINLINE-RTN.

...
150-OPEN-CURSOR-RTN.
  EXEC SQL OPEN C1
  END-EXEC.
150-EXIT.
  EXIT.
250-FETCH-A-ROW.
  EXEC SQL FETCH C1 INTO
    :DEPT-TBL:DEPT-NULL,
    .....
  END-EXEC.
250-EXIT.
  EXIT.
300-CLOSE-CURSOR-RTN.
  EXEC SQL CLOSE C1 END-EXEC.
300-EXIT.
  EXIT.

350-EXIT.
  EXIT.

```

Also this program execute a COMPUTE statement 900,000 times as seen below:

```

502-LOOP.
  MOVE 1 to IND.
  PERFORM 504-ADD-COUNTER THRU 506-EXIT
          UNTIL IND = 900000.
504-ADD-COUNTER.
  MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE.
  COMPUTE WS-INTEGER-START-DATE =
    FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
  COMPUTE IND = IND + 1.
506-EXIT.
  EXIT.
...

```

3.2.1 ➡ Scroll up and expand the S reports above by clicking on the icon on the left of **S - Statistics/Storage**



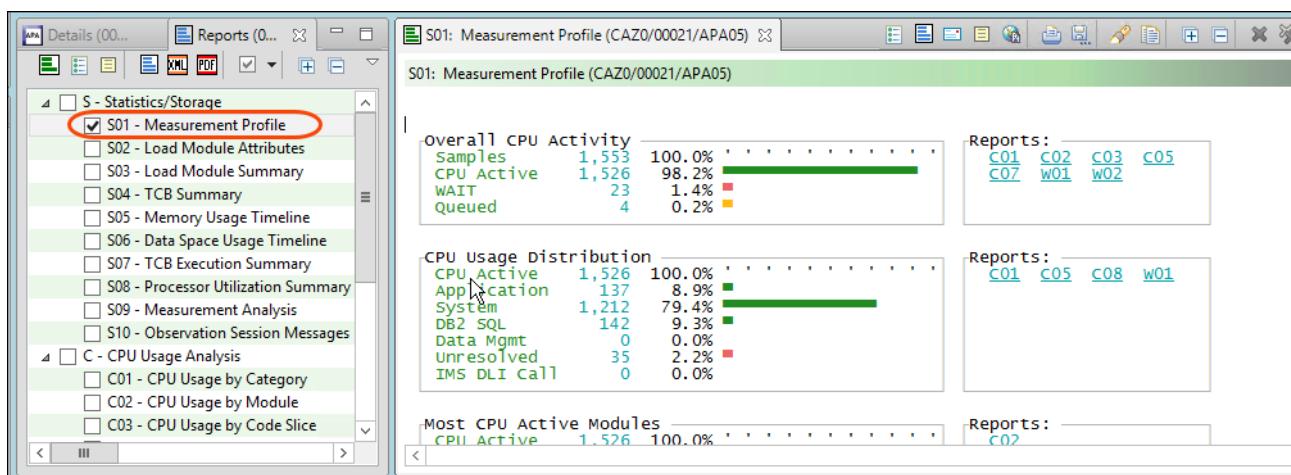
3.2.2 ► Double click on the S01 Measurement profile report. The *Measurement Profile report* is displayed.

This is a good report to examine first when analyzing a measurement. It provides an at-a-glance summary of various aspects of the measurement data and helps you choose which other reports to concentrate on. The first section of this report consists of a series of mini performance graphs illustrating various types of activity that was measured. This is followed by a section that reports measurement values.

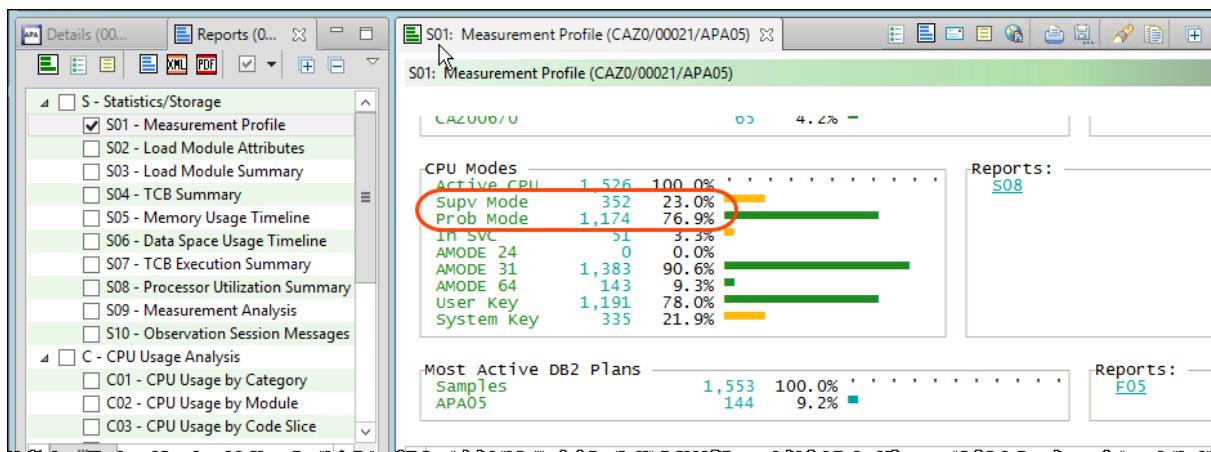
Remember that the actual statistics shown on your reports may be entirely different than the examples shown here. In the example below, the CPU was active for about 98.2% of the samples. An additional 1.4% of the samples show the application in a WAIT state, waiting for resources to become available.

Notice the box down. The CPU Usage Distribution uses the number from the CPU Active Samples to further define what occurred during the Observation Session.

In this case 8.9% was attributed to the Application (APA05 program) and 9.3 % was attributed to DB2 subsystem. While 79.4% was attributed to the Systems programs.



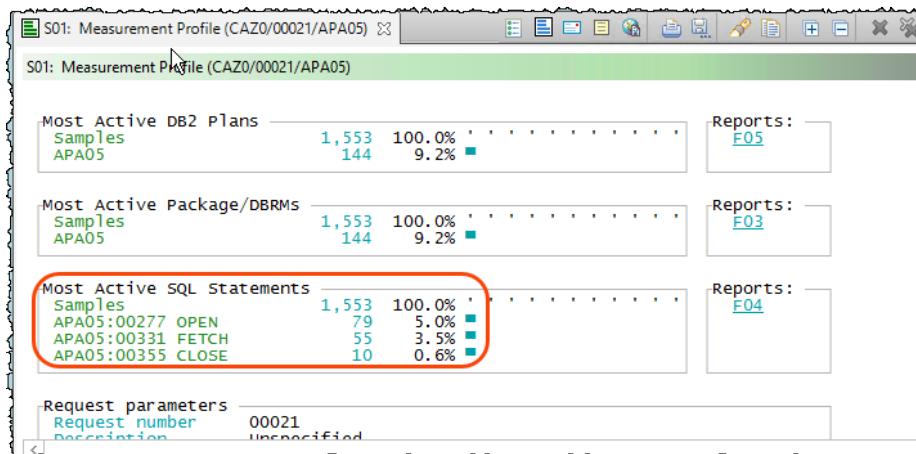
3.2.3 ► Scroll down to the remainder of the S01 report.



Notice on this example that 76.9% of the samples observed occurred in **Problem Mode**, indicating that the application was a bit dependent upon code running in **Supervisor Mode**. Your value probably will be different.

It also shows that this application ran mostly in AMODE 31(31 bit addressing mode), but 9.3% run in 64 bit addressing mode.

3.2.3  Scroll down to the remainder of the S01 report to see the CPU usage on the DB2 statements.



Notice that 5.0% of the DB2 CPU usage is on the SQL OPEN statement.

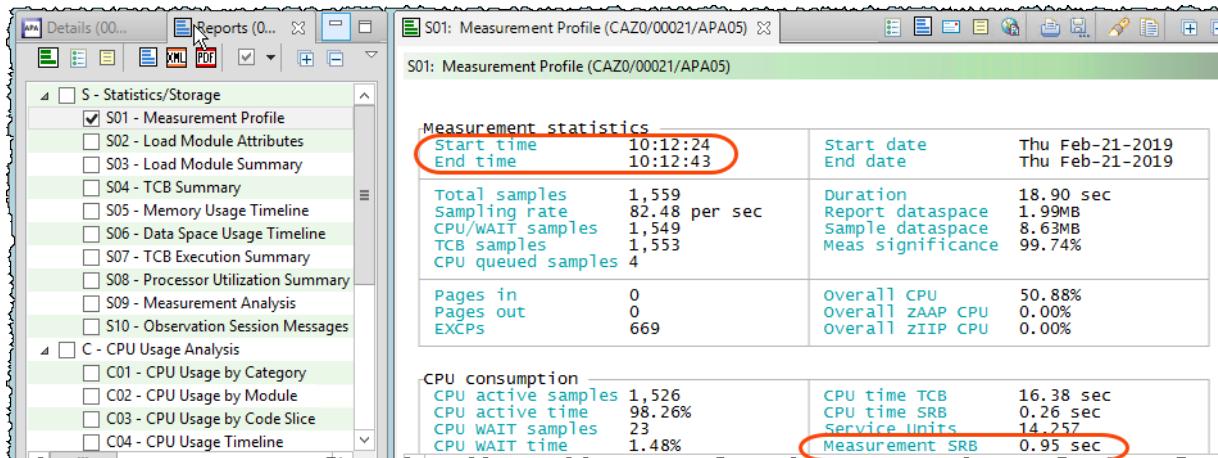
Usage of zIIP and zAAP processors



Did You Know? Usage of zIIP and zAAP processors used in many DB2 and IMS environments are shown by APA. These processors are included in the Number of CPUs count. The S08 Processor Utilization Summary provides a further breakdown of the CPU usage by the application. Specialty CPUs are reported on a separate line...

3.2.4  Scroll down to the bottom of the S01 report.

During monitoring, APA typically does not attach anything to the application in order to collect performance data. The run time of a monitored application should not be affected by APA. It does require CPU time in its own address space, and this is reported as the **Measurement SRB value** (0.95 sec).



3.3 Analyzing the APA Load Module Summary report (S03)

This report can be useful to find which compile options were used for the module creation.

3.3.1 Double click on the S03 report. The Load Module Summary report is displayed.

Module	Locn	Address	Count	size(bytes)	Attributes	DDName	Load Library
APA05	JPA	20458000	1	16,384		SYS00002	EMPOT.ZPICL.LOAD
CAZ00091	CSA	1D769458	1	31,656	RU RN	STEPLIB	APA.SCAZAUTH
CAZ00202	CSA	1D763C88	1	4,984	RU RN		
CAZ00650	CSA	1D73E490	1	2,688	RU RN		
CAZ00670	CSA	1D63BEF0	1	4,368			
CAZ00681	CSA	1D7261A0	1	15,968	RU RN	STEPLIB	APA.SCAZAUTH
CAZ00978	CSA	1D721FA8	1	4,184	RU RN	STEPLIB	APA.SCAZAUTH
CEEINIT	JPA	00072E10	1	45,552	RU RN	-LNKLST-	CEE.SCEERUN
CEEPLPKA	PLPA	05D48000	1	2,182,416			
COEMRETR	JPA	204221D8	1	7,720	RU RN		CEE.SCEELPA
CFLFCN	NHC	A1214110	1	1,688			

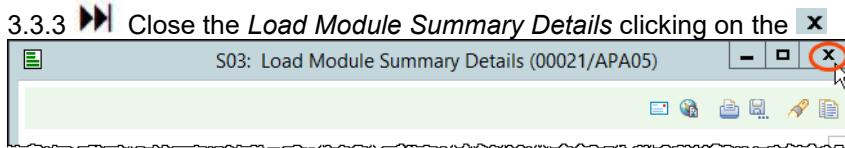
From here you can find that this program was loaded from the dataset EMPOT.ZPICL.LOAD.

3.3.2 Right click on APA05 and select Details

Module	Locn	Address	Count	size(bytes)	Attributes	DDName	Load Library
APA05	JPA	20458000	1	16,384		SYS00002	EMPOT.ZPICL.LOAD
CAZ00091	CSA	1D769458	1	31,656	RU RN	STEPLIB	APA.SCAZAUTH
CAZ00202	CSA	1D763C88	1	4,984	RU RN		
CAZ00650	CSA	1D73E490	1	2,688	RU RN		
CAZ00670	CSA	1D63BEF0	1	4,368			

You may find interesting information as seen below, like which COBOL version the program was compiled, the compile options (like NOOPT – NO Optimizer), etc.....

Module Information for APA05				
Load Address	20458000	to	2045BFFF	
Module Size	16,384			
Attributes	NOREUS, NORENT			
Module Location	JPA			
Loadlib DDNAME	SYS00002			
Load Library	EMPOT.ZPICL.LOAD			
ESD Information for APA05				
External	Offset	Length	Start Addr	End Addr
APA05	000000	8528	20458000	2045A14F
Entry Points: Compiled by COBOL V6.1 at 2019/01/09 14:21:11				
+00000000 APA05 +00000000 APA05				
Compile options:				
ADV AFP(VOLATILE) QUOTE ARITH(COMPAT) NOAWO NOCICS NOCMPR2 NOCURR DATA(31) DBCS NODECK DISPSIGN(COMPAT) NODLL DP NODUMP NODYNAM NOEXP NOFASTSRT HGPR(PRESERVE) INTDATE(ANSI) LIB LIST NOMAP NOMDECK NONAME NONUM NUMCLS(PRIM) NUMPROC(NOPFD) OBJ NOOFFSET NOOPT NOOPTFILE PGMNAME(CO) RENT RMODE(ANY) SEQ SIZE SOURCE SQL SQLCCSID NOSSR NOSTGOPT NOTERM TEST(SOURCE) NOTHREAD TRUNC(STD) NOVBREF NOWORD XP(XMLSS) XREF ZWB				

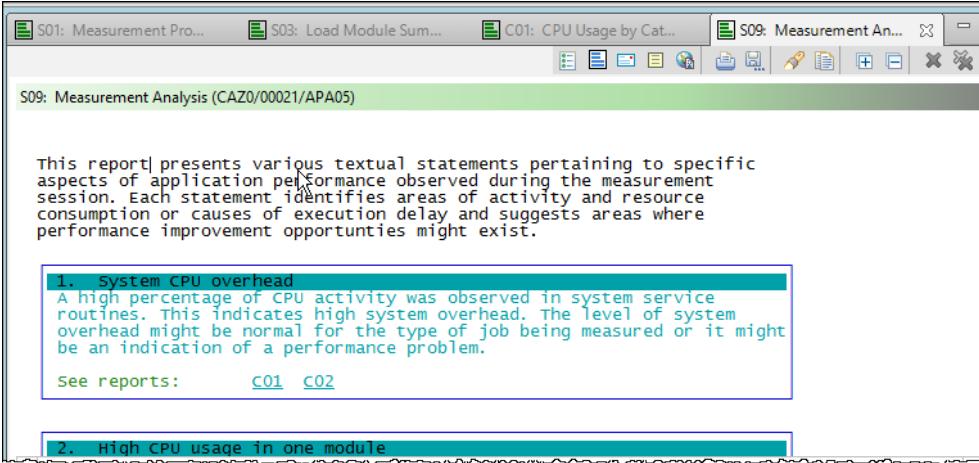


3.4 Measurement Analysis report (S09)

This report can be useful to see a summary of the results.

3.4.1 ► Double click on the **S09** report. The *Measurement Analysis* report is displayed.

This report shows aspects of application performance observed during the measurement session.

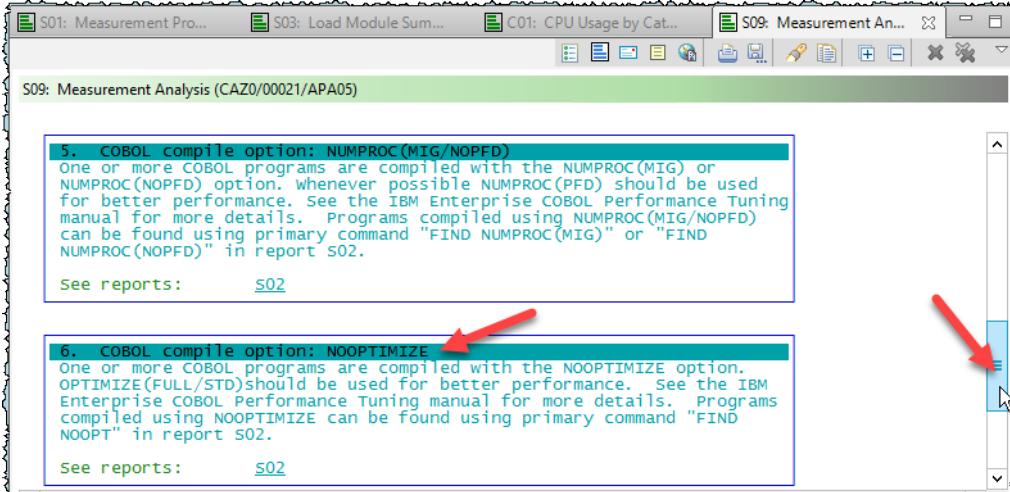


This report presents various textual statements pertaining to specific aspects of application performance observed during the measurement session. Each statement identifies areas of activity and resource consumption or causes of execution delay and suggests areas where performance improvement opportunities might exist.

1. System CPU overhead
A high percentage of CPU activity was observed in system service routines. This indicates high system overhead. The level of system overhead might be normal for the type of job being measured or it might be an indication of a performance problem.
See reports: [C01](#) [C02](#)

2. High CPU usage in one module

3.4.2 ► Scroll down and verify the observations. One interesting one is the one that shows that the COBOL program was compiled with NO OPTIMIZER that could affect the performance.



5. COBOL compile option: NUMPROC(MIG/NOPFD)
One or more COBOL programs are compiled with the NUMPROC(MIG) or NUMPROC(NOPFD) option. Whenever possible NUMPROC(PFD) should be used for better performance. See the IBM Enterprise COBOL Performance Tuning manual for more details. Programs compiled using NUMPROC(MIG/NOPFD) can be found using primary command "FIND NUMPROC(MIG)" or "FIND NUMPROC(NOPFD)" in report S02.
See reports: [S02](#)

6. COBOL compile option: NOOPTIMIZE
One or more COBOL programs are compiled with the NOOPTIMIZE option. OPTIMIZE(FULL/STD) should be used for better performance. See the IBM Enterprise COBOL Performance Tuning manual for more details. Programs compiled using NOOPTIMIZE can be found using primary command "FIND NOOPT" in report S02.
See reports: [S02](#)

TIP: Interested in execute this program again but compiled with OPTIMIZE(2)?

3.4.3 ► (OPTIONAL) The JCL stored in **EMPOT.ZPICL.JCL(APA05OPT)** executes the SAME COBOL program, but compiled with **OPTIMIZE(2)** . Once you finish this lab you may create another observation named **APA05OPT** for this job and analyze the results...

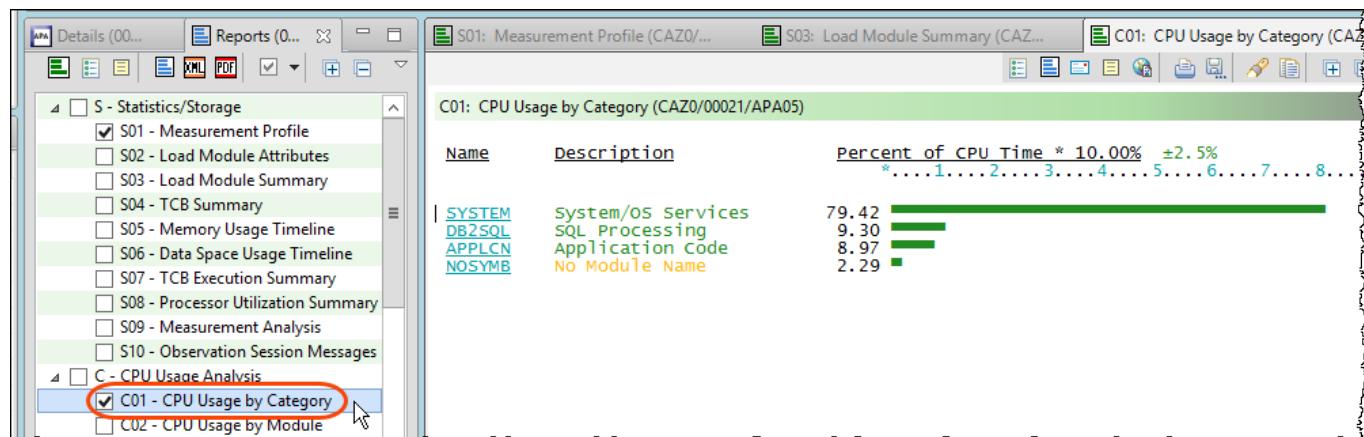
The job used to compile with **OPTIMIZE(2)** is at **EMPOT.ZPICL.JCL(APACL050)** .

3.5 Analyzing the APA CPU Usage by Category report (C01)

This report can be useful to find which compile options were used for the module creation.

3.5.1 ► Double click on the **C01** report. The *CPU Usage by Category* report is displayed.

The CPU Usage by Category shows the major collections of CPU activity with drill down capabilities to the Load Modules, CSECTS, and Program Source statements in each category.

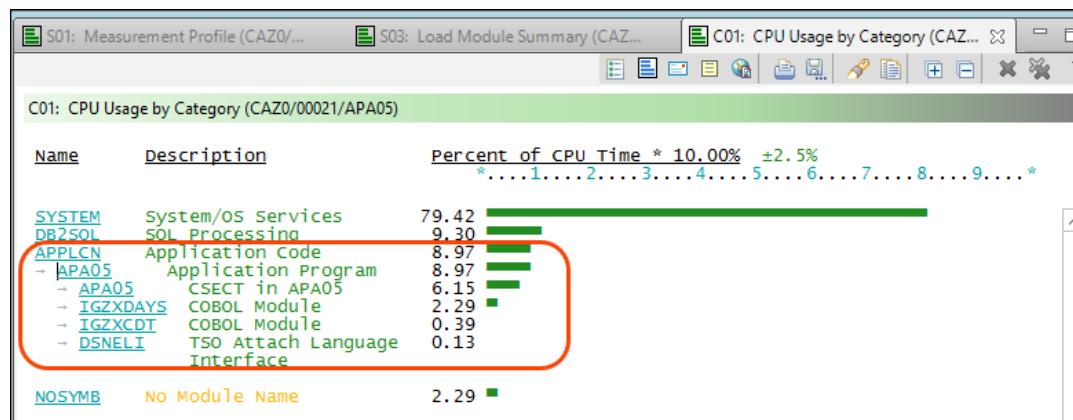


3.5.2 ► Click on the **APPLCN** collection to see the names of the Load Modules which appear in the sampling

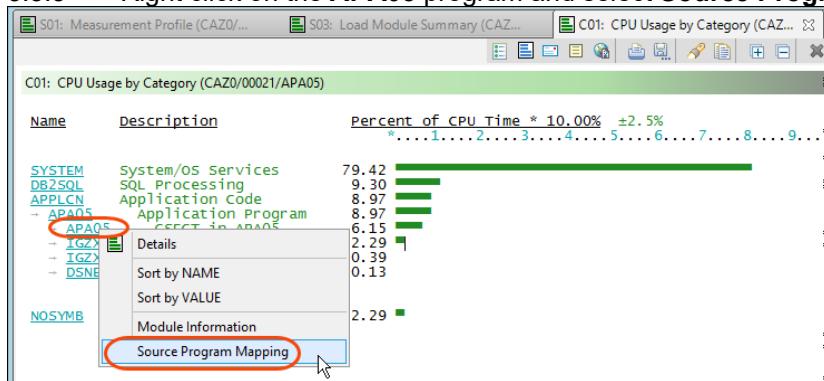
The only Load Module in the **APPLCN** collection is the **APA05** Load Module. In this example **APA05** consumed all of the CPU resource attributed to the **APPLCN** collection.

Since Load Modules can contain multiple programs or CSECTS which are link-edited together, expand the **APA05** program to reveal the individual programs that were observed during the Observation Session.

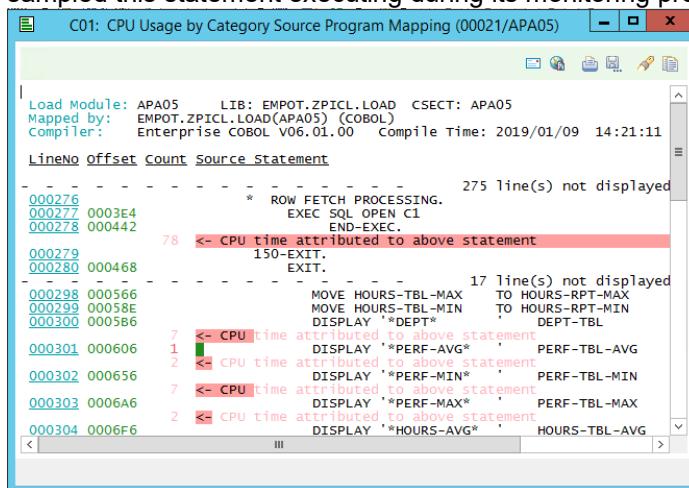
► Click on the **APA05** program to expand the information



3.5.3 ► Right click on the APA05 program and select Source Program Mapping

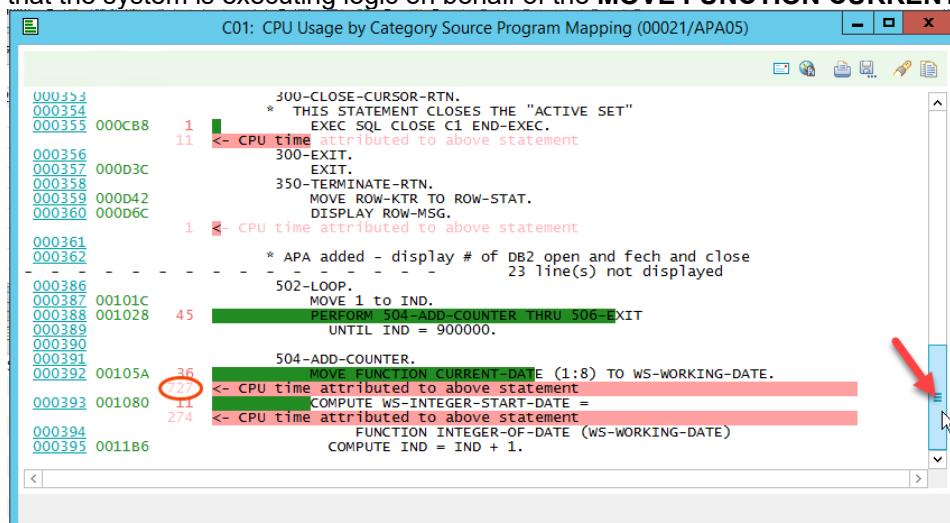


3.5.4 Some of the program source statements are displayed. The **Count** column indicates how many times APA sampled this statement executing during its monitoring process.



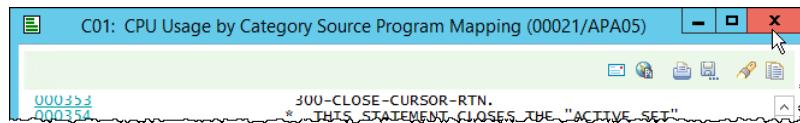
3.5.5 ► Scroll up to the end

APA also indicates when system level activity is taking place on behalf of a source statement. In this example, the Count column carries a high number and the statement under the **MOVE FUNCTION** statement (727) indicates that the system is executing logic on behalf of the **MOVE FUNCTION CURRENT-DATE** statement.



Tip: A Possible performance improving here would define **WS-WORKING DATE** as packed or binary numeric?

3.5.6 Close the panel clicking on the

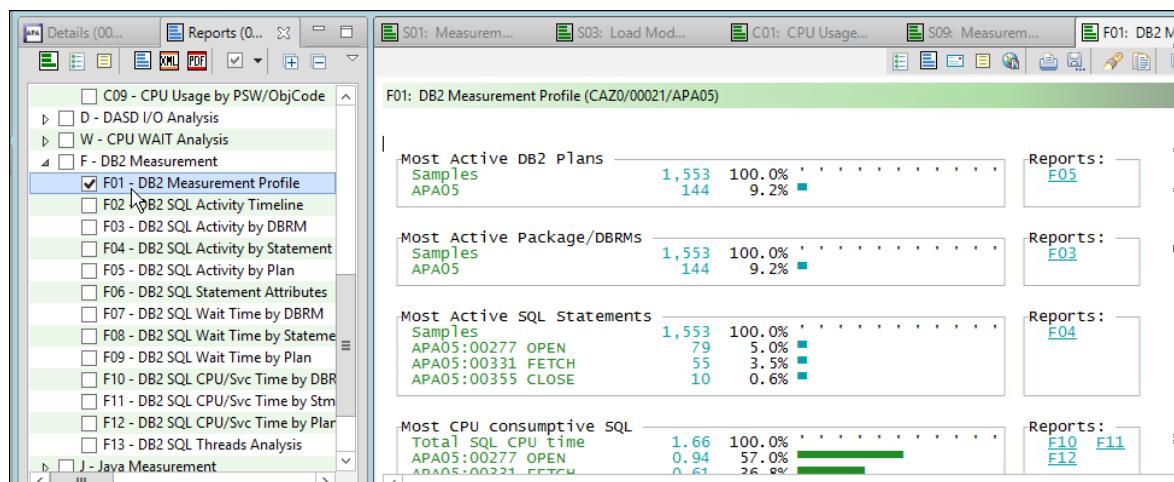


3.6 Analyzing the APA DB2 Measurement Profile report (F01)

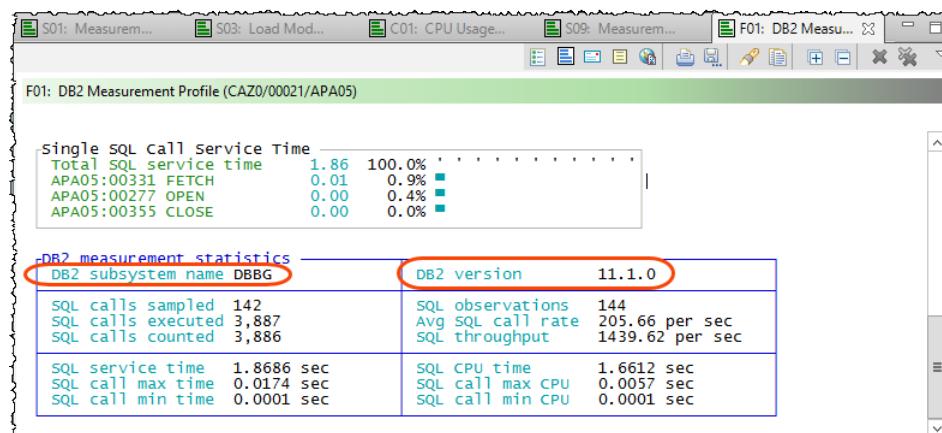
This is a good report to examine first when analyzing DB2 information. It provides an at-a-glance summary of various aspects of the measurement data and helps you choose which other reports to concentrate on. The first section of this report consists of a series of mini performance graphs illustrating various types of activity that was measured. This is followed by a section that reports measurement values.

3.6.1 Scroll down the Reports view and double click on the F01 report. The DB2 Measurement Profile report is displayed.

This report shows aspects of application performance observed during the measurement session



3.6.2 Scroll down the DB2 Measurement Profile Reports view and verify which DB2 level we used, the DB2 subsystem, etc.



3.7 Analyzing the APA DB2 Activity by Statement report (F04)

Use this report to see how time was consumed by SQL request processing. The percentage of time is reported by each SQL request.

3.7.1 ► Double click on the F04 report. The *DB2 SQL Activity by Statement* report is displayed. Notice that the Open statement used most of the resources on this example.

Seqno	Program	Stmt#	SQL Function	Percent of Time
S00003	APA05	277	OPEN	5.08
S00001	APA05	331	FETCH	3.54
S00002	APA05	355	CLOSE	0.64

3.7.2 ► Click S00003. You will see the details of this OPEN statement in the program **APA05**.

```

Seqno Program Stmt# SQL Function Percent of Time * 10.00% ±2.5%
*....1....2....3....4....5....6....7....8....9.

S00003 APA05 277 OPEN 5.08
> 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 GROUP BY DEPT

S00001 APA05 331 FETCH 3.54
S00002 APA05 355 CLOSE 0.64

```

3.7.3 ► Right click S0003 and select **Source Program Mapping**

3.7.4 The reports shows where the **Open** is used in the program.

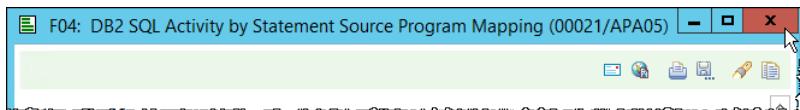
```

Load Module: APA05      LTB: EMPOT.ZPICL.LOAD  CSECT: APA05
Mapped by: EMPOT.ZPICL.LOAD(APA05) (COBOL)
Compiler: Enterprise COBOL V06.01.00  Compile Time: 2019/01/09 14:21:11

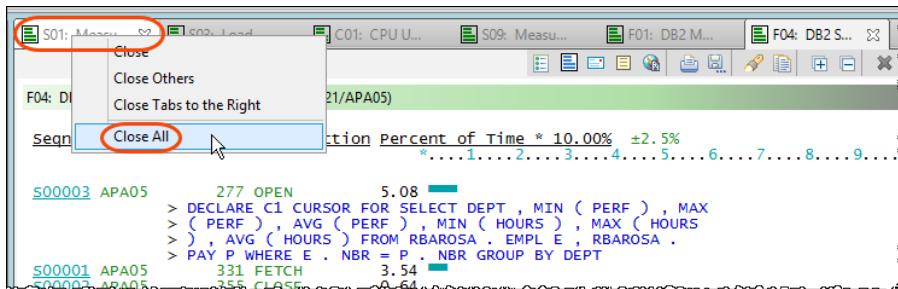
LineNo Offset Count Source Statement
----- -----
000268          267 line(s) not displayed
000269          WHERE E.NBR = P.NBR
000270          *
000271          AND PERF > :PERF
000272          GROUP BY DEPT
000273          END-EXEC.
000274          100-EXIT.
000275          EXIT.
000276          150-OPEN-CURSOR-RTN.
000277          * THIS STATEMENT OPENS THE "ACTIVE SET" IN PREPARATION OF
000278          * ROW FETCH PROCESSING.
000279          EXEC SQL OPEN C1
000280          END-EXEC.
000281          150-EXIT.
000282          EXIT.
000283          200-FETCH-RTN.
000284          * THIS PARAGRAPH SETS UP THE SQL PARAMETERS, PERFORMS THE
000285          * PARAGRAPH TO FETCH THE ROW, AND DISPLAYS THE RESULTS.
000286          * ----> HINT <--- USE ISPF EXCLUDE (XX) OR BLOCK COPY
000287          * FROM YOUR CURSOR DECLARE STATEMENT TO VERIFY PROPER
000288

```

3.7.5 ► Close the panel clicking on the **x**



3.7.6 ► Right click on the **S01** Report view and select **Close All** to close all opened reports.



APA have more reports that you can view.. We just touched few of them.

Congratulations! You have completed the Lab 8.

LAB 9 – (OPTIONAL) Using IBM Z VTP to test a COBOL /CICS / DB2 transaction (60 minutes)

Updated August 16 2021 by Regi (reviewed by Wilbert Kho) –

This lab will take you through the steps of using the automated testing capabilities of the [IBM Z Virtual Test Platform](#) (VTP) to create a test at the transaction levels of a COBOL/CICS/ DB2 application.

IBM Z Virtual Test Platform transforms testing on IBM z/OS by facilitating shift left transaction-level testing. It enables you to record online transactions and batch jobs, and with the recorded data execute the test without the need for original middleware or data. This enables mainframe shops to test their applications early in the development process without impacting existing environments and applications.

This is done by stubbing out CICS, IMS, DB2, DL/1 and MQ calls, enabling the program to be tested without a DB2 and CICS environment being active.

In this lab you will record the transaction using an existing COBOL/CICS/DB2 application and write the recorded data into a flat file (playback file). Then you will make a change to the COBOL/CICS/DB2 program adding a bug and rerun the test that should catch the bug.

Overview of development tasks

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

PART #1 – Record existing COBOL/CICS/DB2 transactions and introduce a bug

1. **Use VTP to record the CICS/DB2 application in action**
→ You will record the CICS application in action. This application starts multiple CICS transactions using COBOL programs with and without DB2 access.
2. **Run a JCL to execute the recorded replay sequence**
→ You will submit a JCL to execute the sequence of interactions that you recorded and verify the results.
3. **Modify one COBOL/DB2 program (introduce a bug) and rerun the VTP JCL**
→ You will modify the COBOL/DB2 program to introduce a bug, rerun the replay JCL, and observe the failure due the introduced bug.

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

4. **Run the CICS transaction and verify the bug**
→ You will run CICS application and observe the bug introduced.
5. **Use IDz to fix the bug, recompile the COBOL/DB2 program**
→ The bug is fixed using IDz, program fixed is rebuilt.
6. **Rerun the VTP JCL and verify that the bug is eliminated**
→ When running the JCL recorded replay you can verify that the program is fixed.

PART #1 – Record existing COBOL/CICS/DB2 transactions and introduce a bug

Section 1. Use VTP to record the CICS/DB2 application in action

You will start a 3270 emulation and execute a transaction named **HCAZ** to record some of the interactions using VTP.

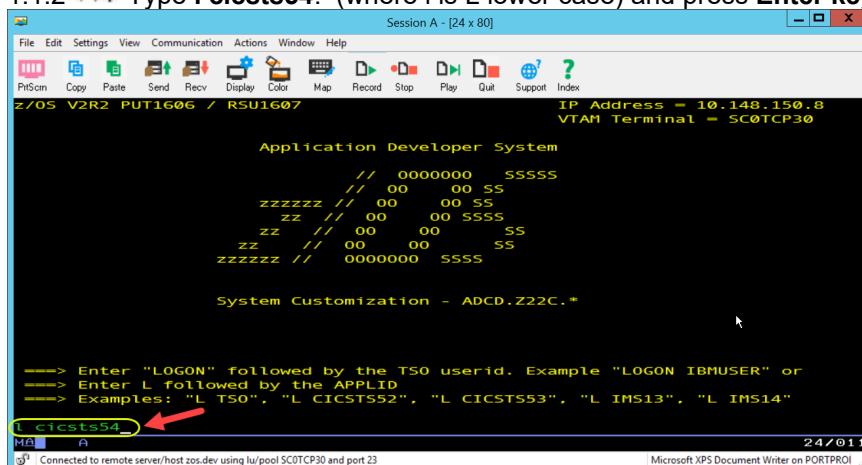
1.1 Emulate a 3270 terminal and Connect to CICS version 5.4

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



This opens the host emulator.

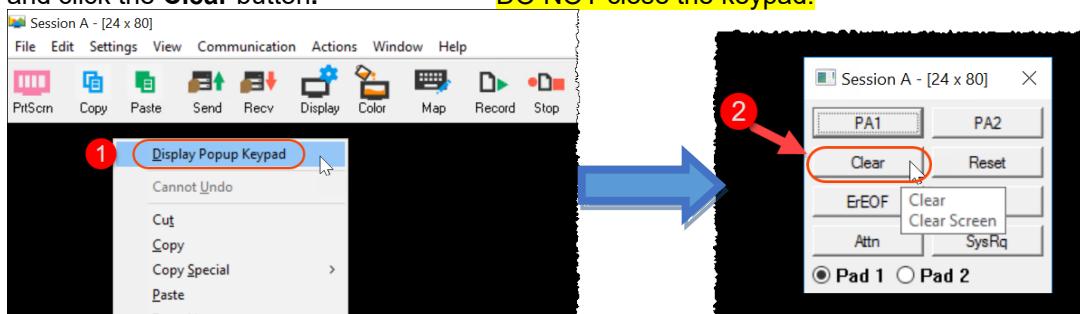
- 1.1.2 Type **I cicsts54.** (where I is L lower case) and press **Enter key**.



- 1.1.3 Sign on using **ibmuser** as the userid and **sys1** as the password.

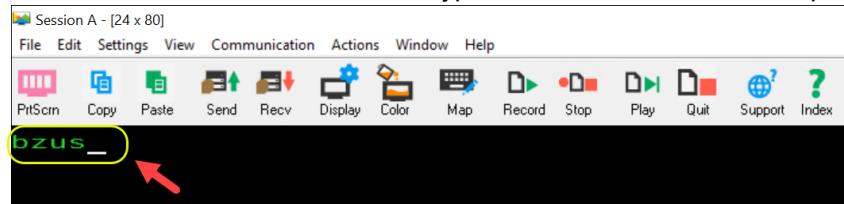


- 1.1.4 To clear the screen, right click on the dark space and select **Display Popup Keypad** and click the **Clear** button. **DO NOT close the keypad.**

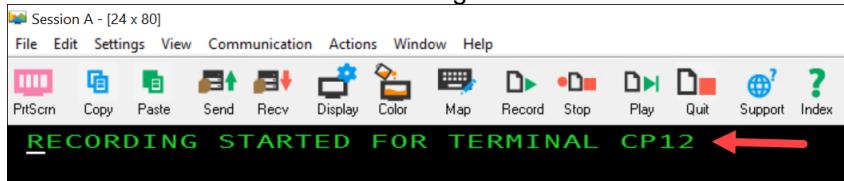


1.2 Using VTP to record the Health Care Application transaction sequence

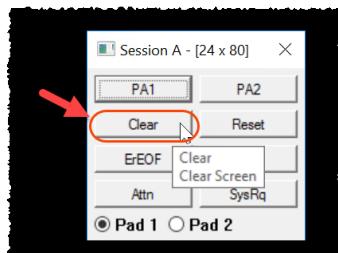
1.2.1 Click on the black area to type the transaction **bzus** and press **Enter key**



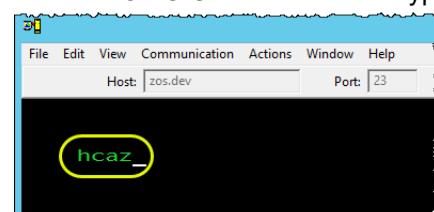
1.2.2 This invokes the Z VTP recording in the CICS terminal..



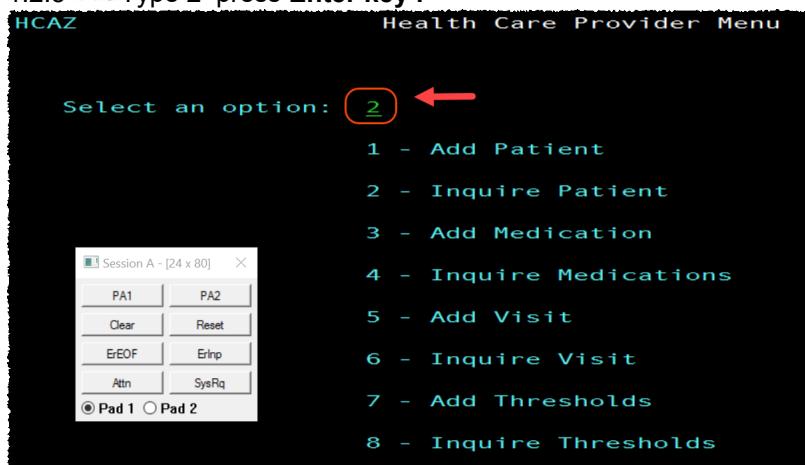
1.2.3 Clear the screen again clicking the **Clear** button.



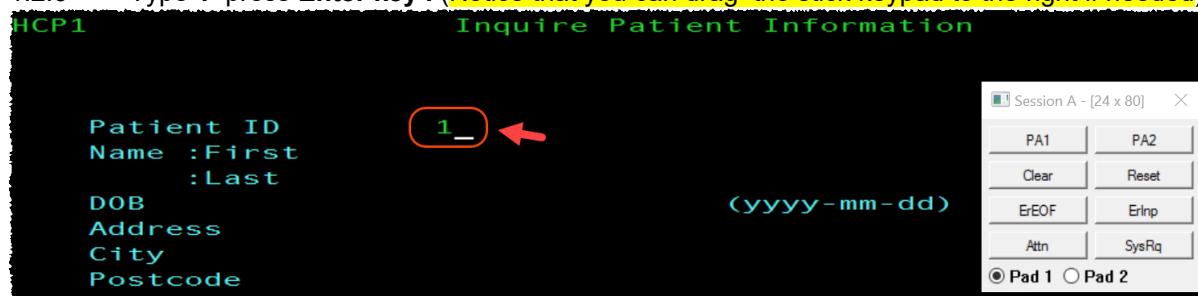
1.2.4 On CICS main terminal type **hcaz** and press **Enter key** to invoke the CICS transaction application



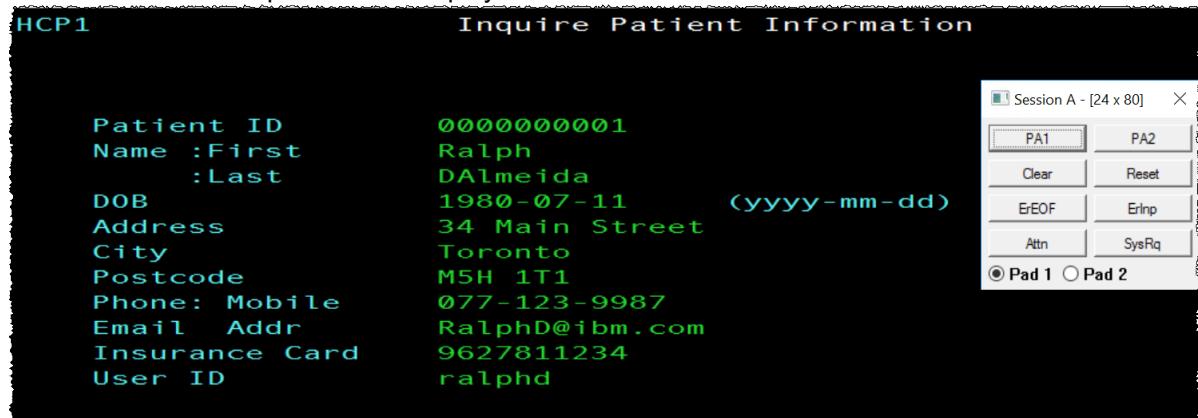
1.2.5 Type 2 press **Enter key**.



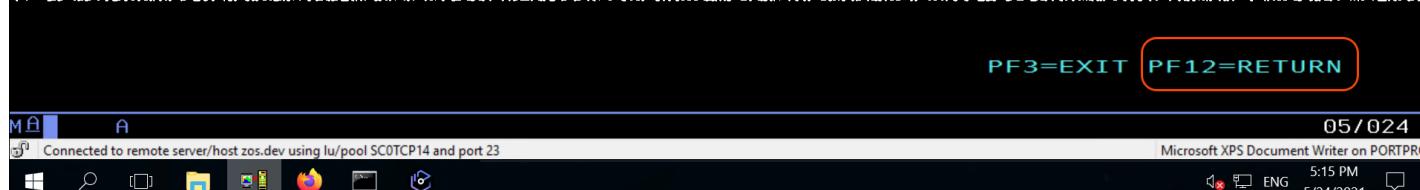
1.2.6 ► Type 1 press **Enter key**. (Notice that you can drag the stick keypad to the right if needed)



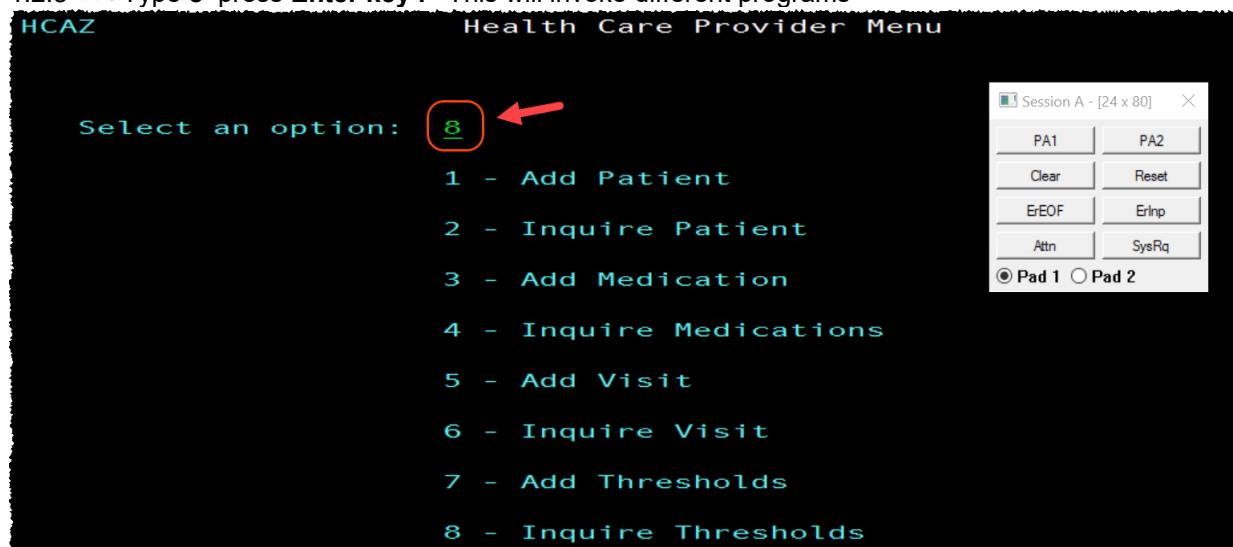
You should have the patient data displayed.



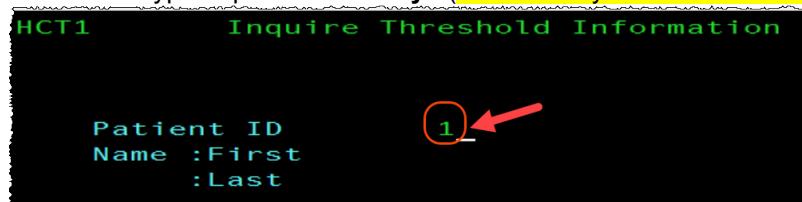
1.2.7 ► Press **PF12** to return to the previous dialog



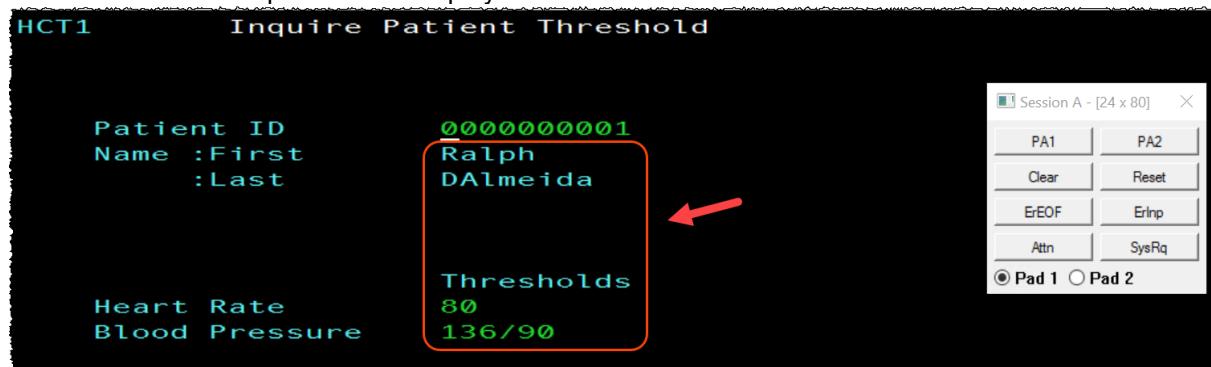
1.2.8 ► Type 8 press **Enter key**. This will invoke different programs



1.2.9 ➡ Type 1 press **Enter key**. (Notice that you can move the stick keypad to the right if needed)

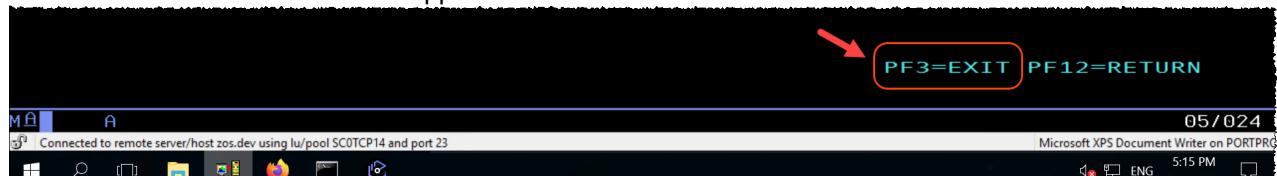


You should have the patient data displayed.

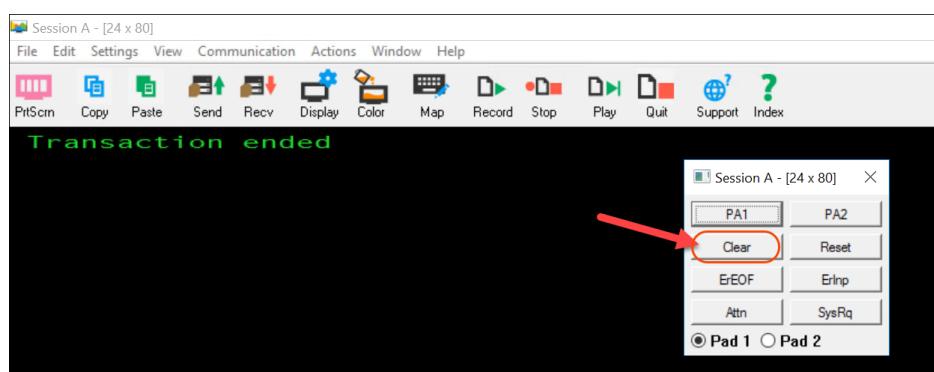


IMPORTANT: Remember the *first name* of this patient (Ralph) Later you will introduce a bug.. Instead of **Ralph** the COBOL program will display “**BAD NAME**” as first name.

1.2.10 ➡ Press **PF3** to end the application



1.2.11 ➡ After the Transaction ended, use the key pad and click **Clear**
It will clear the screen and return to the main CICS terminal..



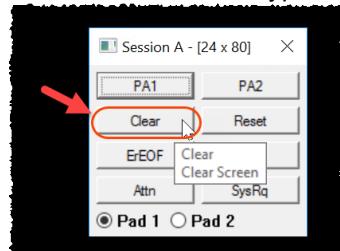
1.2.12 ➡ Type the transaction **bzue** and press **Enter key**



1.2.13 ► The following message appears in the screen.



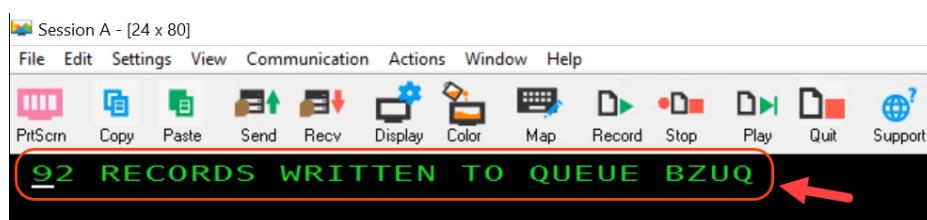
1.2.14 ► Use the Keypad to click on the **Clear** button and return to the main CICS terminal.



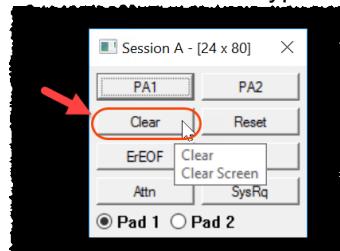
1.2.15 ► Type the transaction **bzuw** and press **Enter key**



1.2.16 This captures the recording into a CICS Transient Data Queue (TDQ) called **BZUQ** as displayed in the message.



1.2.17 ► Use the Keypad to click on the **Clear** button and return to the main CICS terminal.



1.2.18 You need to close this CICS Queue to be used in a batch job in next steps.

► Type **CEMT INQ TD** and press **Enter key**.



1.2.19 This TDQ is opened.

► Using the Tab key move the cursor under the letter "O" on the BZUQ queue.

```
INQ TD
STATUS: RESULTS - OVERTYPE TO MODIFY
Tdq(BZUC) Ext           Ena Clo
                      Shr Inp   Dat(001) Ddn(BZUCFG )
Tdq(BZUQ) Ext           Ena Ope  Dat(001) Ddn(BZUBZUQ )
                      Old Out   Dat(001) Ddn(BZUBZUQ )
Tdq(CADL) Ind Nam(CSSL)
```

1.2.20 ► Type C and press Enter key. Notice the message NORMAL

This file is now closed and can be used on a batch job.

```
INQ TD
STATUS: RESULTS - OVERTYPE TO MODIFY
Tdq(BZUC) Ext           Ena Clo
                      Shr Inp   Dat(001) Ddn(BZUCFG )
Tdq(BZUQ) Ext           Ena Clo ← NORMAL
                      Old Out   Dat(001) Ddn(BZUBZUQ )
Tdq(CADL) Ind Nam(CSSL)
```

1.2.22 You need to know which is the z/OS data set name that you closed..

► Using the tab key move the cursor on the Tdq(BZUQ) line and press Enter Key.

```
INQ TD
STATUS: RESULTS - OVERTYPE TO MODIFY
Tdq(BZUC) Ext           Ena Clo
                      Shr Inp   Dat
Tdq(BZUQ) Ext           Ena Clo
                      Old Out   Dat
Tdq(CADL) Ind Nam(CSSL)
```

1.2.23 You will see the Data set to be used in the VTP JCL that holds the recording.

This dataset is named **BZU100.ZUNIT.PLAYBACK**

```
INQ TD
RESULT = OVERTYPE TO MODIFY
Tdqueue(BZUQ)
Type(Extra)
Nameind()
Triggerlevel(      )
Enablestatus( Enabled )
Openstatus( Closed )
Termid()
Tranid()
Userid()
Disposition(Old)
Iotype(Output)
Indoubt()
Indoubtwait()
Databuffers(001)
Ddname(BZUBZU0)
Dsname(BZU100.ZUNIT.PLAYBACK) ←
Member()
+ Installtime(05/20/21 14:45:23)
```



You have now successfully created a VTP test case. It is a good practice to copy this data set into another file. In case you made mistakes on this lab on the recording we kept a version already recorded under the name **IBMUSER.HCAZ.VTP.PLAYBACK**

1.2.24 ►| Press **PF3** to end this dialog



1.2.25 It will end the dialog



1.2.26 ►| Close the terminal emulation



What have you done so far?

You executed the CICS transaction **HCAZ** and using VTP you recorded a simple interaction with the Health Care application. The recorded data is saved on a Z/OS dataset..

Section 2 – Run a JCL to execute the recorded replay sequence

Using IDz you will submit a JCL to execute on z/OS that will verify the sequence recorded by VTP against the COBOL/CICS/DB2 programs that are being used on CICSTS5.4.

2.1 Connect to Z/OS using IDz

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

►| Using the desktop double click on **IDz V15** icon.

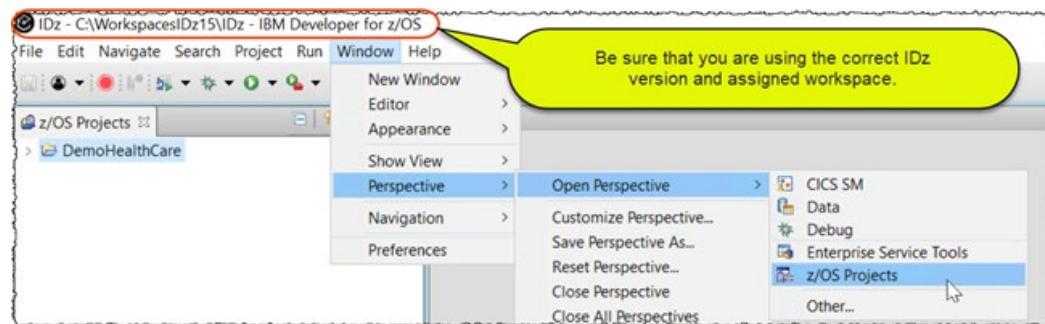
►| Verify that the message indicates that it is Version **15.0.1**

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.



2.2 Submit a provided VTP JCL for execution

2.2.1 ► Open the z/OS Projects perspective by selecting Window > Perspective > Open Perspective > z/OS Projects

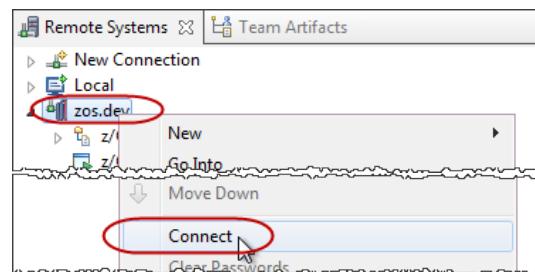


If nothing happen is because you are already at this perspective.

2.2.2 On this lab you will use userid **ibmuser** . and password **sys1**.

If you are connected as **ibmuser**, jump to step 2.2.4 Otherwise.

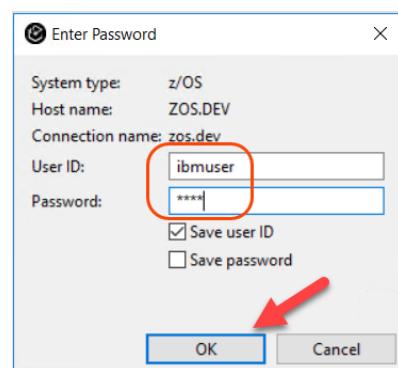
► Using Remote Systems view, right click on **zos.dev** and select **Disconnect** and then **Connect**



2.2.3 ► 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.



2.2.4 ► Using Z/OS Projects view (on left) expand the project **DemoHealthCare** expand **jcl** and double click on **VTP#HCAZ.jcl** to edit the JCL that will be submitted for execution.

This JCL runs a VTP test indicated by the file **BZU100.ZUNIT.PLAYBACK** tagged to the DDNAME **BZUPLAY**.

The programs executed on this test are loaded on the **EMPOT.ZMOBILE.TEST.LOAD** loadlib

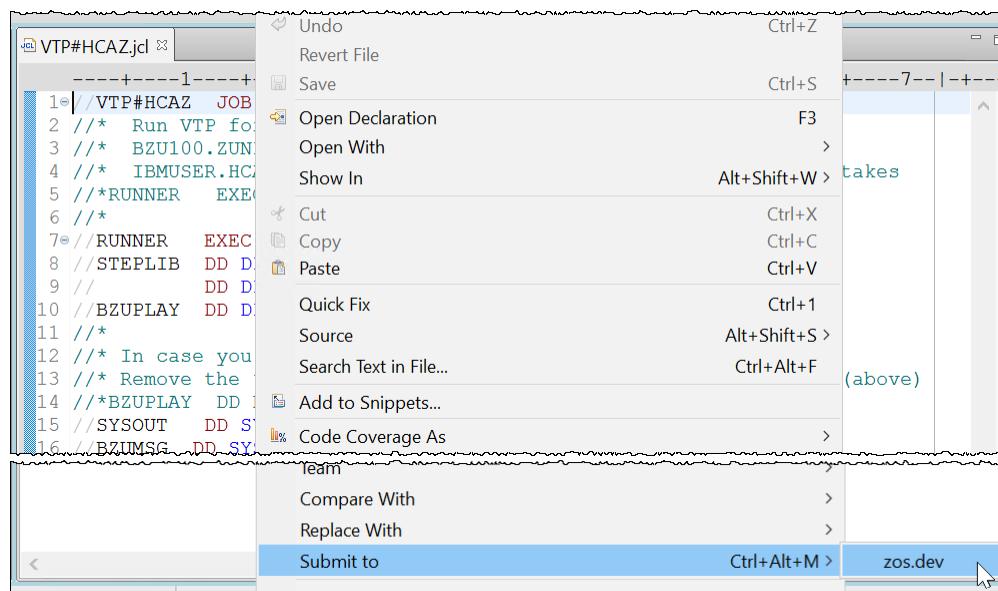
```

z/OS Projects <--> VTP#HCAZ.jcl
  +--- jcl
    +--- HCAZPLA2.jcl
    +--- HCAZPLA3.jcl
    +--- HCAZPLAY.jcl
    +--- HCAZPLCC.jcl
    +--- HCIPCC.jcl
    +--- HCIPDBG.jcl
    +--- HCIPZRUN.jcl
    +--- VTP#HCAZ.jcl (highlighted)
    +--- VTP#TES1.jcl
    +--- VTP#TES2.jcl
    +--- ZUDDBO1.jcl
    +--- ZUDDBO2.jcl
    +--- ZUNITCC.jcl

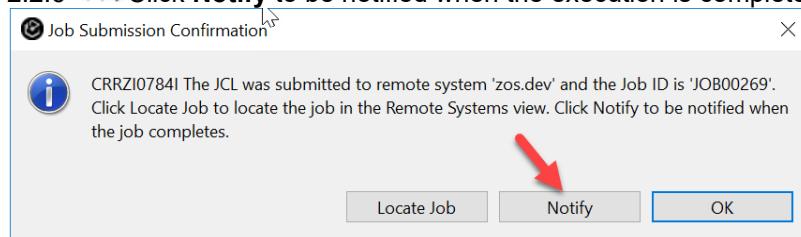
Properties Outline <--> VTP#HCAZ.jcl
  +--- 1 //> VTP#HCAZ JOB
  +--- 2 /* Run VTP for HCAZ
  +--- 3 /* BZU100.ZUNIT.PLAYBACK is your recording
  +--- 4 /* IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case of mistakes
  +--- 5 /* RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
  +--- 6 /*
  +--- 7 /* RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
  +--- 8 // STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
  +--- 9 // DD DISP=SHR,DSN=EMPOT.ZMOBILE.TEST.LOAD (highlighted)
  +--- 10 // DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD (highlighted)
  +--- 11 // BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK (highlighted)
  +--- 12 /*
  +--- 13 /* In case you have wrong results due bad recording..
  +--- 14 /* Remove the * on the line 15 (below) and comment the line 11 (above)
  +--- 15 // BZUPLAY DD DISP=SHR,DSN=IBMUSER.HCAZ.VTP.PLAYBACK
  +--- 16 // SYSOUT DD SYSOUT=*
  +--- 17 // BZUMSG DD SYSOUT=*
  +--- 18 //
  
```

2.2.5 To submit this job to z/OS execution:

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



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

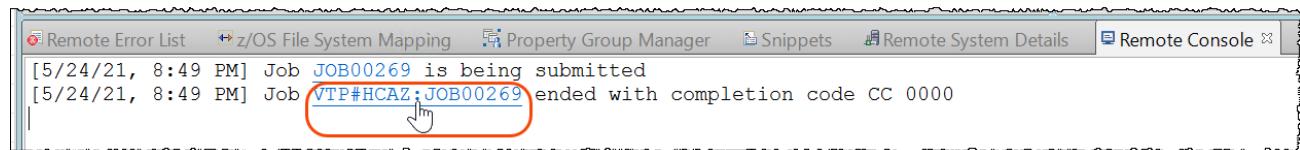


2.2.7 Under *Remote Console*, you will be notified when execution is completed.

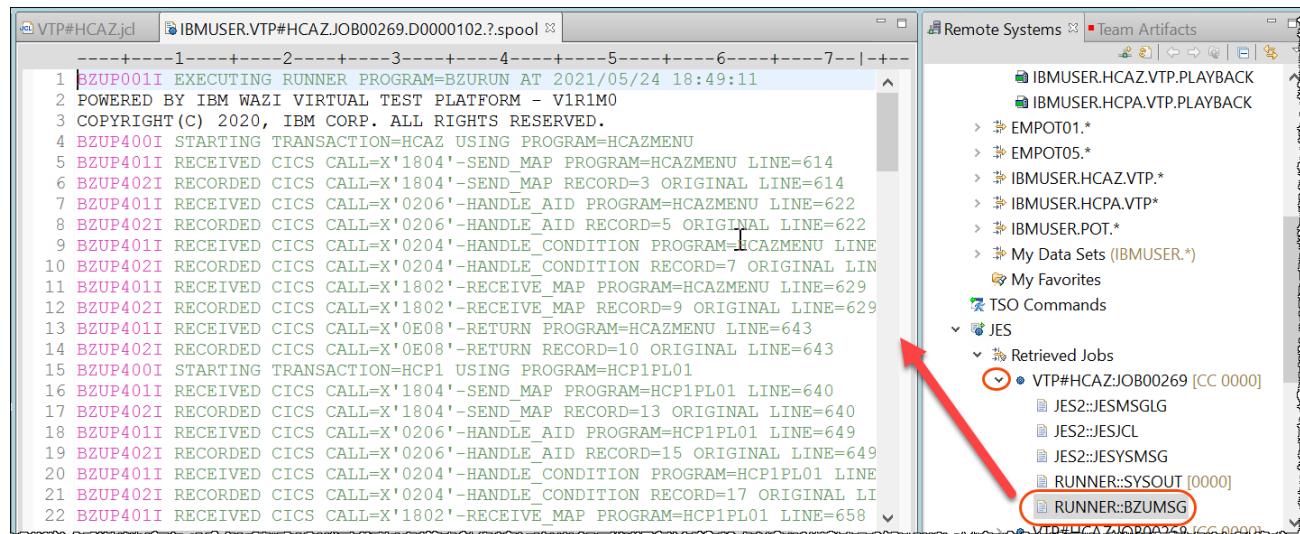
The completion code must be 0.

► Once the execution ends, click on the link **VTP#HCAZ:JOB00xxx**

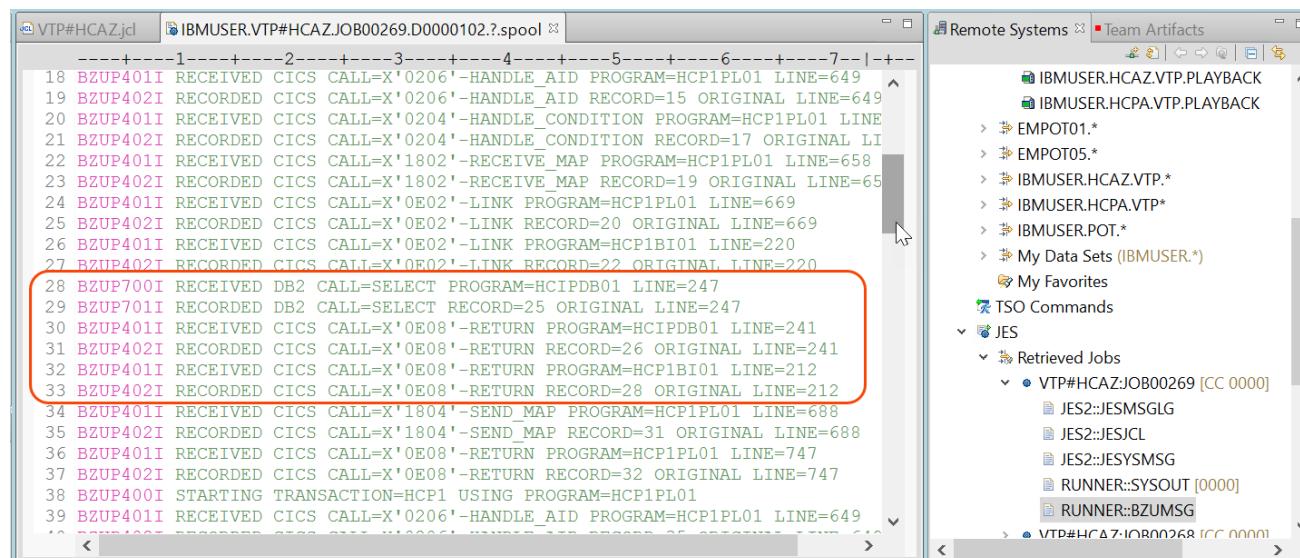
(where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



2.2.8 ► Under *Remote Systems* view expand **VTP#HCAZ:JOB00xxx** and double click **RUNNER:BZUMSG** step.



2.2.9 ► Scroll down the report displayed. In the line 28 you will see that the program **HCPDB01** has **DB2 CALL=SELECT** and is invoked by program **HCP1BI01**



2.2.10 ► Scroll down to the bottom and you will see that the execution finished with RC=00. That means that all data captured on the recording matches the execution. Later you will introduce a bug and verify that the return code will not be 00.

The screenshot shows the IBM Z VTP interface. On the left, a log file titled 'IBMUSER.VTP#HCAZ.JOB00269.D0000102.?spool' is displayed. The log contains several lines of CICS transaction logs, ending with line 103: 'BZUP002I FINISHED EXECUTION RC=00'. A red arrow points from this line to the right margin of the log window. On the right, a 'Remote Systems' window titled 'Team Artifacts' is open, showing a tree view of retrieved jobs. One job, 'RUNNER:BZUMSG', is highlighted with a red circle and a red arrow pointing to it from the log window.

```

96 BZUP401I RECEIVED CICS CALL=X'1802'-RECEIVE_MAP PROGRAM=HCT1PL01 LINE=653
97 BZUP402I RECORDED CICS CALL=X'1802'-RECEIVE_MAP RECORD=89 ORIGINAL LINE=65
98 BZUP401I RECEIVED CICS CALL=X'1806'-SEND_TEXT PROGRAM=HCT1PL01 LINE=734
99 BZUP402I RECORDED CICS CALL=X'1806'-SEND_TEXT RECORD=91 ORIGINAL LINE=734
100 BZUP401I RECEIVED CICS CALL=X'0E08'-RETURN PROGRAM=HCT1PL01 LINE=740
101 BZUP402I RECORDED CICS CALL=X'0E08'-RETURN RECORD=92 ORIGINAL LINE=740
102 ****
103 BZUP002I FINISHED EXECUTION RC=00
104 ****
105 BZUP300I DB2 STATISTICS
106 SELECT NORMAL=3
107 ****
108 BZUP300I CICS STATISTICS
109 HANDLE_CONDITION NORMAL=6
110 HANDLE_AID NORMAL=6
111 LINK NORMAL=6
112 RETURN NORMAL=12
113 RECEIVE_MAP NORMAL=6
114 SEND_MAP NORMAL=6
115 SEND_TEXT NORMAL=1
116 ****

```

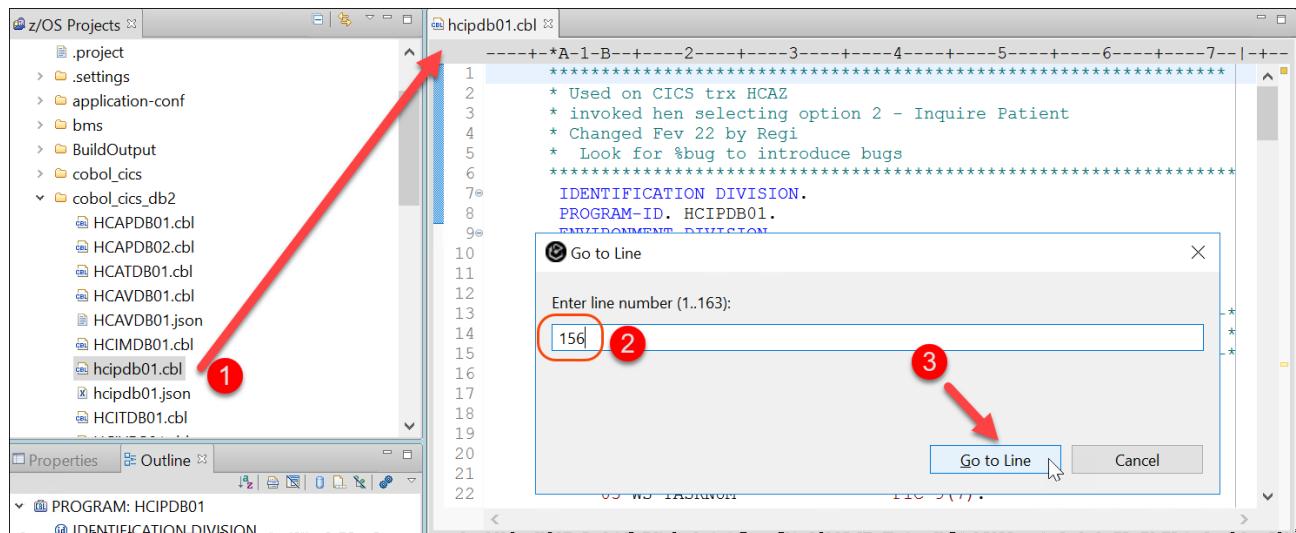
2.2.11 ► Use **Ctrl + Shift + F4** to close all opened editors.

Section 3 – Modify one program (introduce a bug) and rerun the VTP JCL

Using IDz you will modify the program and introduce a bug. Then you will rerun the batch test created with VTP in the previous sections.

3.1 Modifying one COBOL program introducing a bug

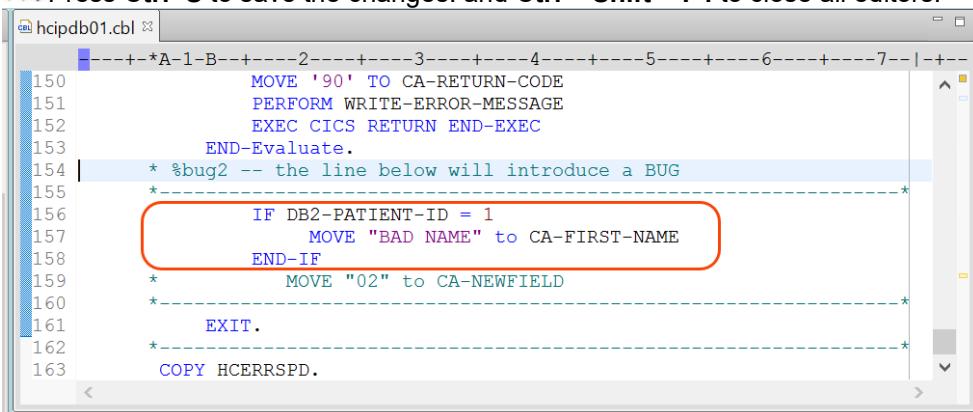
- 3.1.1 ► Using the z/OS Projects view, open **hcipdb01.cbl** under **cobol_cics_db2** by double clicking on it
 3.1.2 ► On the editor, go to line **156** by pressing **Ctrl+L** typing **156** and clicking **Go to Line**.



3.1.3 ► Change the lines 156, 157 and 158 removing the * from the statements that move "BAD NAME", to a COBOL field named CA-FIRST-NAME **Tip -> Could use Source > Toggle Comment**

IMPORTANT → DO NOT modify the line 159.

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



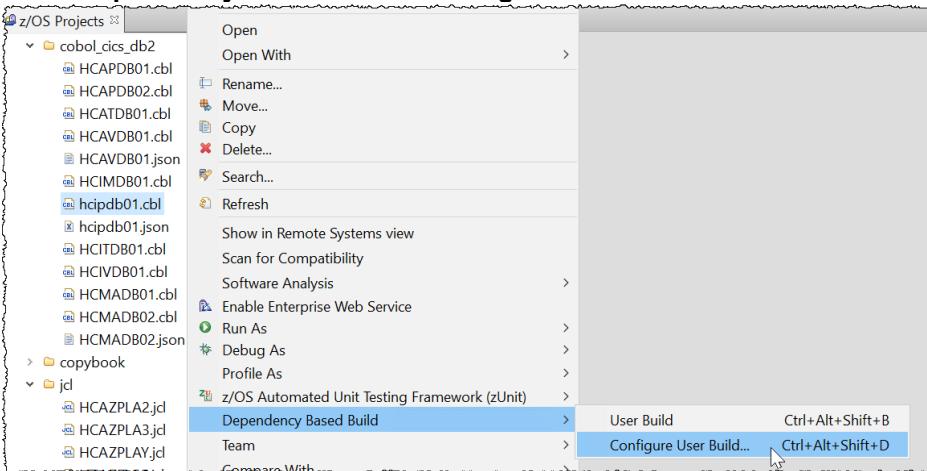
```

150      MOVE '90' TO CA-RETURN-CODE
151      PERFORM WRITE-ERROR-MESSAGE
152      EXEC CICS RETURN END-EXEC
153      END-Evaluate.
154      * %bug2 -- the line below will introduce a BUG
155      *
156      IF DB2-PATIENT-ID = 1
157          MOVE "BAD NAME" to CA-FIRST-NAME
158          END-IF
159      *      MOVE "02" to CA-NEWFIELD
160      *
161      EXIT.
162      *
163      COPY HCERRSPD.

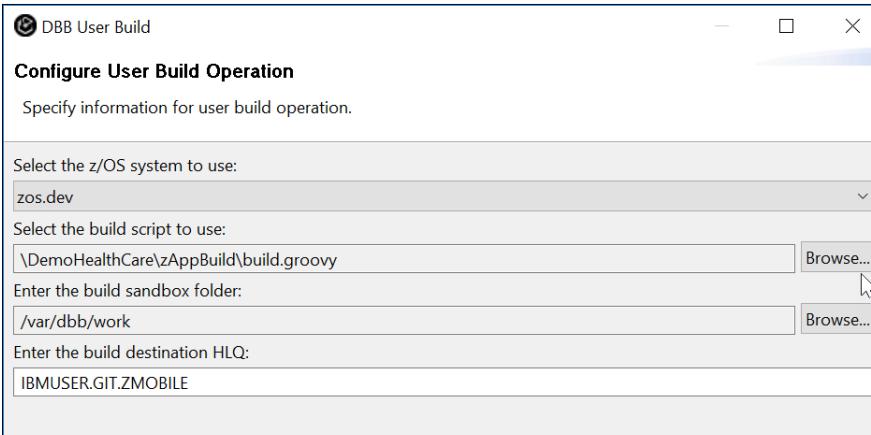
```

3.2 Building the modified program using DBB

3.2.1 ► On the z/OS Projects view, right click on **hcpdb01.cbl** and select **Dependency Based Build > Configure User Build...**



3.2.2 ► If the values are not populated, use the Browse button, specify the values below and **click Next three times**



DBB User Build

Configure User Build Operation

Specify information for user build operation.

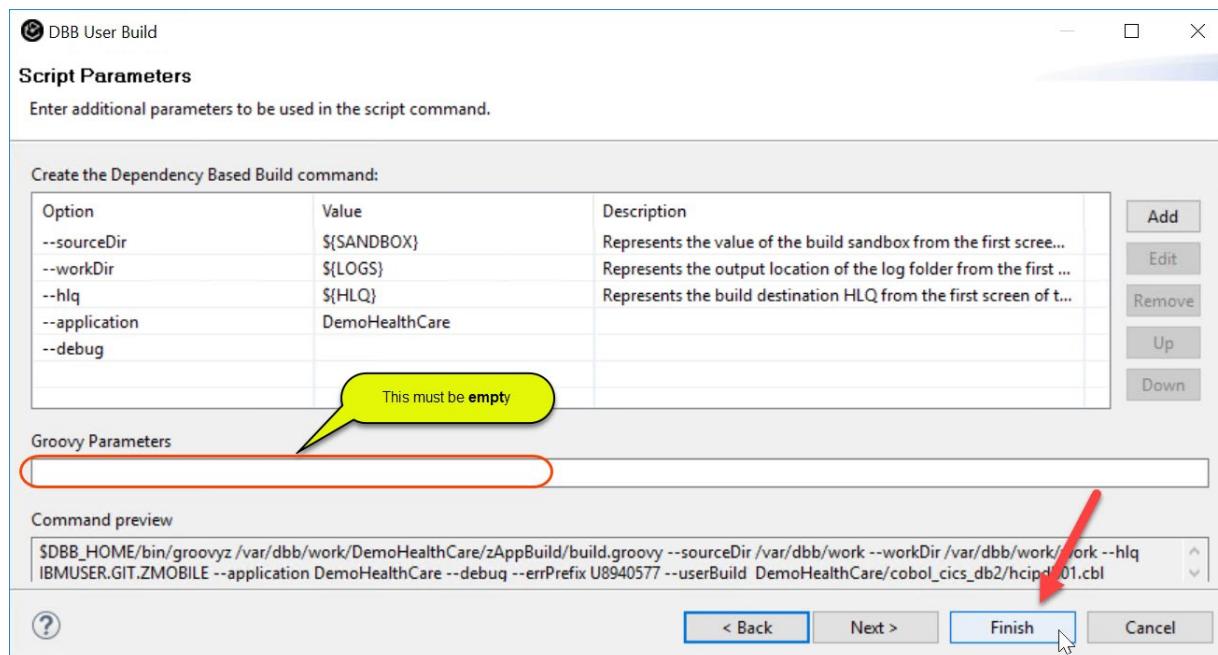
Select the z/OS system to use:
zos.dev

Select the build script to use:
\DemoHealthCare\zAppBuild\build.groovy

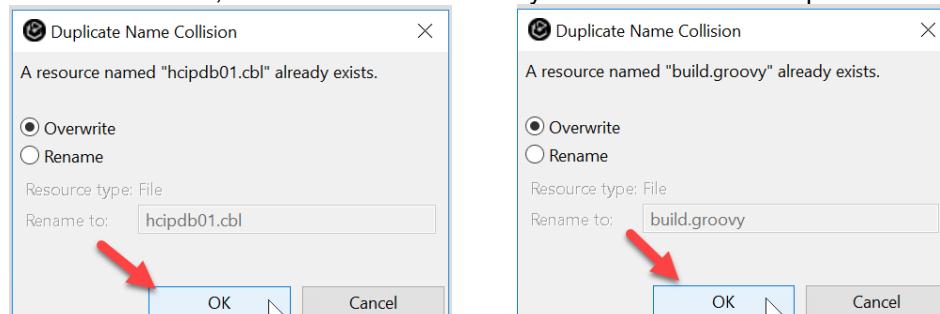
Enter the build sandbox folder:
/var/dbb/work

Enter the build destination HLQ:
IBMUSER.GIT.ZMOBILE

3.2.3 ► Be sure that the field **Groovy Parameters** is empty and click **Finish**

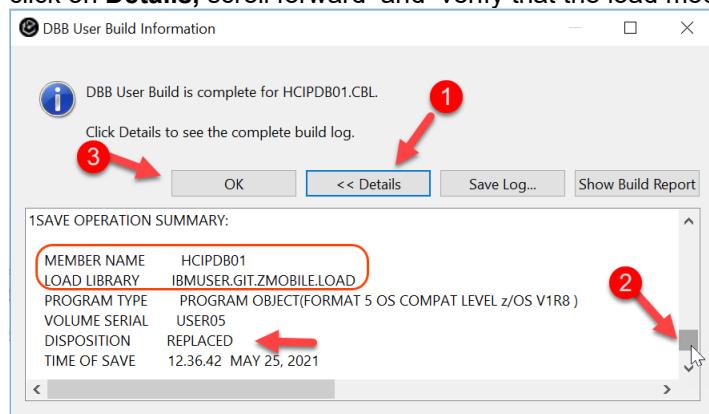


3.2.4 ► If asked, overwrite the code already on z/OS as the example below

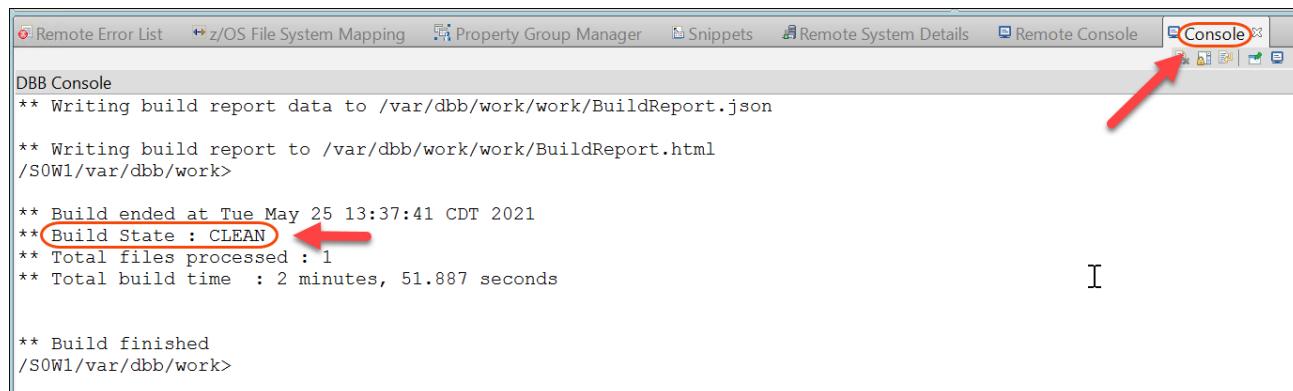


3.2.5 ► Click on **Console** tab (left of Remote Console) to verify what is going on..

► This operation may take 2 to 3 minutes. When finished the dialog below will be displayed and you can click on **Details**, scroll forward and verify that the load module was created. Then click **OK**.



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



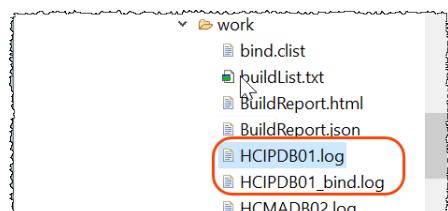
```
DBB Console
** Writing build report data to /var/dbb/work/work/BuildReport.json
** Writing build report to /var/dbb/work/work/BuildReport.html
/SOW1/var/dbb/work>

** Build ended at Tue May 25 13:37:41 CDT 2021
** Build State : CLEAN
** Total files processed : 1
** Total build time : 2 minutes, 51.887 seconds

** Build finished
/SOW1/var/dbb/work>
```

3.2.7 The logs of the COBOL Compiler/Link is at **z/OS UNIX Files** on
/var/dbb/work/work/HCIPDB01.log

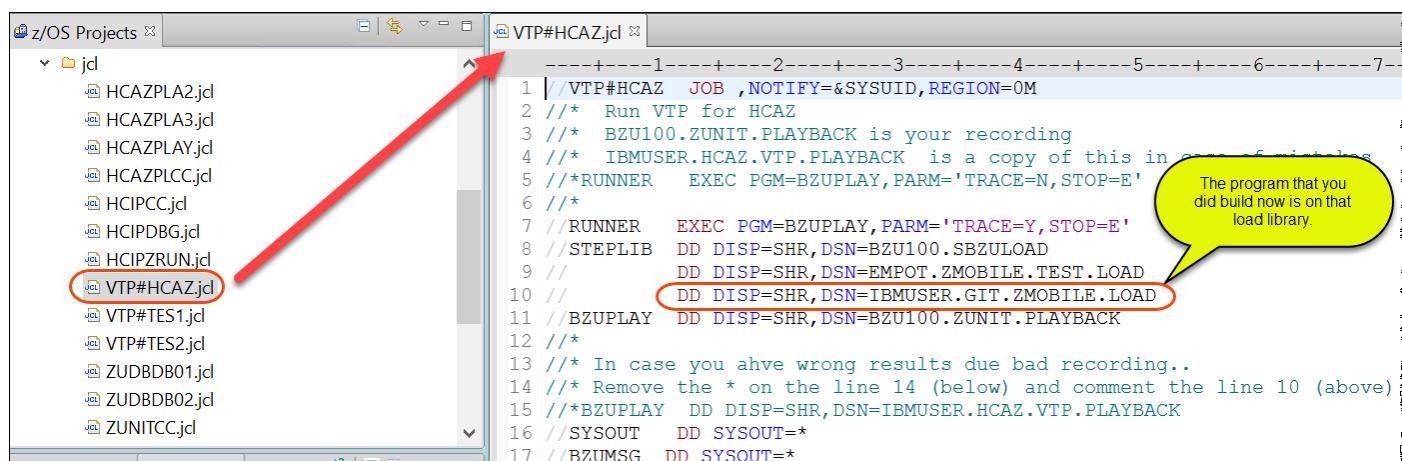
The log of the DB2 Bind is at **/var/dbb/work/work/HCIPDB01_bind.log**



3.3 Running the VTP JCL again against the modified program

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

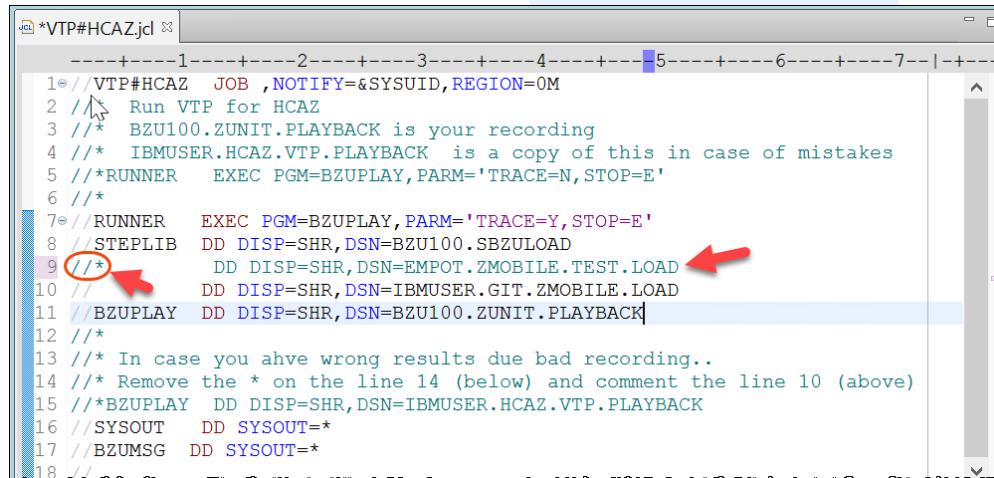
3.3.1  Using the project **DemoHealthCare** expand **jcl** and double click on **VTP#HCAZ.jcl** to edit the JCL that will invoke the VTP.



```
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7-
1 //VTP#HCAZ JOB ,NOTIFY=&SYSUID,REGION=0M
2 //* Run VTP for HCAZ
3 // BZU100.ZUNIT.PLAYBACK is your recording
4 // IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case of mistakes
5 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
6 //*
7 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
8 //STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
9 //
10 // DD DISP=SHR,DSN=EMFOT.ZMOBILE.TEST.LOAD
11 //BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK
12 //*
13 //* In case you ahve wrong results due bad recording..
14 //* Remove the * on the line 14 (below) and comment the line 10 (above)
15 //BZUPLAY DD DISP=SHR,DSN=IBMUSER.HCAZ.VTP.PLAYBACK
16 //SYSOUT DD SYSOUT=*
17 //BZUMSG DD SYSOUT=*
```

3.3.2 The modified program is now at **IBMUSER.GIT.ZMOBILE.LOAD**

► Add an * as below to comment the PDS **EMPOT.ZMOBILE.TEST.LOAD**



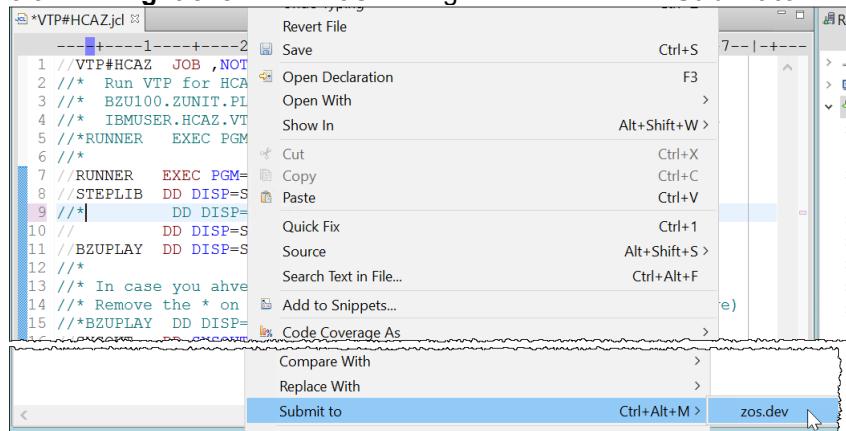
```

*VTP#HCAZ.jcl
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8---+
1 //> VTP#HCAZ JOB ,NOTIFY=&SYSUID,REGION=0M
2 //> Run VTP for HCAZ
3 //* BZU100.ZUNIT.PLAYBACK is your recording
4 //** IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case of mistakes
5 //**RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
6 /**
7 //> /RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
8 //> STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
9 //> /* DD DISP=SHR,DSN=EMPOT.ZMOBILE.TEST.LOAD
10 //> DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD
11 //> BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK
12 /**
13 //** In case you ahve wrong results due bad recording..
14 //** Remove the * on the line 14 (below) and comment the line 10 (above)
15 //**BZUPLAY DD DISP=SHR,DSN=IBMUSER.HCAZ.VTP.PLAYBACK
16 //SYSOUT DD SYSOUT=*
17 //BZUMSG DD SYSOUT=*
18 //

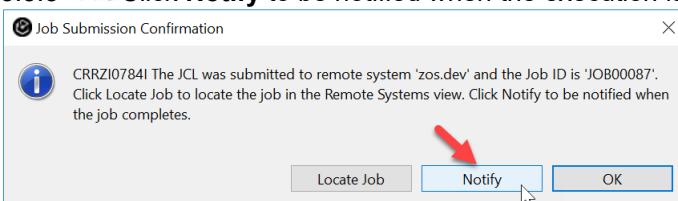
```

3.3.3 ► Use **Ctrl + S** to save the JCL updates,

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



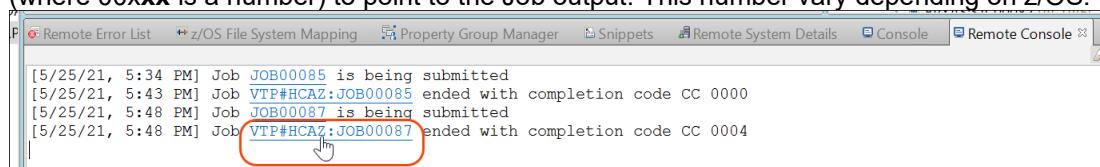
3.3.5 ► Click **Notify** to be notified when the execution is complete.



3.3.6 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 [VTP#HCAZ:JOB00xxx](#)** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



3.3.7 ► Under **Remote Systems** view expand **VTP#HCAZ:JOB00xxx (CC0004)** and double click **RUNNER:BZUMSG** step.

3.3.8 ► Scroll down the report displayed and on line 35 you will see that the program **HCIPDB01** has **BAD NAME** while the recorded data was **Ralph**

```

*VTP#HCAZ.jcl IBMUSER.VTP#HCAZ.JOB00087.D0000102?.spool
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7---|+-
20 BZUP401I RECEIVED CICS CALL=X'0E02'-LINK TROGR=ACT1B1C1 LINE=220
21 BZUP402I RECORDED CICS CALL=X'0E02'-LINK RECORD=22 ORIGINAL LINE=220
22 BZUP700I RECEIVED DB2 CALL=SELECT PROGRAM=HCIPDB01 LINE=235
23 BZUP701I RECORDED DB2 CALL=SELECT RECORD=25 ORIGINAL LINE=247
24 BZUP401I RECEIVED CICS CALL=X'0E08'-RETURN PROGRAM=HCIPDB01 LINE=230
25 BZUP402I RECORDED CICS CALL=X'0E08'-RETURN RECORD=26 ORIGINAL LINE=241
26 BZUP202W REQUEST=RETURN ARG=CMA RECORD=26 ADDRESS=20757100 OFFSET=28
27 PROGRAM=HCIPDB01 LINE=230 MODE=INPT
28 PROGRAM COLS=-+---3---+---4---+---5---+---6---+---7---+
29 PROGRAM DATA=1234BAD NAME DALmeida 1980-07-1134 Main
30 PROGRAM DATA=FFFFC4DCDC44CC9988844444444444FFFF6FF6FFFF4D889
31 PROGRAM DATA=12342140514500413459410000000000001980070113404195
32 BZUP202W REQUEST=RETURN ARG=CMA RECORD=26 ADDRESS=20757100 OFFSET=28
33 PROGRAM=HCIPDB01 LINE=230 MODE=INPT
34 PROGRAM COLS=-+---3---+---4---+---5---+---6---+---7---+
35 PROGRAM DATA=1234BAD NAME DALmeida 1980-07-1134 Main
36 PROGRAM DATA=FFFFC4DCDC44CC9988844444444444FFFF6FF6FFFF4D889
37 PROGRAM DATA=12342140514500413459410000000000001980070113404195
38 RECORDED DATA=1234Ralph DALmeida 1980-07-1134 Main
39 RECORDED DATA=FFFFD899844444CC9988844444444444FFFF6FF6FFFF4D889
40 RECORDED DATA=1234913780000413459410000000000001980070113404195
41 BZUP401I RECEIVED CICS CALL=X'0E08'-RETURN PROGRAM=HCP1BI01 LINE=212
42 BZUP402I RECORDED CICS CALL=X'0E08'-RETURN RECORD=28 ORIGINAL LINE=212
43 BZUP202W REQUEST=RETURN ARG=CMA RECORD=28 ADDRESS=20757100 OFFSET=28
44 PROGRAM=HCP1BI01 LINE=212 MODE=INPT
45 PROGRAM COLS=-+---3---+---4---+---5---+---6---+---7---+
46 PROGRAM DATA=1234BAD NAME DALmeida 1980-07-1134 Main
47 PROGRAM DATA=FFFFC4DCDC44CC9988844444444444FFFF6FF6FFFF4D889
48 PROGRAM DATA=12342140514500413459410000000000001980070113404195

```

3.3.9 This means that the data returned by the program HCIPDB01 is not the same as it was recorded. That could indicate a fail on the program.

► Scroll down to the bottom and you will see that the execution finished with RC=04. You will see various mismatches. That means that not all data captured on the recording matched during the execution.

```

*VTP#HCAZ.jcl IBMUSER.VTP#HCAZ.JOB00119.D0000102?.spool
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7---|+-
213 BZUP401I RECEIVED CICS CALL=X'1802'-RECEIVE_MAP PROGRAM=HCT1PL01 LINE=653
214 BZUP402I RECORDED CICS CALL=X'1802'-RECEIVE_MAP RECORD=89 ORIGINAL LINE=65
215 BZUP401I RECEIVED CICS CALL=X'1806'-SEND_TEXT PROGRAM=HCT1PL01 LINE=734
216 BZUP402I RECORDED CICS CALL=X'1806'-SEND_TEXT RECORD=91 ORIGINAL LINE=734
217 BZUP401I RECEIVED CICS CALL=X'0E08'-RETURN PROGRAM=HCT1PL01 LINE=740
218 BZUP402I RECORDED CICS CALL=X'0E08'-RETURN RECORD=92 ORIGINAL LINE=740
219 ****
220 BZUP002I FINISHED EXECUTION RC=04
221 ****
222 BZUP300I DB2 STATISTICS
223 SELECT NORMAL=3
224 ****
225 BZUP300I CICS STATISTICS
226 HANDLE_CONDITION NORMAL=6
227 HANDLE_AID NORMAL=6
228 LINK NORMAL=6
229 RETURN NORMAL=12
230 RECEIVE_MAP NORMAL=6
231 SEND_MAP NORMAL=6
232 SEND_TEXT NORMAL=1
233 ****
234

```

3.3.10 ►| Use **Ctrl + Shift + F4** to close all opened editors.

What have you done so far?

On Section 1 -You executed the CICS transaction **HCAZ** and using VTP you recorded a simple interaction with the *Health Care application*. The recorded data is saved on a Z/OS dataset..



On Section 2 -You executed the VTP JCL that execute the sequence recorded. Since you made no changes the return code must be 00.

On Section 3 -You modified the COBOL program and introduced a bug. The program modified was compiled and DB2 was bind using the DBB User Build capability. Again the VTP JCL is submitted for batch execution and we can verify the bug on the batch output execution.



You introduced a bug on the application.
On the next part you must fix it and verify that its fixed using the **IBM Z VTP**.

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

Section 4. Run the CICS transaction and verify the bug .

Since the modified load module was deployed to a loadlib that is on the datasets that CICS uses you could see the bug on the execution. You may need to execute a CICS NEWCOPY to see that.
Below the details

4.1 Issuing a CICS New copy using 3270 emulation terminal

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



This opens the host emulator.

4.1.2 ►| Type **I cicsts53**. (where I is L lower case) and press **Enter key**.

```
====> 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 cicsts53_
```

B

24/011

4.1.3 ► Sign on using **ibmuser** as the userid and **sys1** as the password and press **Enter key**.

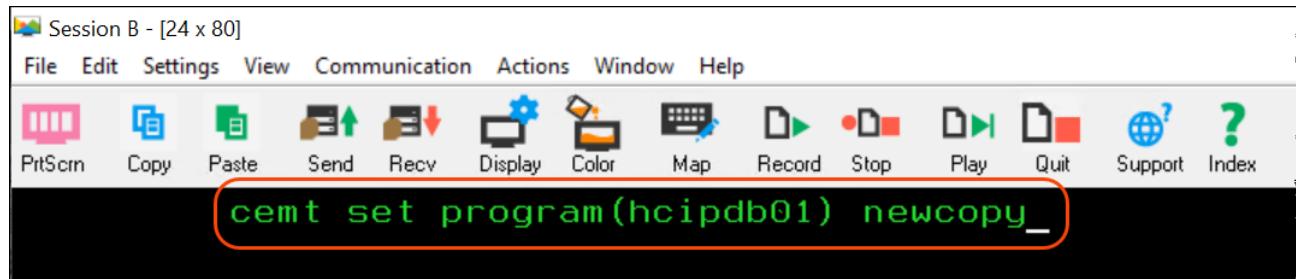
WELCOME TO CICS TS 5.3

Type your userid and password, then press ENTER:

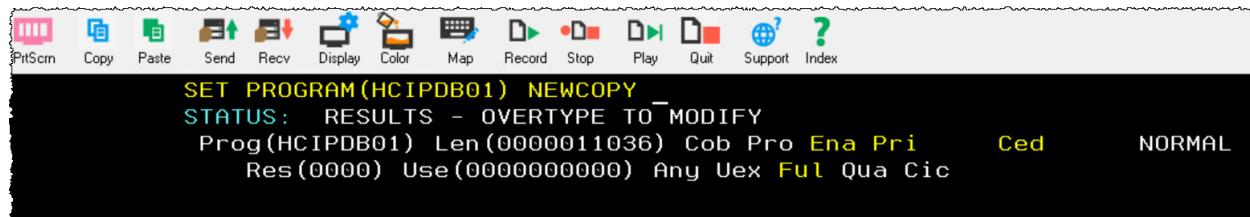
Userid . . .	ibmuser	Groupid . . .	_____
Password . . .	—		
Language . . .	—		

New Password . . .

4.1.4 ► Type **cemt set program(hcipdb01) newcopy** and press **Enter key**.



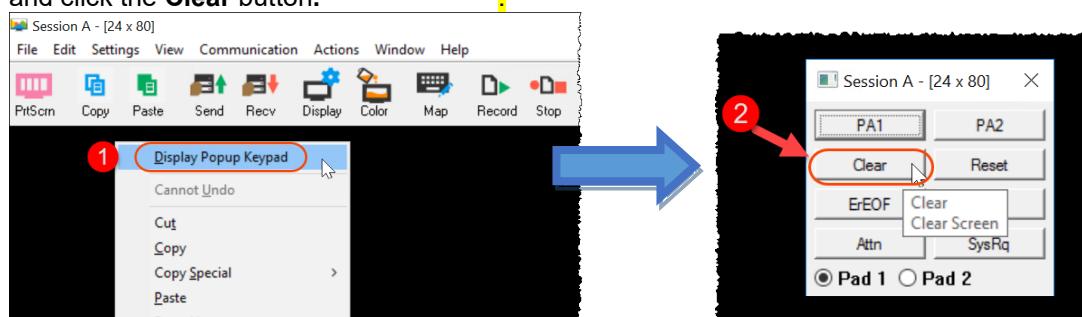
4.1.5 This below indicates the **NEWCOPY** was successful:



4.1.6 ► Press **PF3** to end the dialog



4.1.7 ► To clear the screen, right click on the dark space and select **Display Popup Keypad** and click the **Clear** button.

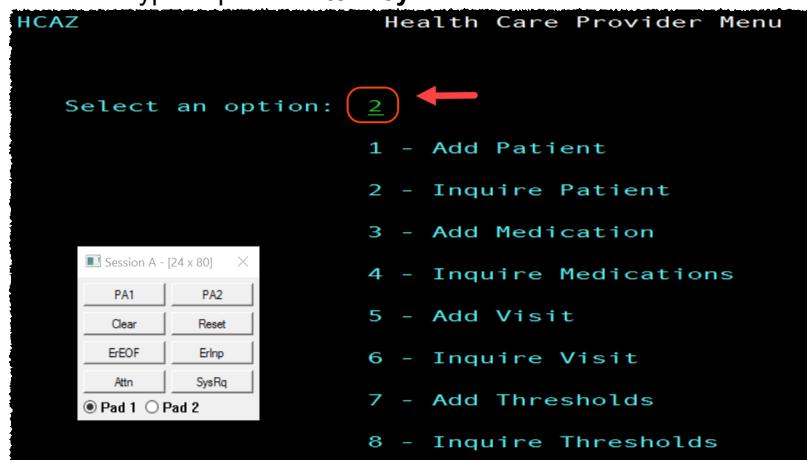


4.2 Executing HCAZ transaction

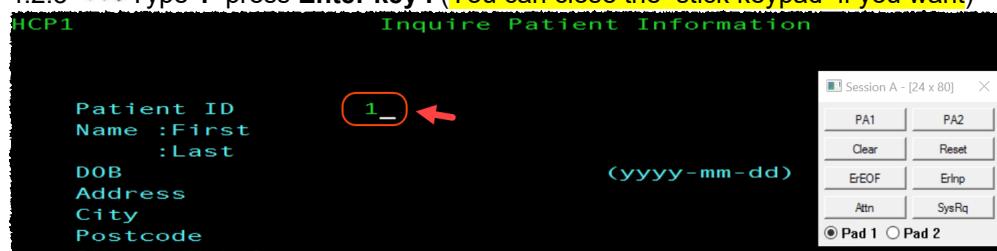
4.2.1 ➡ On CICS main terminal type **hcaz** and press **Enter key** to invoke the CICS transaction application



4.2.2 ➡ Type 2 press **Enter key**.



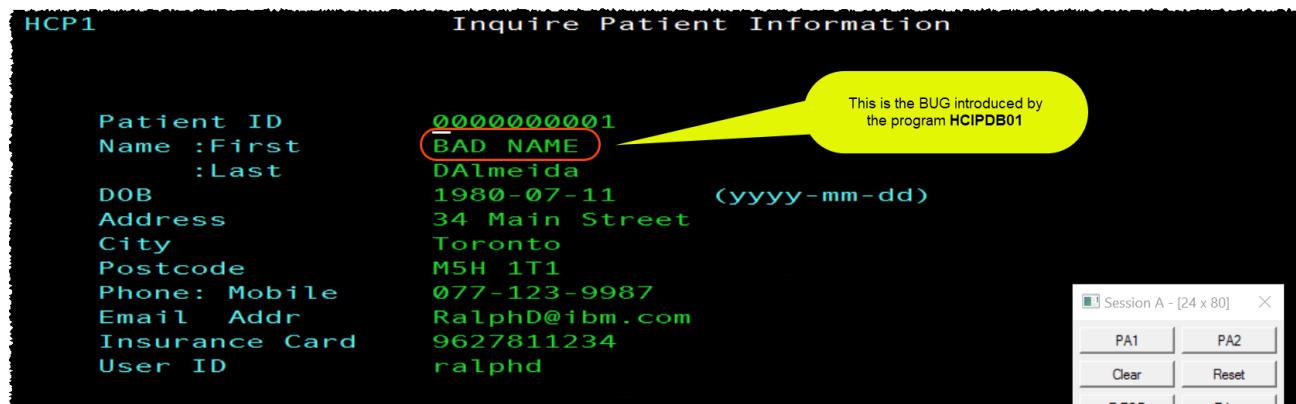
4.2.3 ➡ Type 1 press **Enter key**. (You can close the stick keypad if you want)



You should have the patient data displayed.

Notice that instead of **Ralph** the **BAD NAME** is displayed

This confirm the report provided by the batch JCL executed before.



4.2.4 ➡ Press PF3 to exit the program



4.2.5 ➡ Close the terminal emulation



As the IBM Z VTP batch execution pointed the program now has a bug.
The patient first name now is **BAD NAME** instead of **Ralph**.

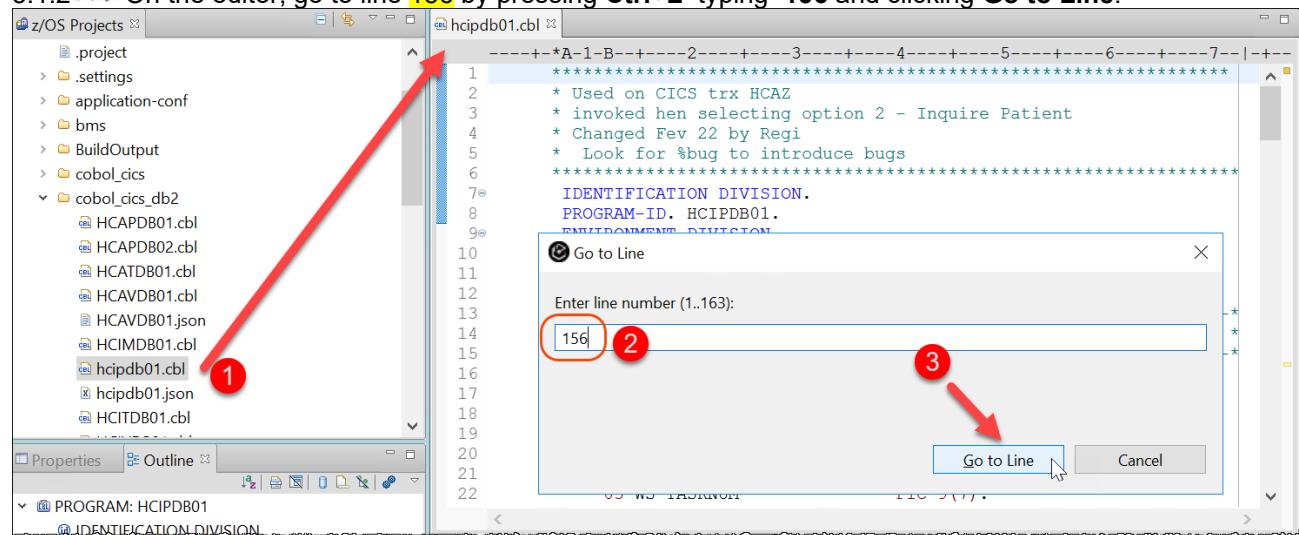
Section 5 Use IDz to fix the bug and recompile/bind the program.

Using IDz again you will remove the bug introduced Below the details

5.1 Modifying one COBOL program again to remove the bug

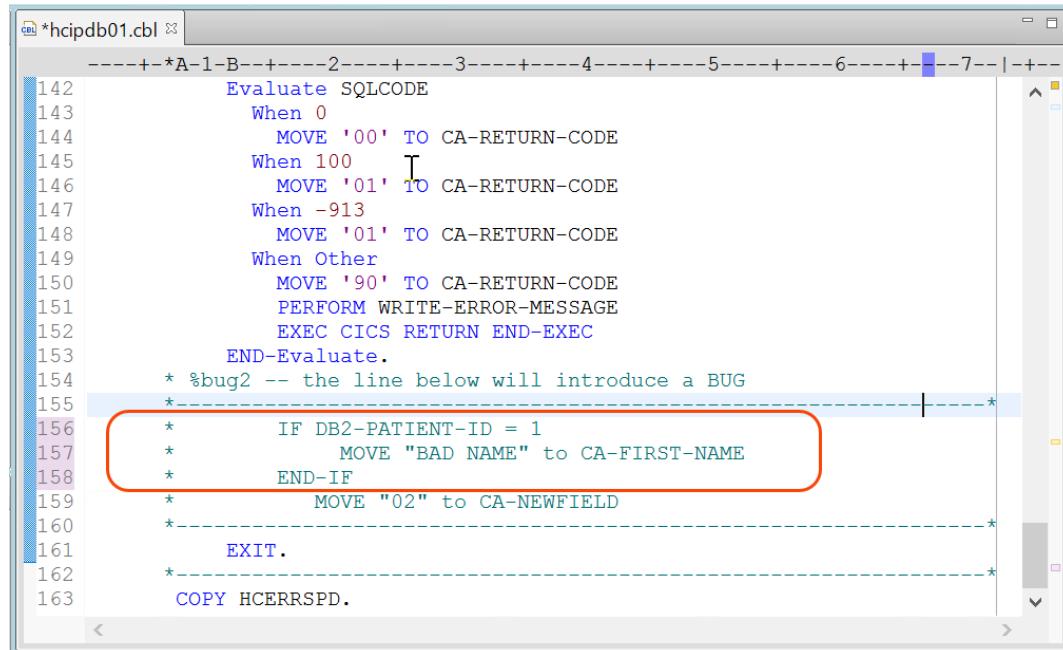
5.1.1 ➡ Using IDz and the z/OS Projects view, open **hcipdb01.cbl** under **cobol_cics_db2** by double clicking on it

5.1.2 ➡ On the editor, go to line 156 by pressing **Ctrl+L** typing **156** and clicking **Go to Line**.



5.1.3 ► Change the lines 156, 157 and 158 adding the * on column 7 to the statements that move "BAD NAME", to a COBOL field named CA-FIRST-NAME **Tip -> Could use Source > Toggle Comment**

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



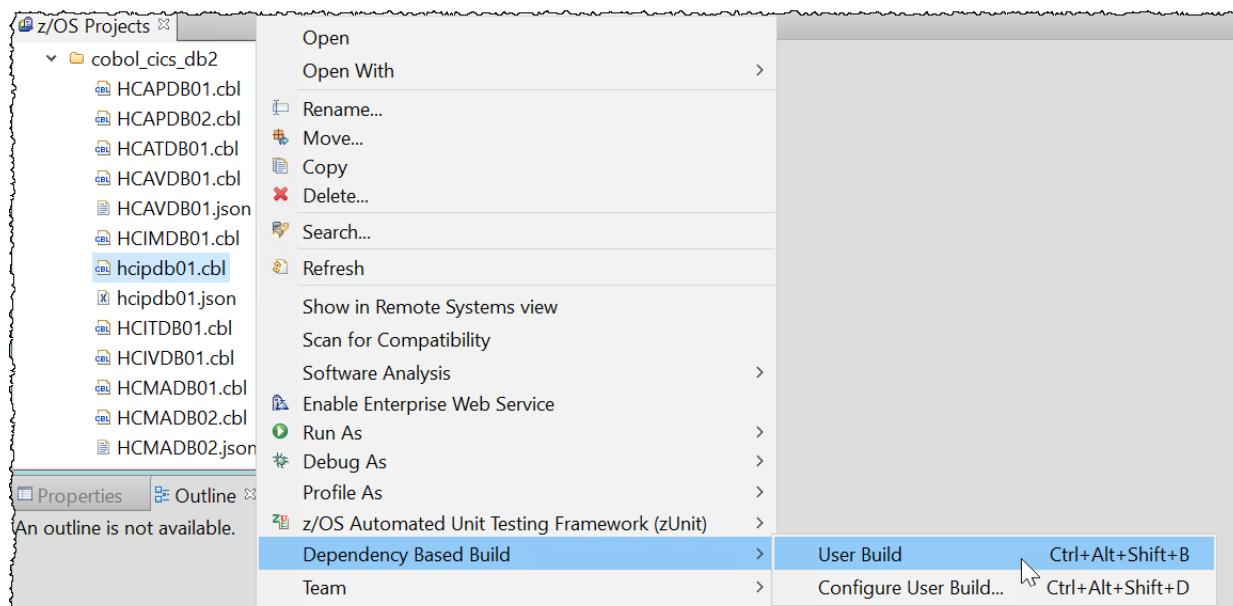
```

142      Evaluate SQLCODE
143          When 0
144              MOVE '00' TO CA-RETURN-CODE
145          When 100
146              MOVE '01' TO CA-RETURN-CODE
147          When -913
148              MOVE '01' TO CA-RETURN-CODE
149          When Other
150              MOVE '90' TO CA-RETURN-CODE
151          PERFORM WRITE-ERROR-MESSAGE
152          EXEC CICS RETURN END-EXEC
153      END-Evaluate.
154      * %bug2 -- the line below will introduce a BUG
155      *-----*
156      *       IF DB2-PATIENT-ID = 1
157      *           MOVE "BAD NAME" to CA-FIRST-NAME
158      *       END-IF
159      *       MOVE "02" to CA-NEWFIELD
160      *-----*
161      EXIT.
162      *-----*
163      COPY HCERRSPD.

```

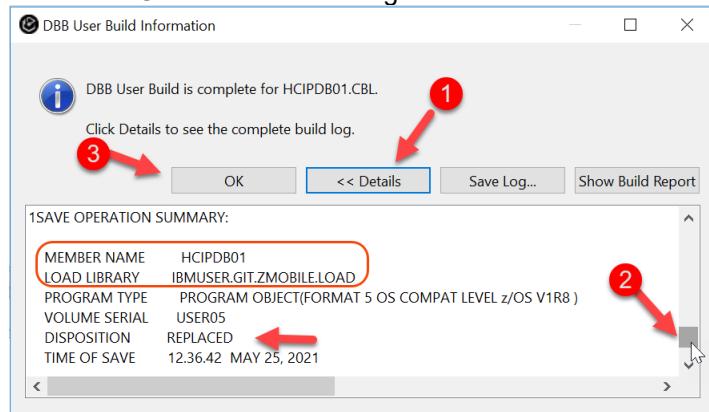
5.2 Rebuilding the fixed program using DBB

5.2.1 ► On the z/OS Projects view, right click on **hcpdb01.cbl** and select **Dependency Based Build > User Build...**



5.2.3 ► Click on the **Console** view (left of Remote Console) to see the results.
This operation may take up to 2 minutes

► When finished click on **Details**, advance forward and verify that the load module was created. Then click **OK** to close the dialog.



5.2.4 ► Click on the **Console** view to see the results:

```
** Writing build report data to /var/dbb/work/work/BuildReport.json
** Writing build report to /var/dbb/work/work/BuildReport.html
/S0W1/var/dbb/work>

** Build ended at Tue May 25 13:37:41 CDT 2021
** Build State : CLEAN
** Total files processed : 1
** Total build time : 2 minutes, 51.887 seconds

** Build finished
/S0W1/var/dbb/work>
```

The logs of the COBOL Compiler/Link is at </var/dbb/work/work/HCIPDB01.log>
The log of the DB2 Bind is at /var/dbb/work/work/HCIPDB01_bind.log

Section 6 Rerun the VTP JCL and verify that the bug is eliminated.

Once you fixed the program you may resubmit the VTP JCL and verify, that there is no errors

6.1 Running the VTP JCL again against the modified program

Since we fixed the bug , now the VTP test should execute with Return Code 00..

6.1.1 ► Using the project **DemoHealthCare** expand **jcl** and double click on **VTP#HCAZ.jcl** to edit the JCL that runs VTP.

```
1 // VTP#HCAZ JOB ,NOTIFY=&SYSUID,REGION=0M
2 /* Run VTP for HCAZ
3 // BZU100.ZUNIT.PLAYBACK is your recording
4 // IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case you ahve wrong results
5 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
6 /**
7 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
8 //STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
9 // DD DISP=SHR,DSN=EMPT.ZMOBILE.TEST.LOAD
10 // DD DISP=SHR,DSN=IBMUER.GIT.ZMOBILE.LOAD
11 //BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK
12 /**
13 /* In case you ahve wrong results due bad recording..
14 /* Remove the * on the line 14 (below) and comment the line 10 (above)
15 //BZUPLAY DD DISP=SHR,DSN=IBMUER.HCAZ.VTP.PLAYBACK
16 // SYSOUT DD SYSOUT=*
17 //BZUMSG DD SYSOUT=*
```

6.1.2 The modified program is now at **IBMUSER.GIT.ZMOBILE.LOAD**

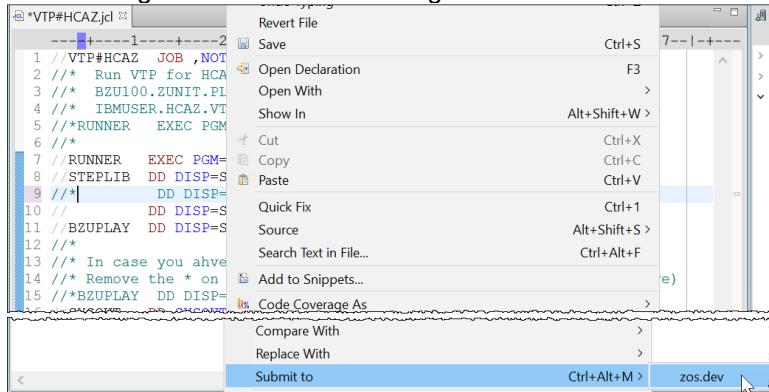
► If necessary add an * as below to comment the PDS **EMPOT.ZMOBILE.TEST.LOAD**

```

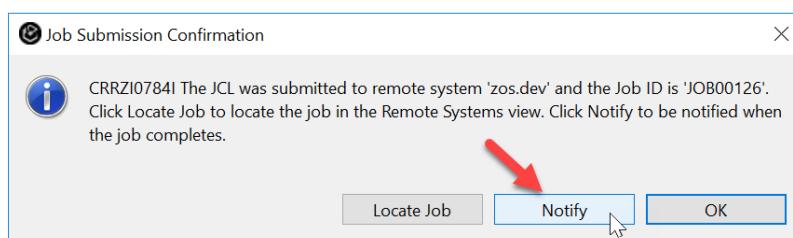
*VTP#HCAZ.jcl
1 //VTP#HCAZ JOB ,NOTIFY=&SYSUID,REGION=0M
2 /* Run VTP for HCAZ
3 // BZU100.ZUNIT.PLAYBACK is your recording
4 // IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case of mistakes
5 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
6 /**
7 //RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
8 //STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
9 //          DD DISP=SHR,DSN=EMPOT.ZMOBILE.TEST.LOAD
10 //         DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD
11 //BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK
12 /**
13 /* In case you have wrong results due bad recording..
14 /* Remove the * on the line 14 (below) and comment the line 10 (above)
15 //BZUPLAY DD DISP=SHR,DSN=IBMUSER.HCAZ.VTP.PLAYBACK
16 //SYSOUT DD SYSOUT=*
17 //BZUMSG DD SYSOUT=*
18 /**

```

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



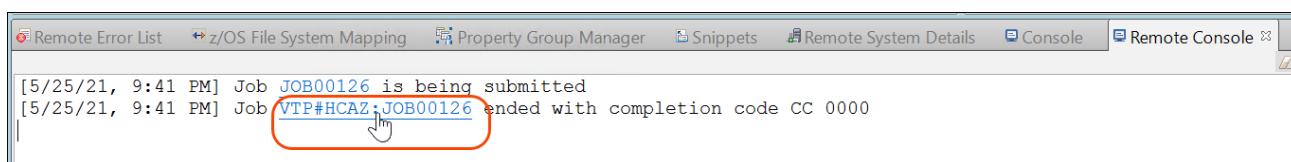
6.1.4 ► Click **Notify** to be notified when the execution is complete.



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

The completion code must be 00.

► Once the execution ends, **click on the link **VTP#HCAZ:JOB00xxx**** (where 00xxx is a number) to point to the Job output. This number varies depending on z/OS.



6.1..6 ► Under **Remote Systems** view expand **VTP#HCAZ:JOB00xxx** and **double click** **RUNNER:BZUMSG** step.

6.1..7 ► Scroll down the report displayed and you will see the program **Hcipdb01** returned data matches what you have recorded originally. **No errors any longer**

The screenshot shows the IBM Z VTP interface. On the left is a code editor window titled 'VTP#HCAZ.jcl' containing a spool file for job 'JOB00126'. The spool file shows various CICS transactions (BZUP401I, BZUP402I) and DB2 statements. A red box highlights the last few lines of the spool file, which end with 'BZUP002I FINISHED EXECUTION RC=00'. On the right is the 'Remote Systems' tree view under 'Team Artifacts'. It shows a hierarchy including 'web', 'websphere', 'Z22C', 'My Favorites', 'z/OS UNIX Shells', 'MVS Files', 'TSO Commands', 'JES', and 'Retrieved Jobs'. Under 'Retrieved Jobs', there are entries for 'VTP#HCAZ:JOB00126 [CC 0000]' and 'VTP#HCAZ:JOB00127 [CC 00041]'. A red arrow points from the 'Retrieved Jobs' section towards the spool file window.

6.1..8 ► Scroll down to the bottom and you will see that the execution finished with **RC=00**. That means that all data captured on the recording matches the execution.

This screenshot is similar to the previous one but shows a different spool file for job 'JOB00269'. The spool file content is identical to the previous one, ending with 'BZUP002I FINISHED EXECUTION RC=00'. A red arrow points to this line. The 'Remote Systems' tree view on the right shows the same hierarchy and retrieved jobs as the previous screenshot, with a red box highlighting the 'RUNNER:BZUMSG' entry under 'Retrieved Jobs'.

6.1..9 ► Use **Ctrl + Shift + F4** to close all opened editors.

Notice that this capability allows you to invoke Z VTP execution using pipelines like Jenkins.

What have you done so far?

On Section 1 -You executed the CICS transaction **HCAZ** and using VTP you recorded a simple interaction with the *Health Care application*. The recorded data is saved on a Z/OS dataset..

On Section 2 -You executed the VTP JCL that execute the sequence recorded. Since you made no changes the return code must be 00.



On Section 3 -You modified the COBOL program and introduced a bug. The program modified was compiled and DB2 bind was done using the DBB User Build capability. Again the VTP JCL is submitted for batch execution and we can verify the bug on the batch output execution.

On Section 4 -You execute a CICS NEWCOPY and use the HCAZ transaction again to verify the bug introduced by your COBOL modified program.

On Section 5 -You fix the bug introduced and rebuild the program using IDz.

On Section 6 -You run again the VTP JCL and verify that the bug is fixed. No more data mismatching from the original VTP recording.

Congratulations! You have completed the Lab 9.

LAB 10 – (OPTIONAL) Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline (60 minutes)

Updated Sept 02, 2021 (Regi) Created by Mathieu Dalbin , Regi Barosa, Ronnie Geraghty and Wilbert Kho

GitLab is an integrated DevOps platform, which fundamentally transforms the way Development, Security, and Ops teams collaborate to build and deploy software. From development to production, GitLab helps teams optimize the development cycle by decreasing time to market and increasing developer productivity through the implementation of DevOps best practice. GitLab enables collaboration by providing features to manage issues and milestones through dashboards. These features allow the team to organize their tasks into groups, projects, epics and issue boards.

This lab will take you through the steps of using GitLab CI along with DBB, ZUnit and UrbanCode Deploy (UCD) on z/OS.

On this lab you will fix a bug of an existing COBOL/CICS application stored in GitLab.

You will use **IDz** to change the code and perform a personal test for later delivery and commit to *GitLab* and then use *GitLab CI* for the final build and continuous delivery.

The updated code will be deployed to CICS using *UrbanCode Deploy* (UCD)

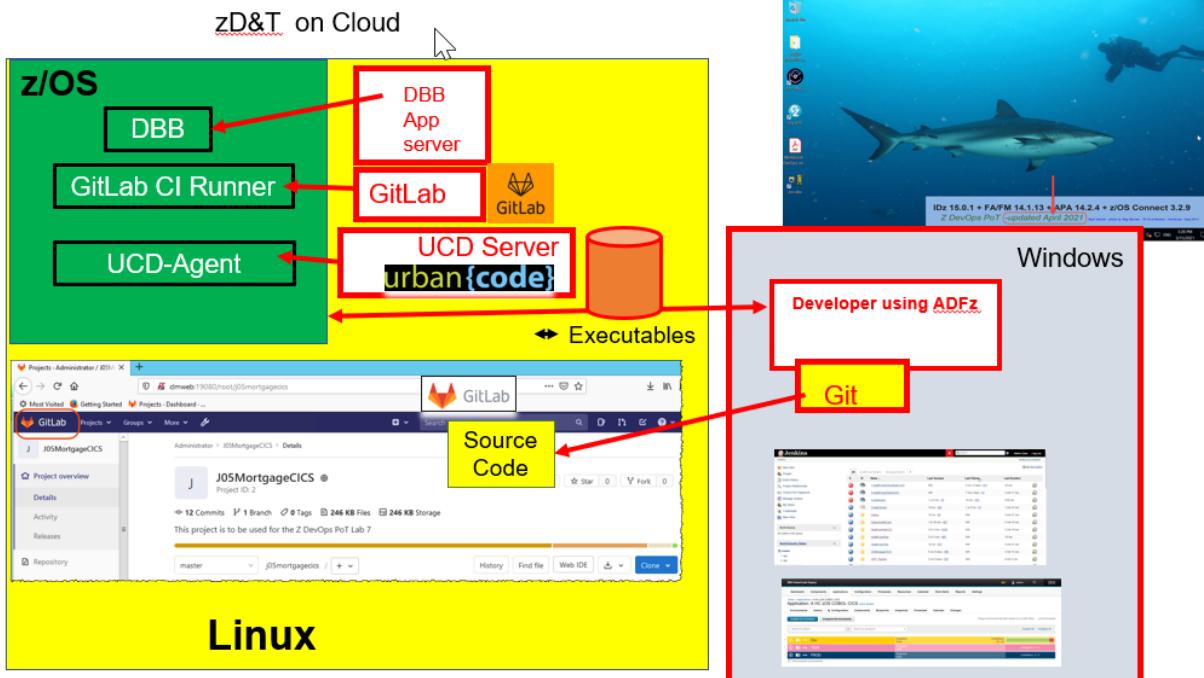
The process would be similar for a PL/I program or IMS instead of CICS.

The environment used on this Lab is pictured below.

Note that we have few “servers” running on Linux like UCD server, Jenkins, DBB Application Server. Also the **GitLab Repository** is on Linux.

The z/OS has many other running tasks including **CICS**, **DB2**, **DBB code** and agents that interact with the Linux servers.

Topology used on the labs



Overview of development tasks

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

1. **Review the GitLab issue and verify the bug using the 3270 terminal**
 - You will start GitLab, verify the issue and using a 3270 emulation execute a transaction named **SSC1** to become familiar with the Application that you intend to modify.
 - When using customer number 1 notice that the DOB is 0000-00-00. This is a bug that needs to be fixed
2. **Load the source code from GitLab to the local IDz workspace**
 - using IDz you will clone the GitLab loading all COBOL source code to your windows client.
3. **Use IDz to run the zUnit test case and verify the error**
 - Running zUnit you will verify that the DOB is incorrect,
4. **Modify the COBOL/CICS/DB2 program that has the bug using IDz.**
 - Using IDz you will modify the COBOL program LGICDB01 to fix the bug.
5. **Use IDz DBB User Build to compile/link and run the zUnit**
 - You will compile and link the modified code using the *DBB User Build Function*. When complete you will run zUnit generated test case and verify that the bug is fixed.
6. **Push and Commit the changed code to GitLab .**
 - You will commit the changes to *Git*. This will initiate the GitLab CI pipeline
7. **Verify the GitLab CI Build execution.**
 - You will use Gitlab to build the new changed code, run zUnit and push the executables to be deployed using *UCD* .
8. **Verify the GitLab CI Package and Deploy to UCD and test the CICS transaction again using 3270**
 - You will verify the results after the final deploy to *C/CS* using *UCD*

What is Git and DBB ?

Git is an open Source Code Management tool that is very popular in the distributed world.

In early 2017, Rocket Software ported Git into the mainframe – with the necessary checks to handle EBCDIC to UTF-8 conversions and vice-versa.

In Q3 2017, IBM released an Open Beta of **Dependency Based Build (DBB)**. DBB provides a build tool that provides the build framework, dependency understanding, and tracking for builds run on z/OS. This build system is not dependent on any SCM or Continuous integration automation tool. In this lab, we use DBB to build our z/OS COBOL source code which resides in the distributed Git Repositories.

DBB is part of IBM Developer for Z Systems EE (Enterprise Edition) or ADFz and was announced on March 13, 2018.

GitHub vs. Bitbucket vs. GitLab ?

More at: <https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab>

GitHub, **Bitbucket**, and **GitLab** are code collaboration and version control tools offering repository management. They each have their share of fans, though **GitHub** is by far the most used of the three.

Of the three, only **GitLab** is open source, though all three support open source projects.

GitHub offers free public repositories; **Bitbucket** also offers free private repositories; **GitLab** offers a Community Edition which is entirely free

Section 1. Review the GitLab issue and verify the bug using the 3270 terminal

You will access GitLab, verify the issue and using a 3270 emulator execute a transaction named SSC1 to become familiar with the Application that you intend to modify.

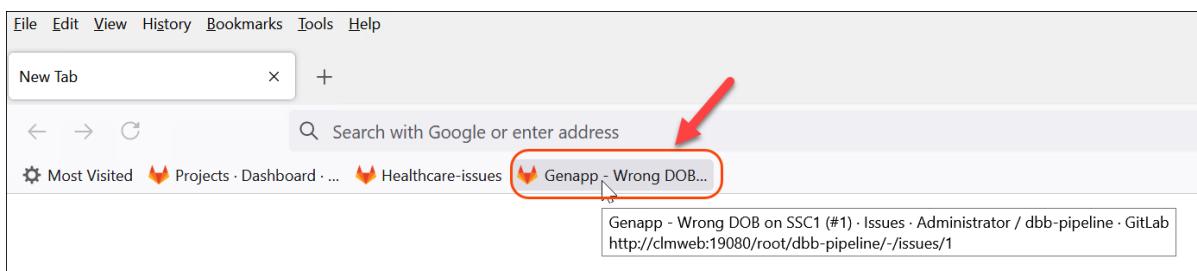
When using customer number 1 notice that the DOB is 0000-00-00. This is a bug that needs to be fixed.

1.0 Access GitLab and verify the issue.

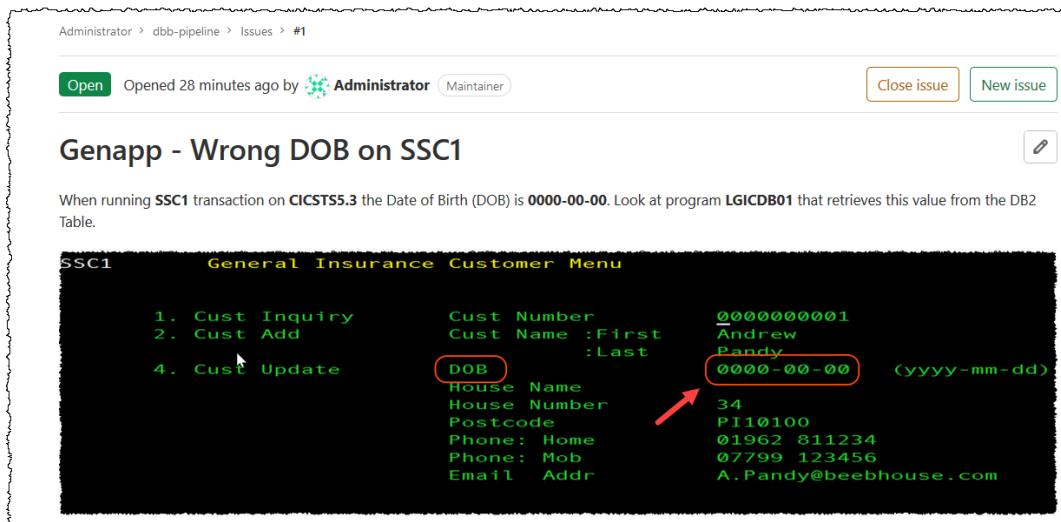
- 1.0.1 Start a browser by clicking the icon in the bottom of your screen



- 1.0.2 Click on the issue Genapp - Wrong DOB as below



This explains the current bug on the application



- 1.0.3 Minimize the web browser to go to the Windows desktop

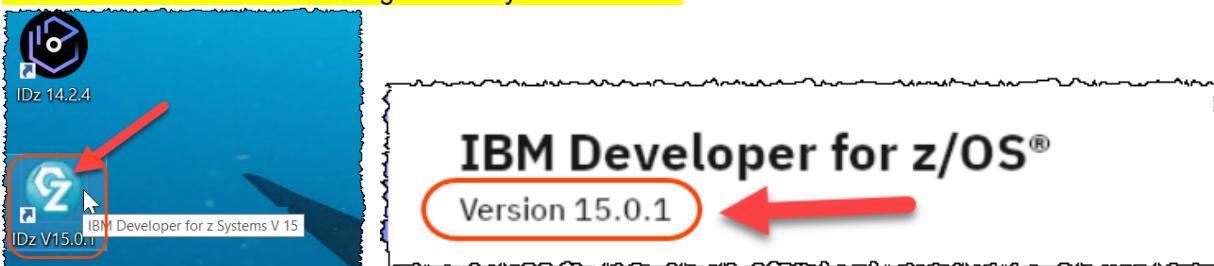
1.1 Connect to z/OS and emulate a CICS 3270 terminal

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.

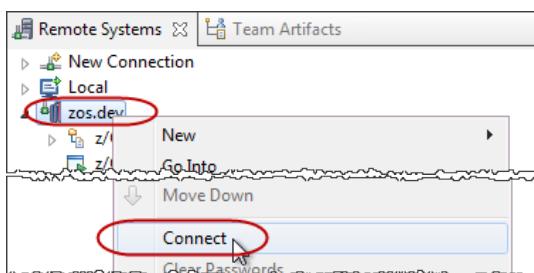
► Verify that the message indicates that it is Version **15.0.1**

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

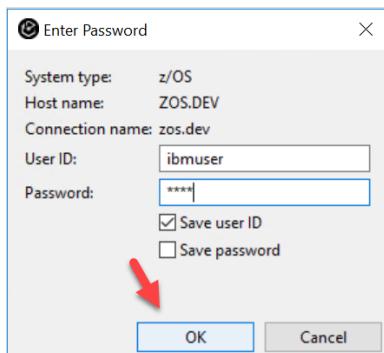


1.1.2 ► Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

1.1.3 ► Right click on **zos.dev** and select **Connect**.

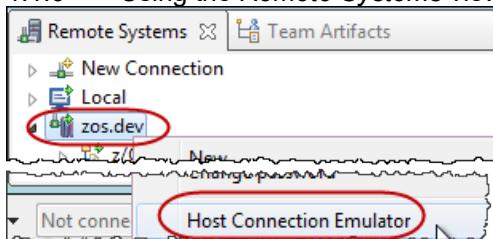


1.1.4 ► Type **ibmuser** as userid and **sys1** as password. Click **OK** to connect to z/OS.

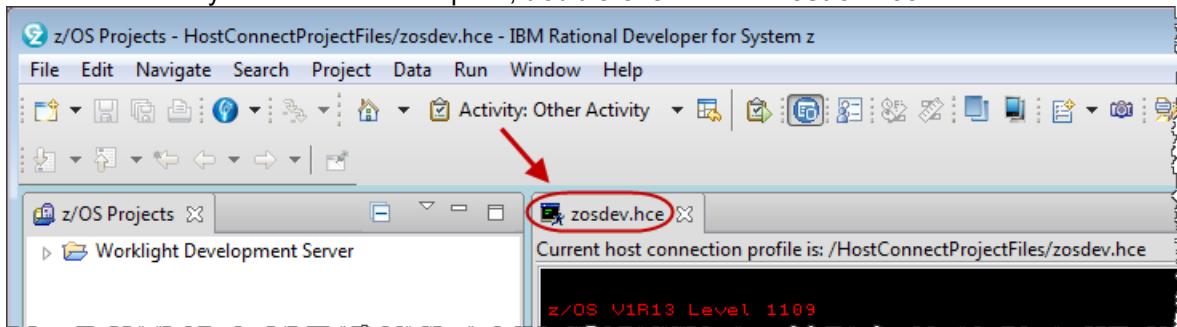


1.1.5 ► Wait until connection is complete (look at the bottom and left until the green bar disappears)

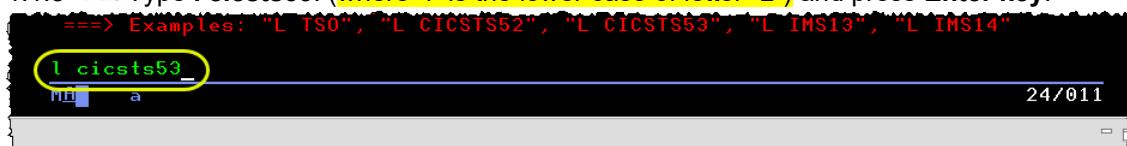
- 1.1.6 ► Using the **Remote Systems** view, right click on **zos.dev** and select **Host Connection Emulator**.



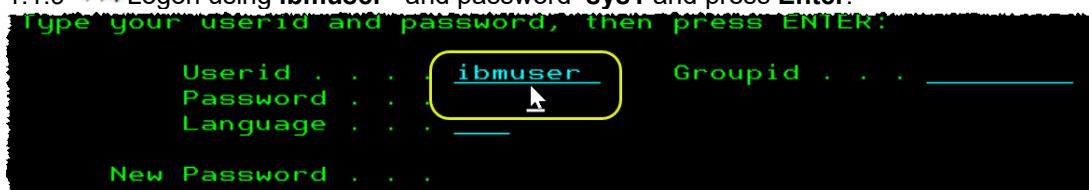
- 1.1.7 ► Since you will need more space, **double-click** on the **zosdev.hce** title



- 1.1.8 ► Type **I cicsts53.** (where "I" is the lower case of letter "L") and press **Enter key**.



- 1.1.9 ► Logon using **ibmuser** and password **sys1** and press **Enter**.



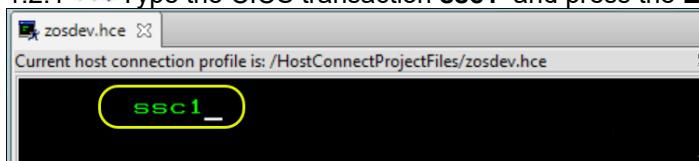
- 1.1.10 The sign-on message is displayed



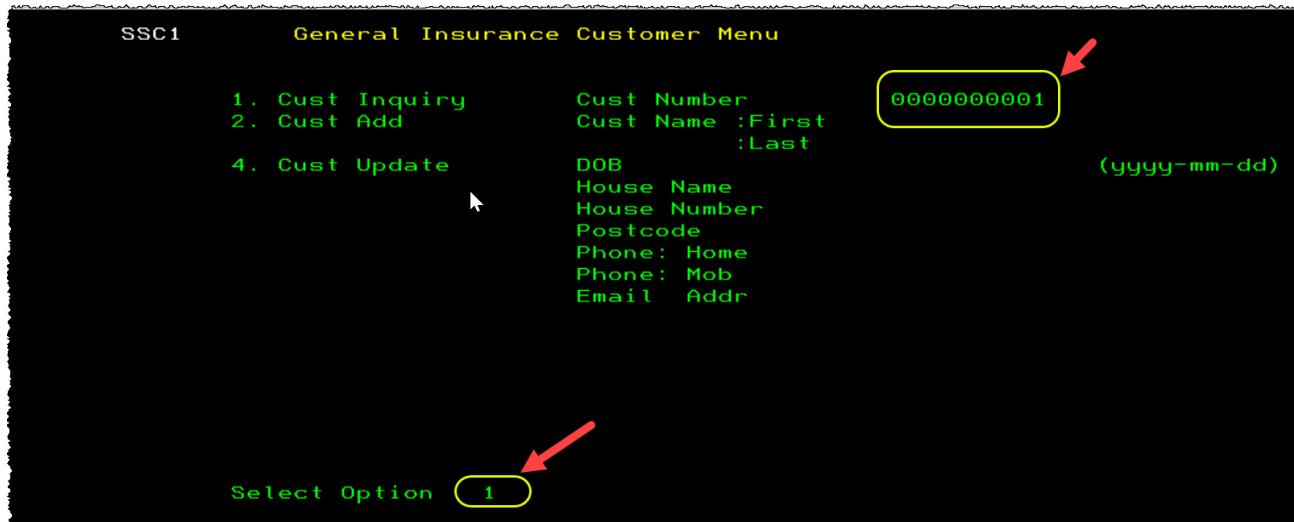
1.2 Run CICS transaction SSC1

You should now be in the z/OS CICS region named C/CSTS53. This is the CICS instance where you will make the program changes.

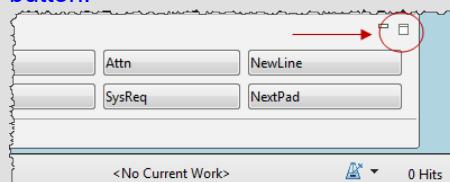
- 1.2.1 ► Type the CICS transaction **ssc1** and press the **Enter key**.



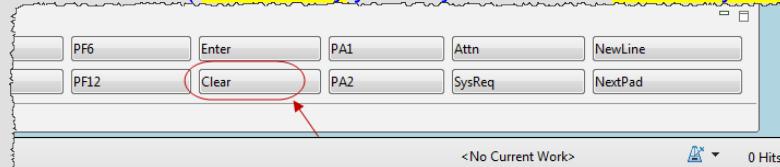
1.2.2 ► Move the mouse to last 0 and type 1, use the tab key and type 1 as **Select option** and press **Enter**



You may need to use the **clear** key. If the clear button is not displayed, look in the right lower corner, select this icon . This will display possible keys, including the clear button.



Click on Clear (If necessary you may also use the Reset key after clicking NextPad)

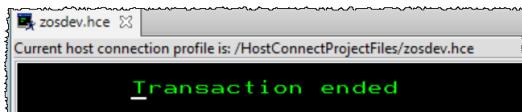


1.2.3 ► The data below is retrieved from a DB2 table .

Notice that **DOB** (*Date Of Birthday*) is **0000-00-00**. **This is a bug that you will need to fix.** The correct date should be a valid year, month and day..



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



1.2.5 ►| Close the terminal emulation clicking on →

Or pressing **CTRL + Shift + F4**.



What have you done so far?

You emulated a 3270 terminal using IBM Developer for System z.

You also executed the CICS transaction **SSC1** and verified an existing bug in the Genapp application. The objective here was to show the bug that you will correct.

Section 2 – Load the source code from GitLab to the local IDz workspace

You will load the COBOL code that is stored on GitLab (Linux) to your Windows client to be modified.

2.1 Cloning the Git Repository

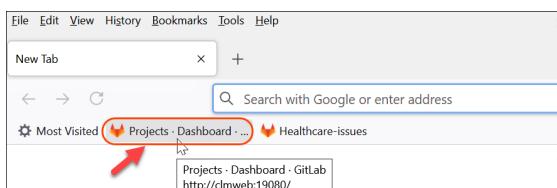
The Genapp Application used in this lab has its source code in the *GitLab Repository* that is on a Linux system. You must connect to the GitLab Repository and clone the Git project into the IDz. This brings the source code into your IDz workspace for edits and build.

2.1.1 Before cloning the Git repository, you should visualize the code stored at GitLab.

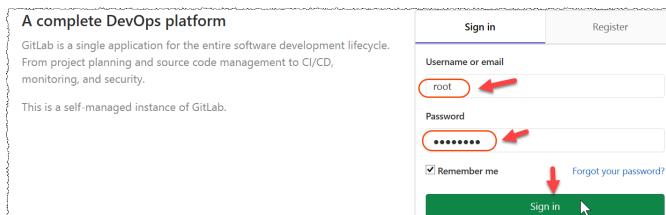
►| Go back to the web browser clicking in the icon in the bottom of your screen



2.1.2 ►| Click on this bookmark below to access **GitLab**. The URL used is <http://clmweb:19080/>



►| If asked for credentials use **root** and password: **zdtlinux**



2.1.3  Click on the hyperlink below to see the **dbb-pipeline** repository.



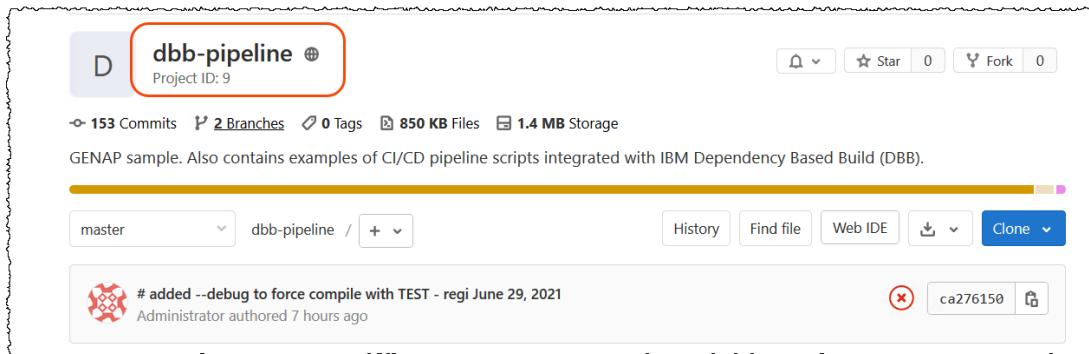
The screenshot shows the GitLab interface with the 'Projects' tab selected. There are two projects listed:

- Administrator / dbb-pipeline** (Maintainer): Description: GENAP sample. Also contains examples of CI/CD pipeline scripts integrated with IBM Dependency Based Build (DBB). Last updated 3 days ago.
- Administrator / dbb-zappbuild-1** (Maintainer): Description: A sample application built with IBM Z Application Build (ZAPPBUILD) and IBM Cloud Functions. Last updated 5 days ago.

A red arrow points to the first project, highlighting its name.

2.1.4  You can see the **dbb-pipeline** repository.

You may browse the content if you want. The source code to be used on this lab is there



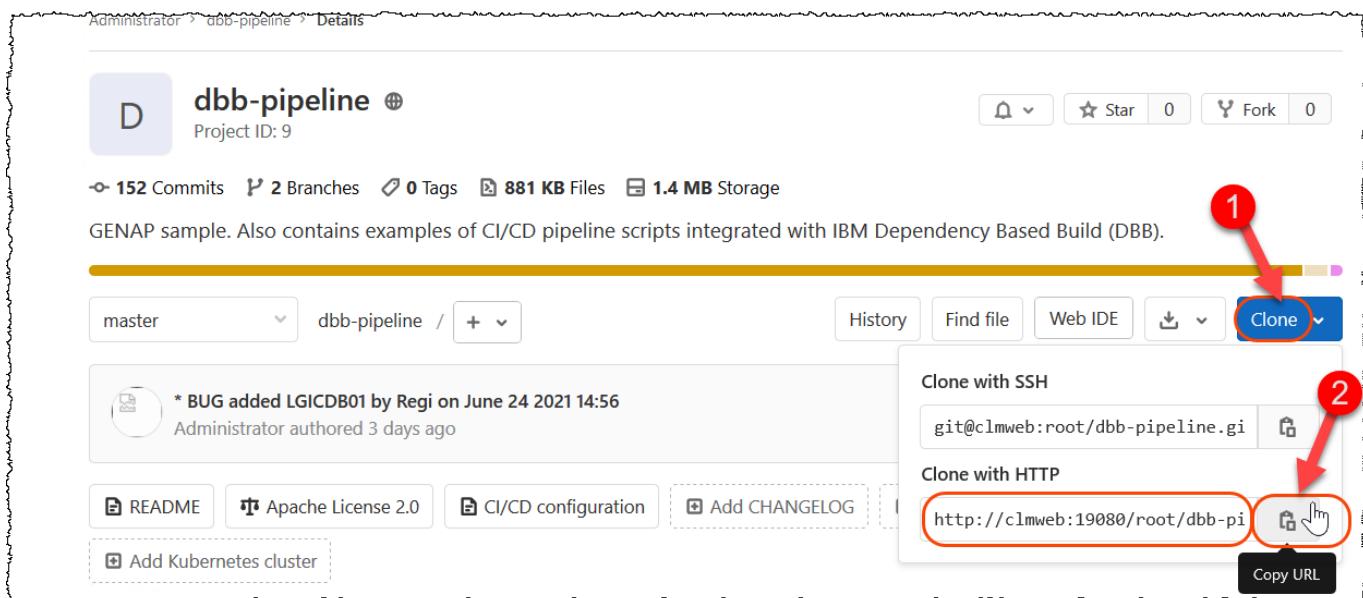
The screenshot shows the details page for the dbb-pipeline repository. Key information includes:

- Project ID: 9**
- 153 Commits**, **2 Branches**, **0 Tags**, **850 KB Files**, **1.4 MB Storage**
- Description: GENAP sample. Also contains examples of CI/CD pipeline scripts integrated with IBM Dependency Based Build (DBB).
- Branch dropdown: master
- Navigation buttons: History, Find file, Web IDE, Clone (blue button)
- Recent commit: # added --debug to force compile with TEST - regi June 29, 2021 (Administrator authored 7 hours ago)

2.1.5 In order to clone it at your window desktop you could use *HTTP* or *SSH*.

Since **SSH** is blocked in our environment we need to use **HTTP**.

 ① Click on **Clone** (the blue button) and ② in the icon  to **copy the URL**. This value will be kept in the windows clipboard.

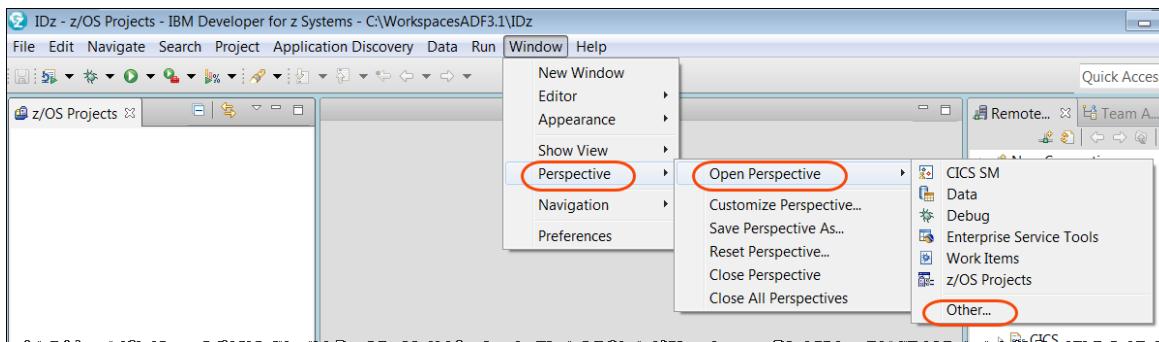


The screenshot shows the details page for the dbb-pipeline repository again. The 'Clone' button is highlighted with a red circle and labeled ①. A red arrow points from ① to the 'Clone' button. Below it, a 'Clone with SSH' field shows the URL `git@clmweb:root/dbb-pipeline.git`. To the right of this, a 'Clone with HTTP' field shows the URL `http://clmweb:19080/root/dbb-pipeline.git`. A red box highlights this URL, and a red arrow points from ② to the copy icon next to it. The 'Copy URL' button is also visible.

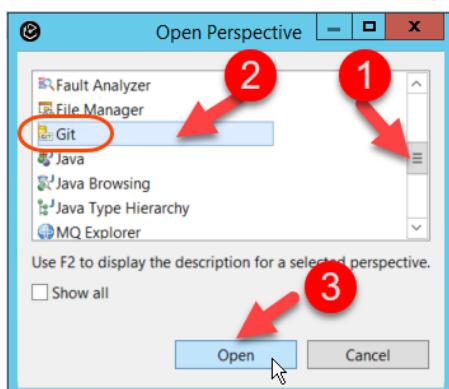
2.1.6 ► Go back to IDz using the icon that is on the base of your screen:



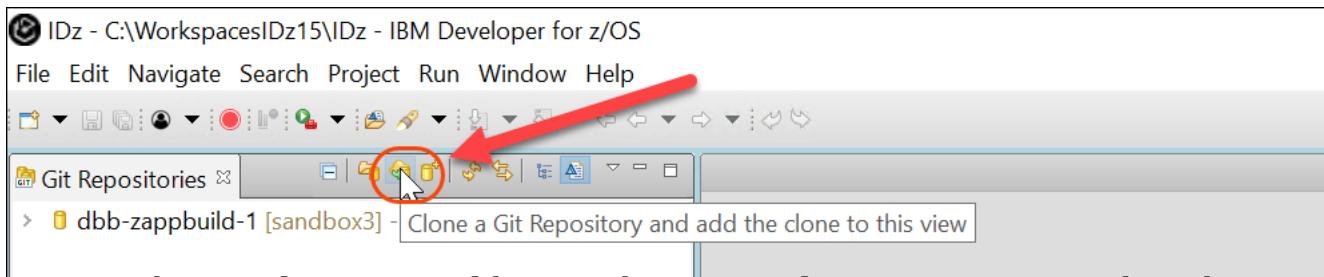
2.1.7 ► Open the **Git** perspective by selecting
Window > Perspective > Open Perspective > Other...



2.1.8 ► Select **Git** and click **Open**

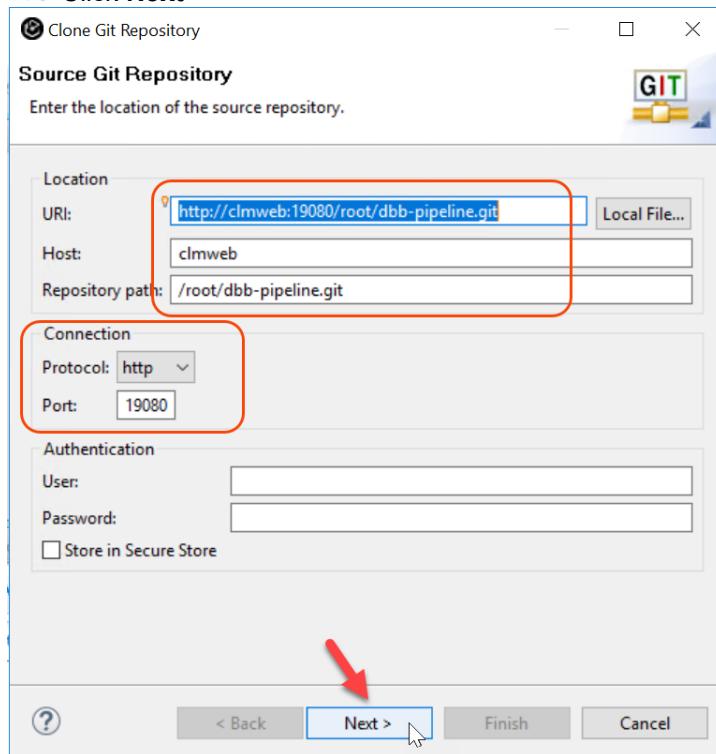


2.1.9 ► In the *Git Repositories* tab, click on the icon to 'Clone a Git repository'.



2.1.9 ► The values copied from the web page (copy URL) will be shown in the Clone Git Repository.
 Tip: In case you don't see it, got back to the page and copy it again (steps 2.1.1-2.1.5 above).

► Click Next

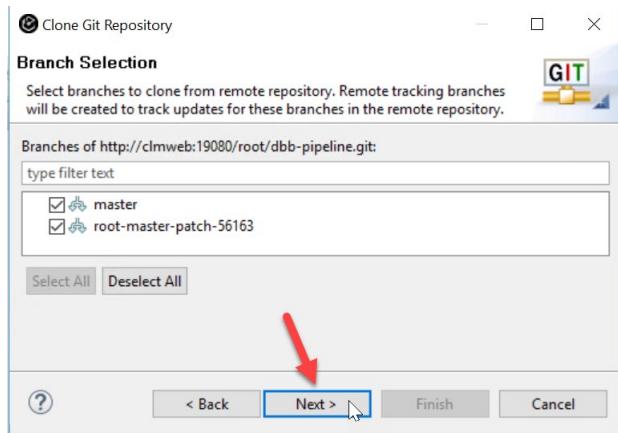


2.1.10 If a dialog asks for login the credentials are a user user: **root** and password **zdtlinux**

► Click Log in

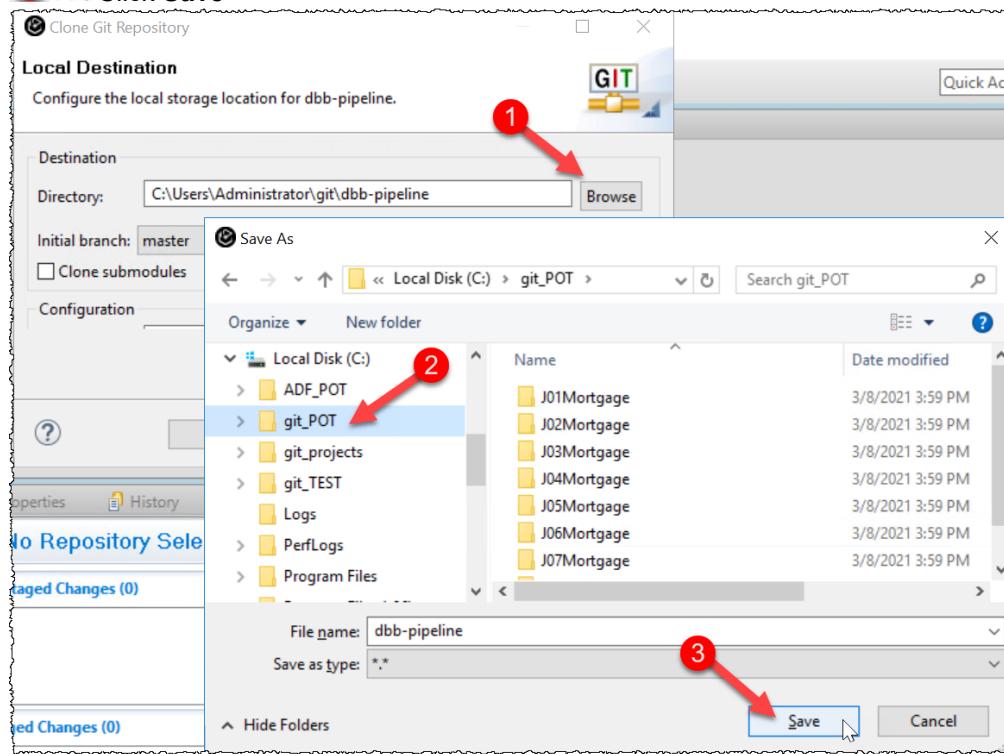


2.1.11 We will clone all branches. ► Click Next



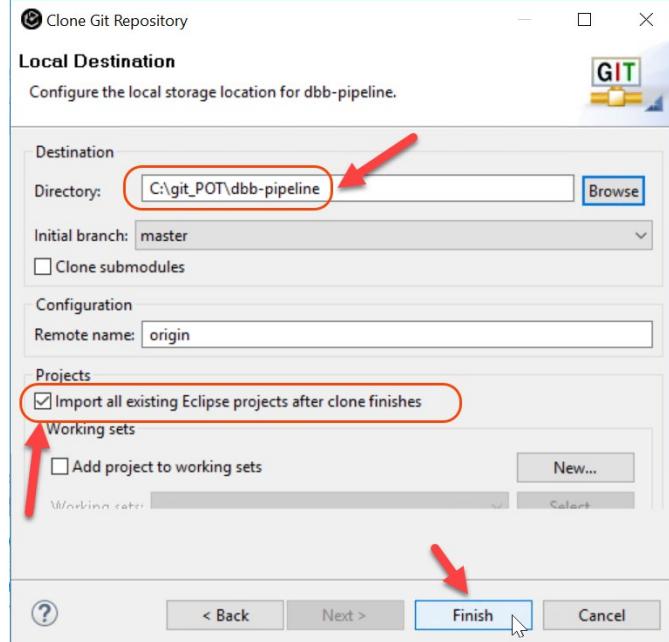
2.1.12 We will clone the repository on your local windows and import to be shown on IDz

- 1 ►► Use **Browse** button to select the directory **C:\git_POT** (on left).
- 2 ►► Click folder **git_POT** on left
- 3 ►► Click **Save**

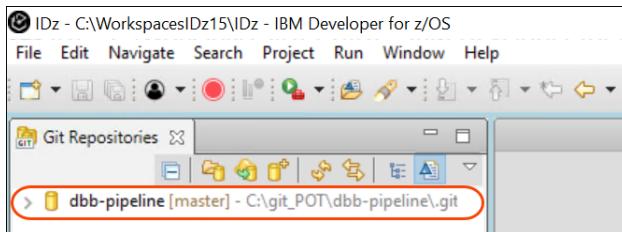


2.1.13 The *Directory* now should point to **C:\git_POT\dbb-pipeline**

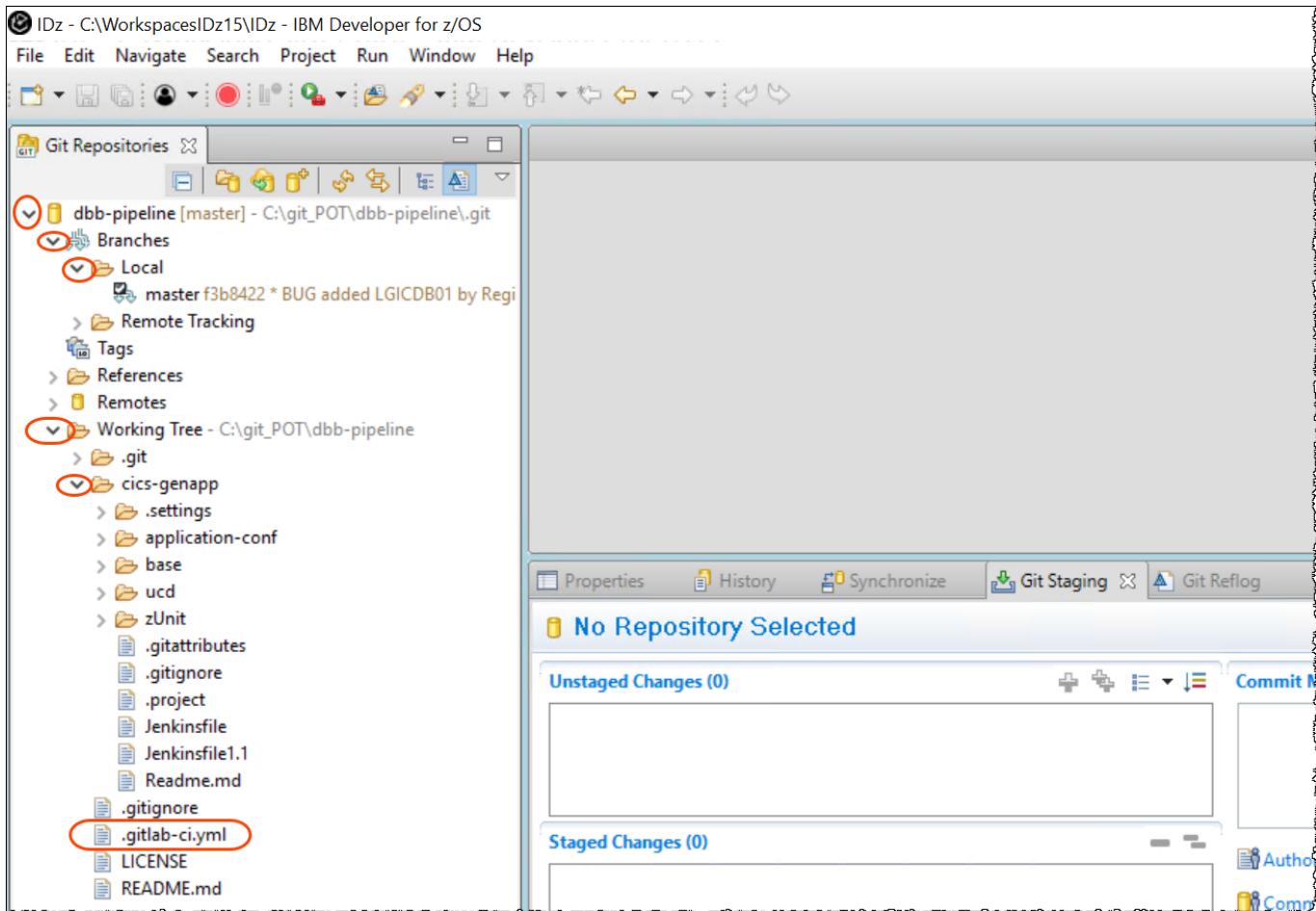
- Be sure that you selected **Import all existing Eclipse projects after clone finishes**
- Click **Finish**



2.1.14 The repository that has the Genapp Application will be cloned from the Remote master repository and will appear in the *Git Repositories* view.



2.1.15 **Expand the nodes** by left clicking on the icon as shown below:
Notice the **.gitlab-ci.yml** that will drive the deploy later

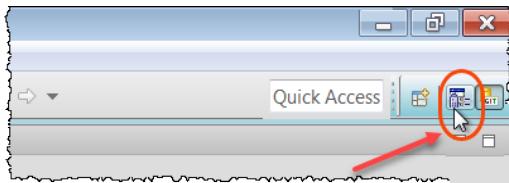


2.2 Verify the code cloned using z/OS projects perspective

The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

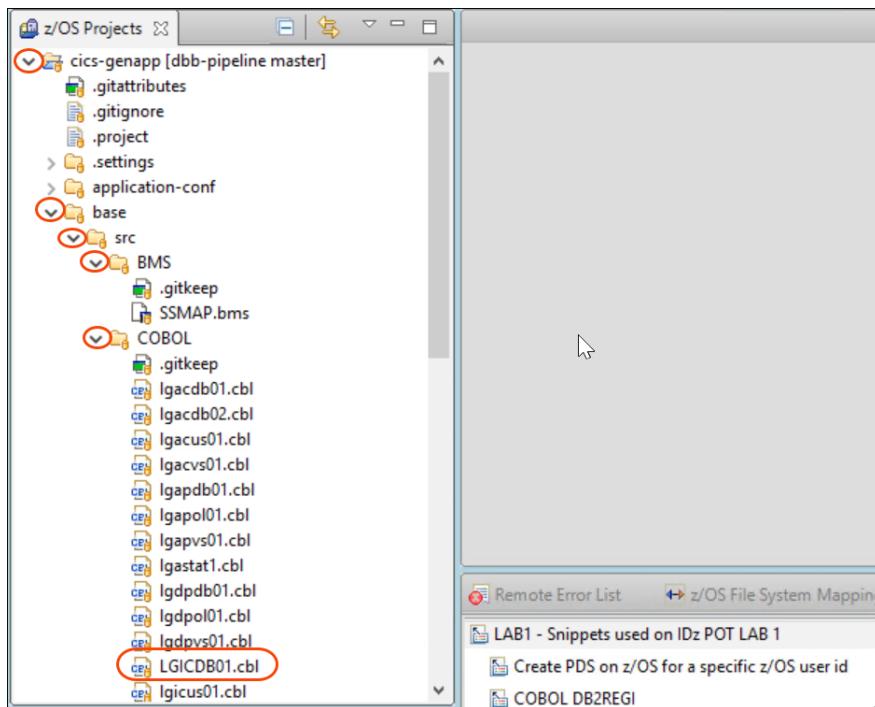
Notice that you have cloned the project at your local workstation but later you will need to connect to z/OS to be able to compile and build your programs.

2.2.1 ➡ Switch to the **z/OS Projects** perspective clicking on icon  on the top right corner



2.2.2 ➡ Expand the project **cics-genapp** clicking on icon  and see the source code loaded

Notice that IDz knows that this code is under a repository and the yellow decorator on the icon  indicates that. You will see later that the program that is causing the bug is **LGICDB01.cbl**



Section 3 – Use IDz to run the zUnit test case and verify the error

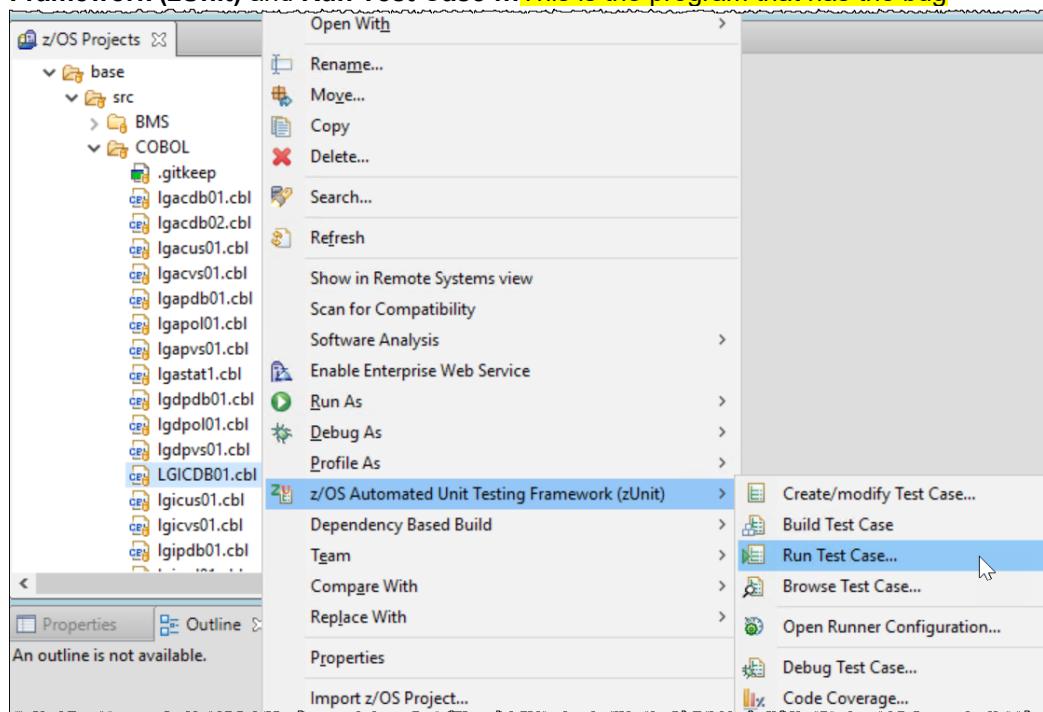
Running zUnit you will verify that the DOB is incorrect. A zUnit test case was created by other developer when the dialog was correct and the DOB date was displayed correct, For details how to create zUnit test cases you can see the zUnit labs provided.

3.1 Running the zUnit in batch

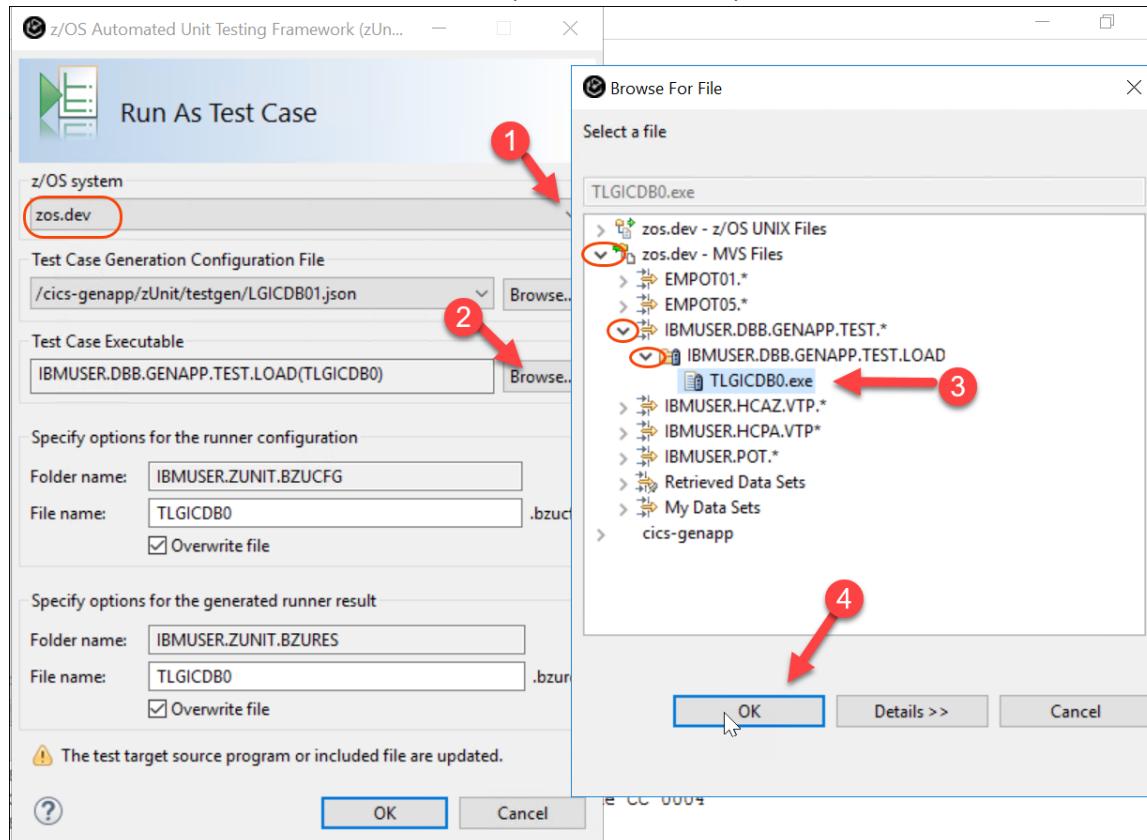
A zUnit test case is already available, and the developer will run it.

Notice that you can also run the test case using Debug option and code coverage. This would require re-build the test case generated with the option **-debug**.

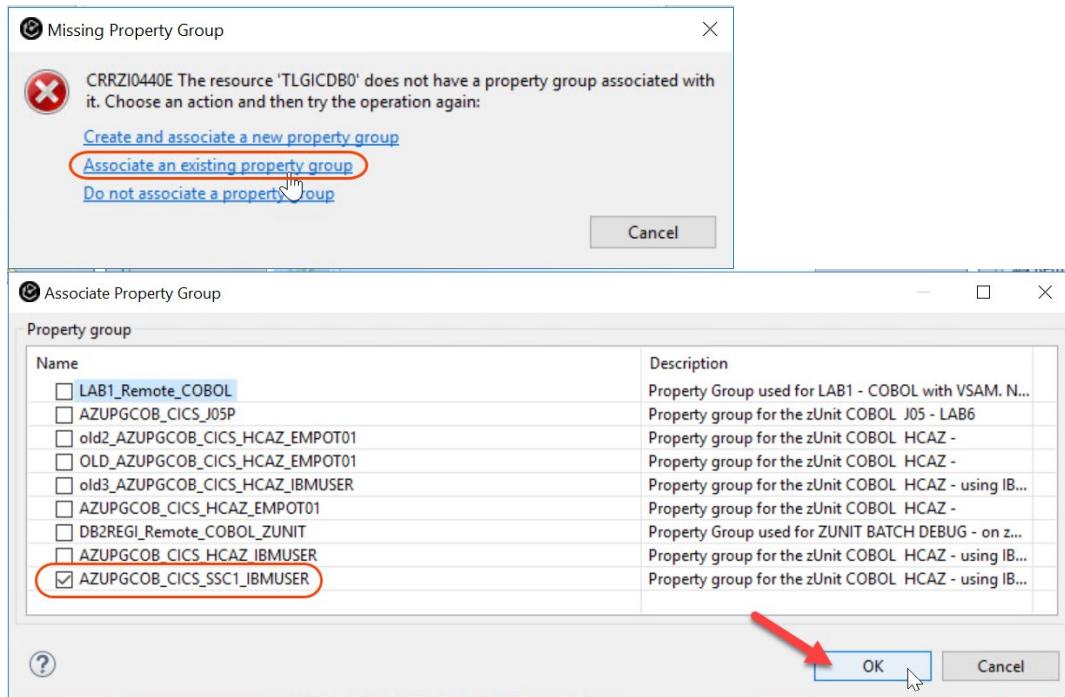
3.1.1 ► Using z/OS Projects view right click **LGICDB01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)** and **Run Test Case ...**
This is the program that has the bug



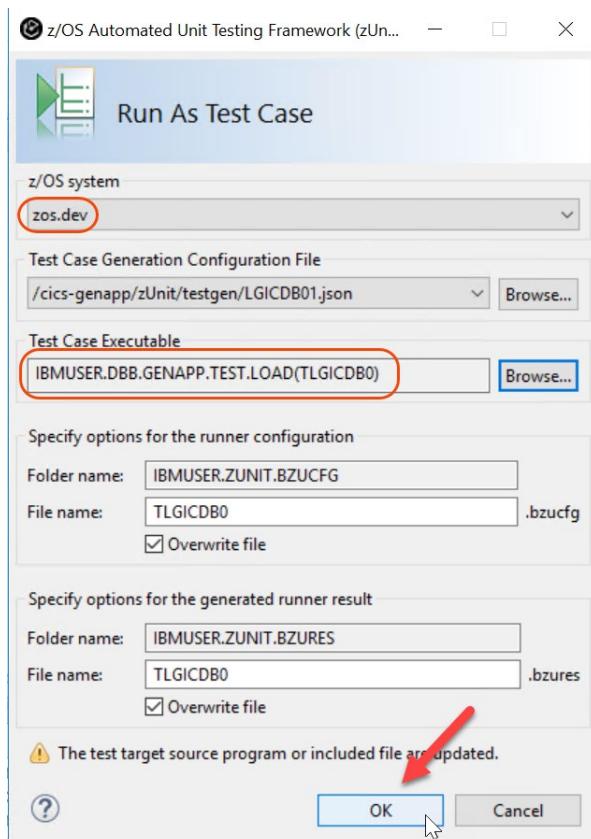
3.1.2 ► Select **zos.dev** for z/OS system, for **Test Case Executable** use the **Browse...** button and choose **IBMUSER.DB.BGENAPP.TEST.LOAD (TLGICDB0.exe)** and Click **OK**.



3.1.3 ► If the dialog below appears select **Associate an existing property group** point to **AZUPGCOB_CICS_SSC1_IBMUSER** and click **OK**

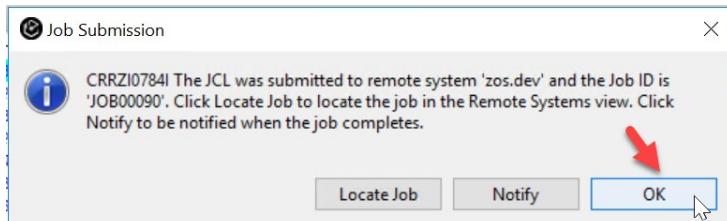


3.1.3 ► Be sure that your dialog values below match the screen below and click **OK**.



3.1.4 This dialog will send a batch job.

► Click **OK** for the Job Submission dialog



3.1.5 The Batch execution starts

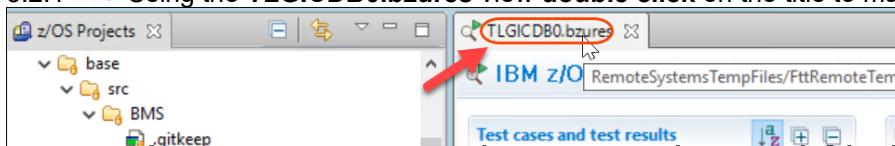
Interested in run in Debug Mode?
You can do this if the generated Test case program **TLGICDB0** is compiled with the **TEST** option. To do that use **DBB User Build** as explained on the step 5.1 to rebuild the program. Notice that the option **--debug** must be specified at the script dialog menu.

Script Parameters
Enter additional parameters to be used in the script command.
Create the Dependency Based Build command:
Option Value
--sourceDir \${SANDBOX}
--workDir \${LOGS}
--hlq \${HLQ}
--application cics-genapp
--debug

3.2 Verifying the zUnit Results

This Batch job can be seen in the JES but also a dialog is displayed with the results.

3.2.1 ► Using the **TLGICDB0.bzures** view double click on the title to maximize it



3.2.2. ► Expand the test case and verify that the expected value is **1950-07-11** but the current value is **0000-00-00**.

IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results

Test cases and test results
Use the tree view to navigate test cases and test results.
type filter text
zUnit runner results
Test case TLGICDB0
Test TEST2 (fail)
Exception message number (3000), Severity (4), Component (0000)

Exception details
If the result of the test is "fail" or "error", this section contains the details of the exception.
BZUPU00W ASSERT=COMPARE FAILED IN PROCEDURE DIVISION.
BZUP220I TEST RUN TEST2 REGISTERED FOR SUBSYS=CICS FILTER ON=LGICDB01 STUB C.
BZUP400I STARTING TRANSACTION=SSC1 USING PROGRAM=TEST2
BZUPU00W ASSERT=COMPARE FAILED IN PROCEDURE DIVISION.
BZUPT001 ITEM NAME=CA-DOB OF CA-CUSTOMER-REQUEST OF DFHCOMMAREA
BZUPT001 VALUE=0000-00-00
BZUPT001 EXPECTED VALUE=1950-07-11
BZUPT002 FINISHED EXECUTION RC=04

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

Section 4.Modify the COBOL/CICS/DB2 program that has the bug using IDz..

Using IDz you will fix the bug modifying the COBOL program that has the defect..

4.1 Edit and modify the code that send the message

4.1.1 ► Using z/OS Projects view double click **LGICDB01.cbl** under COBOL folder. This is the program that you will update. To make easier to see we made it UPPER case

```

z/OS Projects X
  src
    > BMS
    > COBOL
      .gitkeep
      lgacdb01.cbl
      lgacdb02.cbl
      lgacus01.cbl
      lgacvs01.cbl
      lgapdb01.cbl
      lgapol01.cbl
      lgapvps01.cbl
      lgastat1.cbl
      lgdpdb01.cbl
      lgdpol01.cbl
      lgdpvps01.cbl
      lgicu01.cbl
      lgicvs01.cbl
      lgipdb01.cbl
      LGICDB01.cbl
      lgicu01.cbl
      lgicvs01.cbl
      lgipdb01.cbl

```

The right pane shows the content of the LGICDB01.cbl file:

```

-----+*A-1-B-----2-----3-----4-----5-----6-----7-----+
1      * Regi to demonstrate a BUG introduced - look for %bug
2      * LGICDB01 changed on June 24 2021 14:56
3      * Inquire Customer TRX SSC1
4      * first prog invoked is LGICUS01
5      * Select customer details from DB2 table
6
7
8*
9
10*
11
12
13
14
15
16
17
18*
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213

```

4.1.2 ► Use **Ctrl + f** to find the "%bug" on line 203 . You will modify the line 203..

The left pane shows the COBOL source code with line 203 highlighted by a red circle and a red arrow pointing to it. The right pane shows the Find/Replace dialog with the following settings:

- Find:** %bug (highlighted by a red circle)
- Replace with:** (empty)
- Direction:** Forward (highlighted by a red circle)
- Scope:** All (highlighted by a red circle)
- Options:**
 - Case sensitive
 - Wrap search (highlighted by a red circle)
 - Whole word
 - Incremental
 - Regular expressions
- Buttons:**
 - Find** (highlighted by a red circle)
 - Replace/Find
 - Replace
 - Replace All
 - Close

3.1.3 ► Close the find dialog and modify the line 203 adding an * on column 7

Tip → Could right click on this line and select Source > Toggle Comment

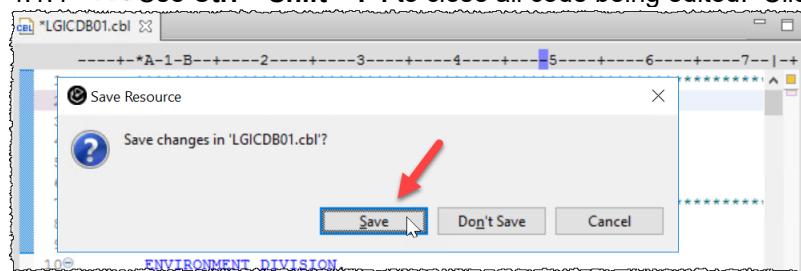
The screenshot shows a COBOL code editor window with the file 'LGICDB01.cbl'. Line 203 contains the instruction 'MOVE '0000-00-00' to CA-DOB'. An asterisk (*) has been added at the beginning of this line, circled in red. The code editor interface includes a status bar at the bottom.

```

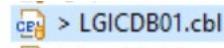
197      Evaluate SQLCODE
198      When 0
199      MOVE '00' TO CA-RETURN-CODE
200
201      *
202      * $bug The line below is the BUG > add * at column 7 to fix
203      * MOVE '0000-00-00' to CA-DOB
204      *
205      When 100
206      MOVE '01' TO CA-RETURN-CODE
207      When -913
208      MOVE '01' TO CA-RETURN-CODE

```

4.1.4 ► Use Ctrl + Shift + F4 to close all code being edited. Click Save to save the modified code.



Notice that since Git is managing this code a mark on the program that indicates a change

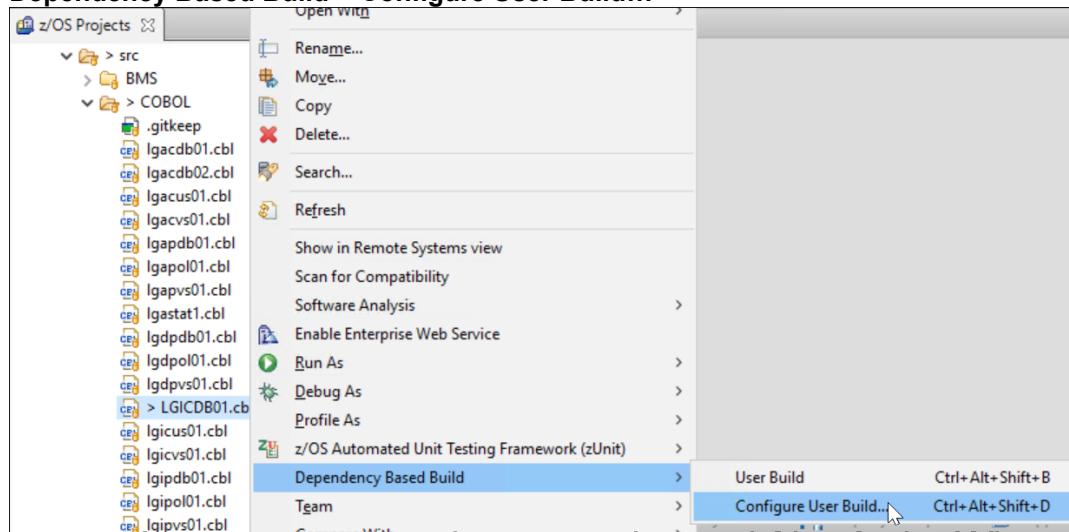


Section 5. Use IDz DBB User Build to compile/link and run the zUnit.

You will compile and link the modified code using the DBB User Build Function. When complete you will run zUnit generated test case and verify that the bug is fixed.

5.1 Using Dependency Based Building option

5.1.1 ► Under z/OS Projects view right click LGICDB01.cbl and select Dependency Based Build > Configure User Build...



5.1.2 ► Be sure that those values are assigned for the build.

4 On *build destination HLQ*: be sure that you are using **IBMUSER.DBB.GENAPP** (must type)

► Click **Next**

DBB User Build

Configure User Build Operation

Specify information for user build operation.

Select the z/OS system to use:

zos.dev

Select the build script to use:

\zAppBuild\build.groovy

Enter the build sandbox folder:

/var/dbb/work_gitlab

Enter the build destination HLQ:

IBMUSER.DBB.GENAPP

4



2

3



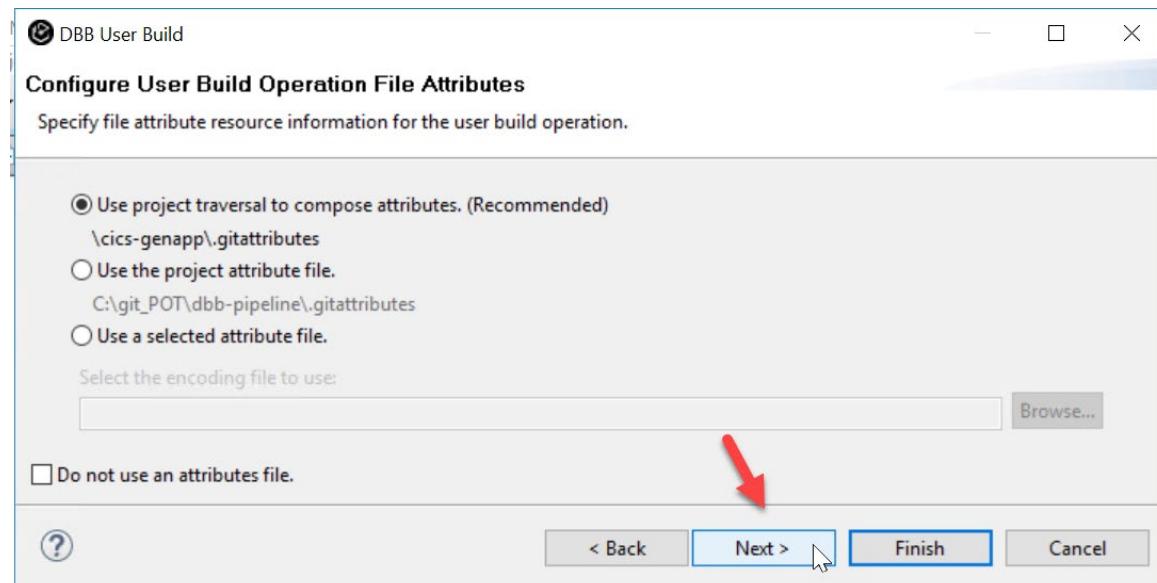
Next >

Finish

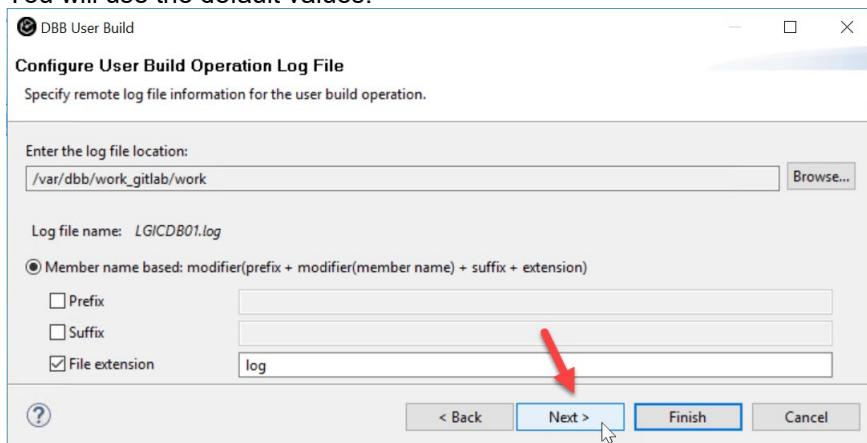
Cancel

5.1.3 ► On *Configure User Build Operation File Attributes* click **Next**

The **.gitattributes** have the information to translate from *UTF-8* to *EBCDIC* when moving the code to z/OS.



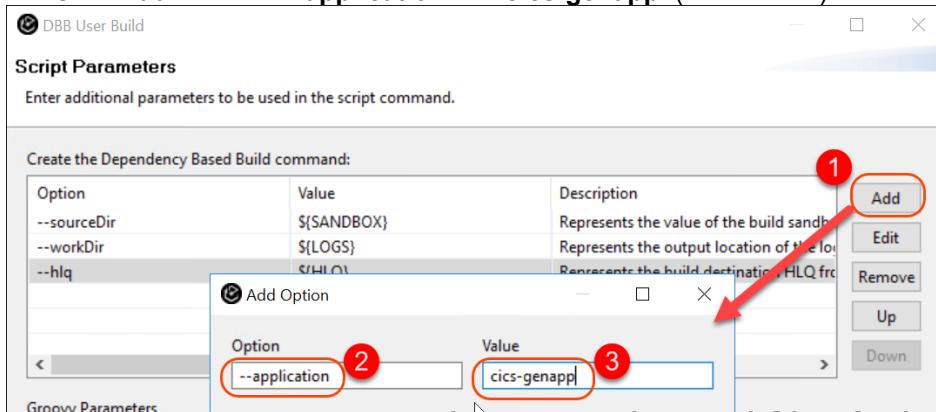
5.1.4 ► On *Configure User Build Operation Log File* click **Next**
 You will use the default values.



5.1.5 ► On dialog *Script Parameters*, if the “**--application**” and “**--debug**” are not there you must add
 ► Otherwise skip to 5.1.7

In case you need to add **--application**.

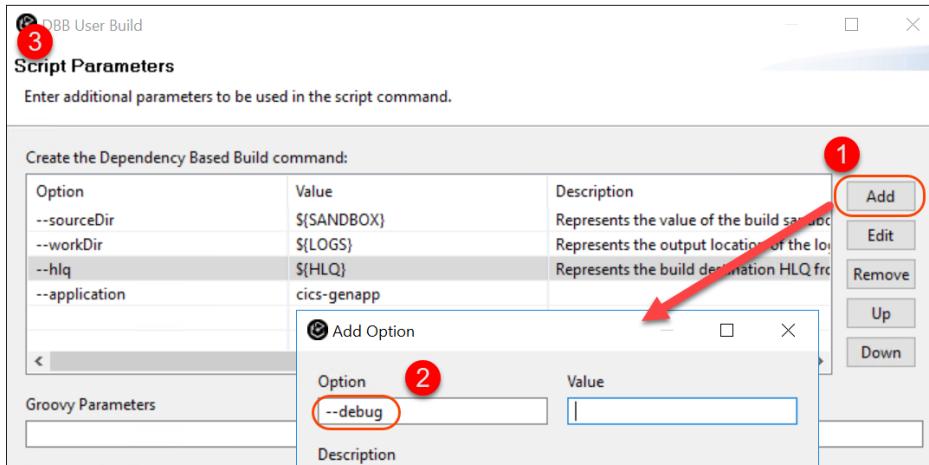
► Click **Add** to insert **--application** and **cics-genapp** (mixed case) and click **OK**



5.1.6 In case you need to add **--debug**.

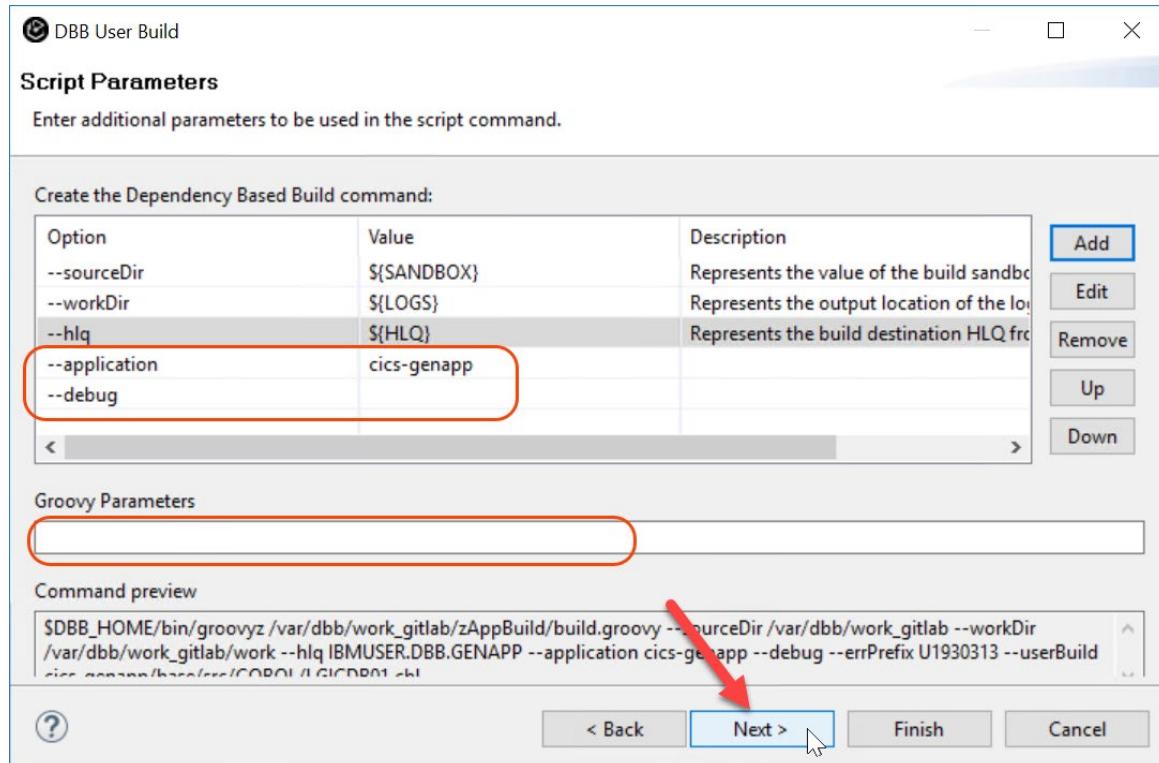
► Click **Add** to insert **--debug** and click **OK**

Verify that the options below were inserted. Notice that there is an “**--**” before each option.



5.1.7 The field **Groovy Parameters** must be empty

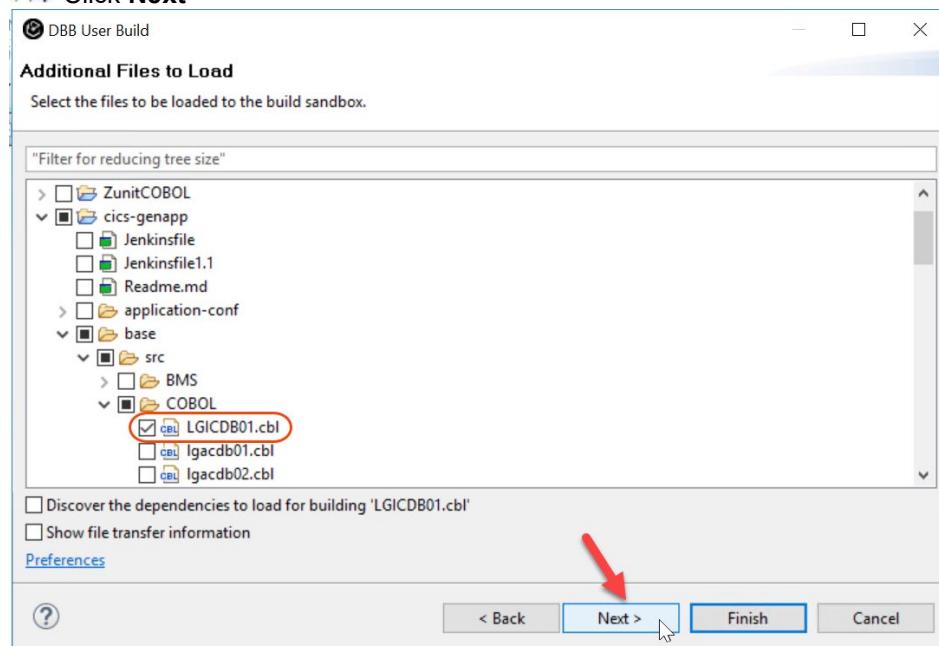
▶▶ Click **Next**



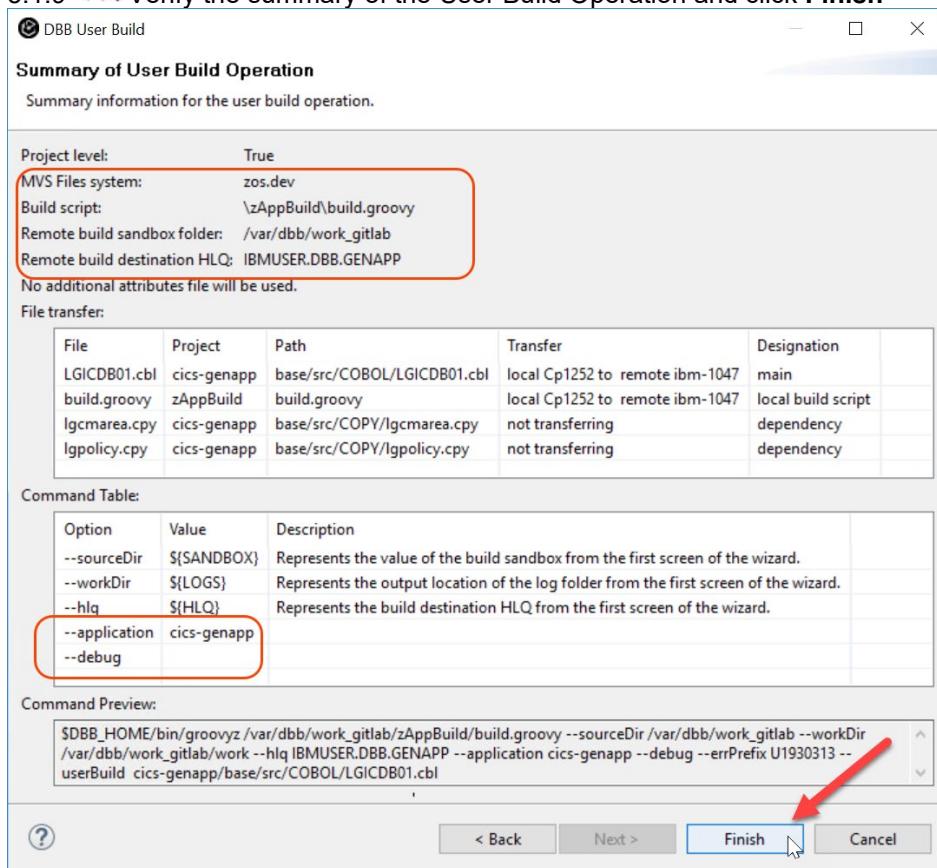
5.1.8 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

▶▶ Expand **cics-genapp** and **base/src/COBOL** to verify that the **LGICDB01.cbl** is selected

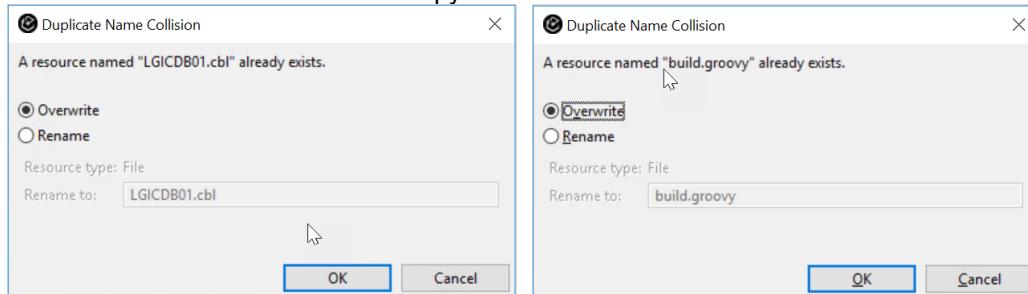
▶▶ Click **Next**



5.1.9 ► Verify the summary of the User Build Operation and click **Finish**



5.1.10 ► If the dialogs below pops up select **Overwrite** and click **OK**
Also select **Overwrite All** for the copybooks

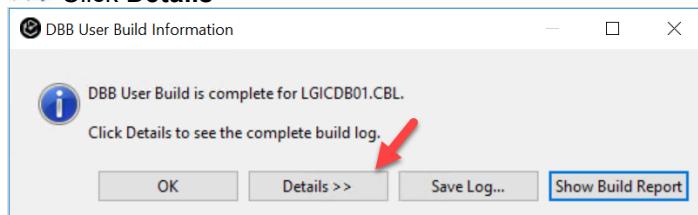


The DBB User build will be running and the messages will be shown at the Console view..

► Click on **Console** tab on lower right corner The execution will start, and this will take a while (2 Minutes) .

5.1.11 When complete you will have the message below:

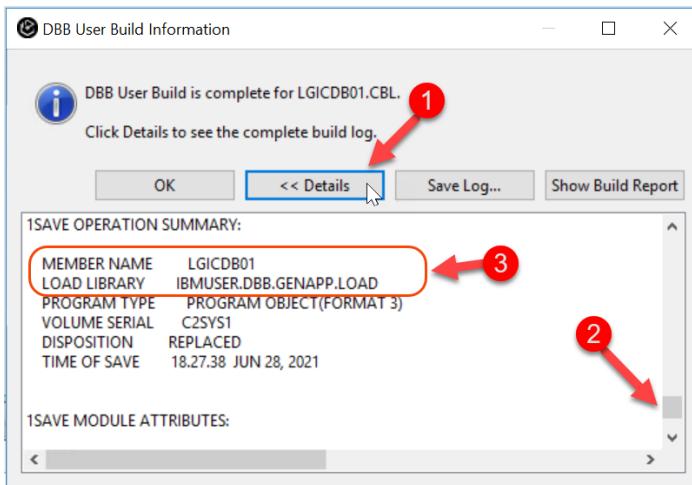
► Click **Details**



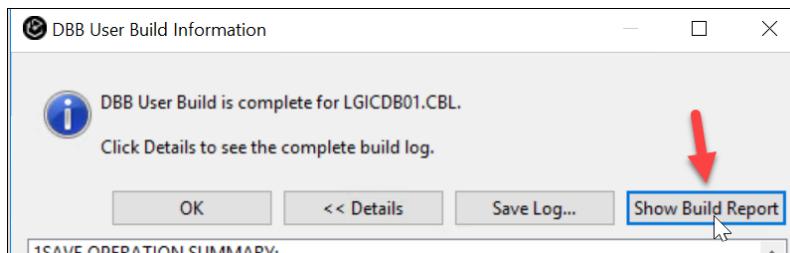
5.2 Verify the DBB User Build results

5.2.1 ► After clicking **Details >>** scroll down and verify that a new load module named **LGICDB01** was replaced on the dataset **IBMUSER.DB.B.GENAPP.LOAD**

► Click **OK** to close this dialog.



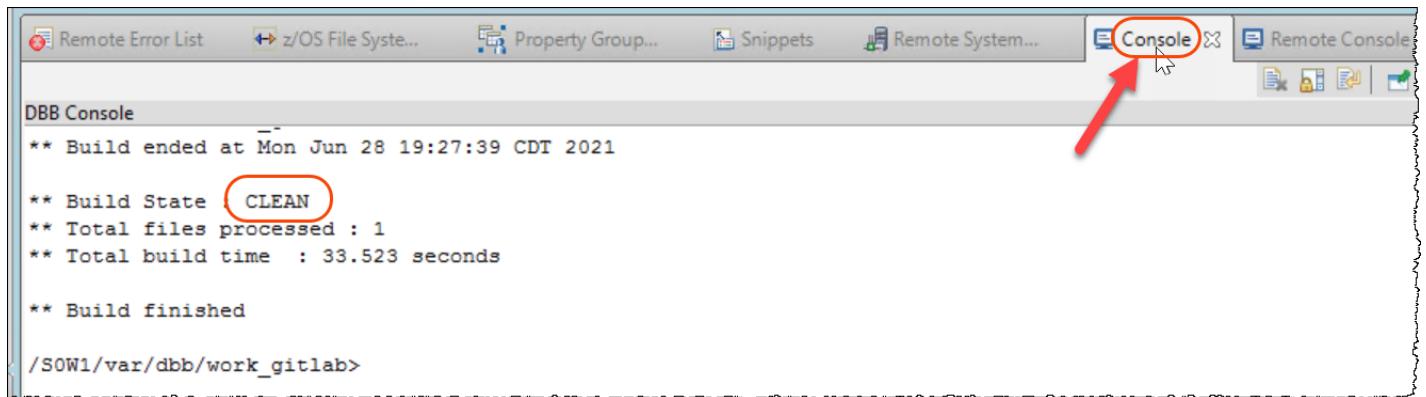
5.2.2 ► Click **Show Build Report** to see the report created:



File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
cics-genapp/base/src/COBOL/LGICDB01.cbl Show Dependencies	IGYCRCTL	4	IBMUSER.DB.B.GENAPP.COBOL (LGICDB01)	IBMUSER.DB.B.GENAPP.DBRM (LGICDB01)	DBRM	LGICDB01.log
	IEWBLINK	0		IBMUSER.DB.B.GENAPP.LOAD (LGICDB01)	LOAD	

This report will also be stored at the DBB Application Server

5.2.3 ► Close the dialogs and Verify at the Console view that the Build State should show a **CLEAN** build.



The screenshot shows the DBB Console window. The console output displays the following text:

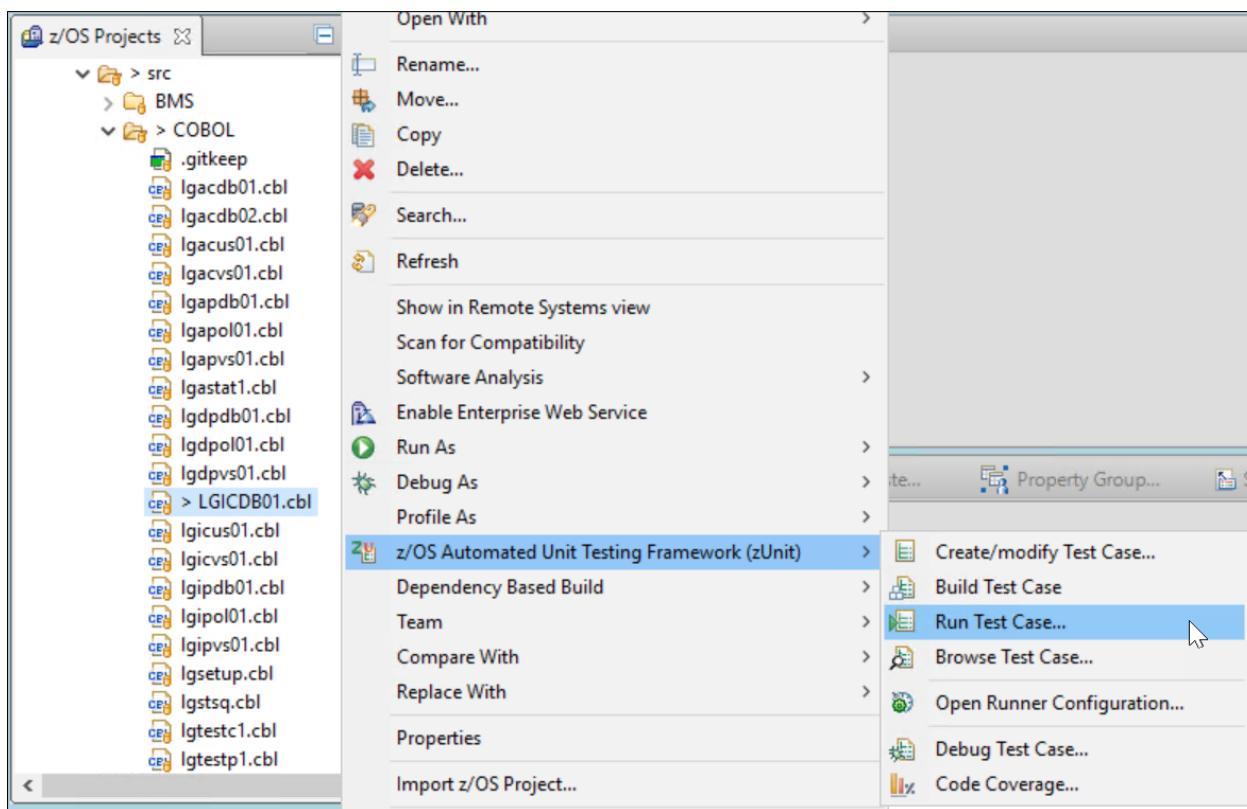
```
** Build ended at Mon Jun 28 19:27:39 CDT 2021
** Build State  CLEAN
** Total files processed : 1
** Total build time   : 33.523 seconds
** Build finished
/SOW1/var/dbb/work_gitlab>
```

A red arrow points to the 'Console' tab in the top right corner of the window, which is highlighted with a red circle. The word 'CLEAN' in the build state line is also circled in red.

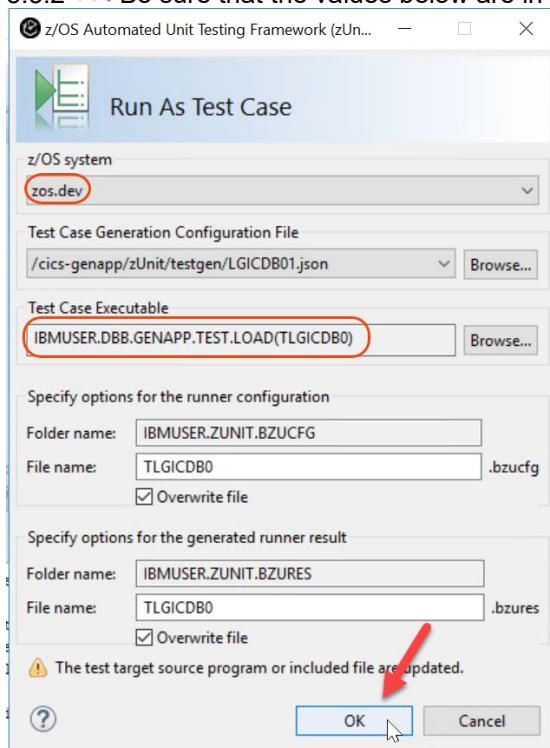
5.3 Running the zUnit again

You should now run the zUnit test case again to verify that the bug has been fixed.

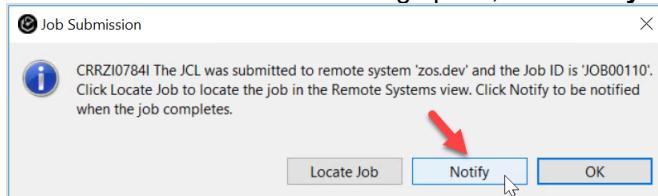
5.3.1 ► Right click on **LGICDB01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > . Run Test Case**



5.3.2 ➡ Be sure that the values below are in effect and click **OK**

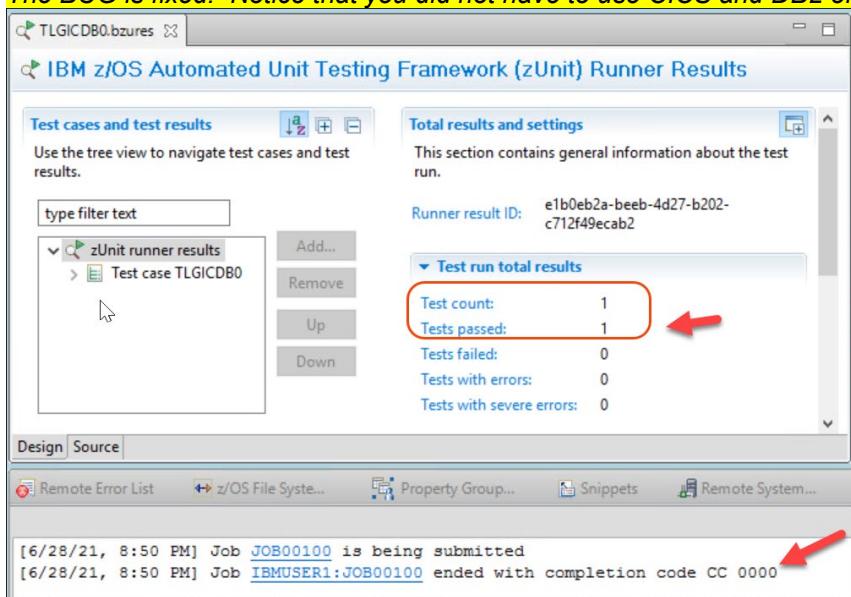


5.3.3 ➡ A Job Submission dialog opens, click **Notify** to be notified of job completion.



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

The BUG is fixed. Notice that you did not have to use CICS and DB2 environment to correct the bug.



5.3.5 ►| Use **Ctrl + Shift + F4** to close all opened editors.

Section 6. Push and Commit the changed code to GitLab.

Using IDz you will commit the changes you made to GitLab This will initiate the GitLab CI pipeline This will initiate the GitLab CI pipeline

6.1 Using IDz and Git perspective to compare the change versus original

6.1.1 ►| Switch to the **Git Perspective** selecting the icon on top and right

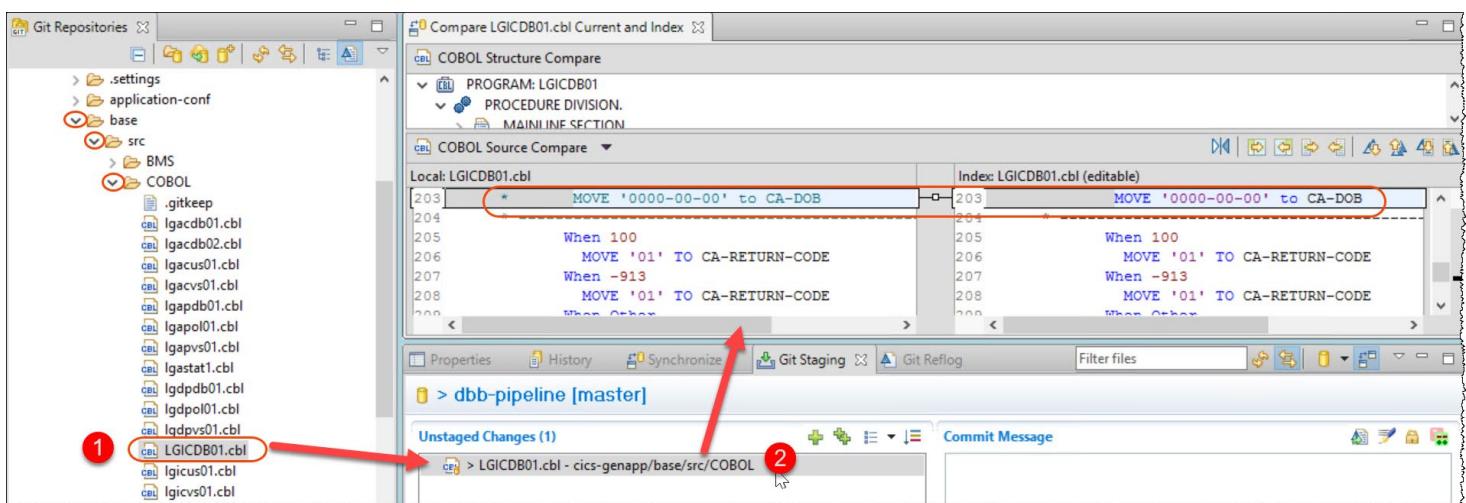


6.1.2 ►| ① Expand the **base/src/COBOL** folders and click on the program **LGICDB01.cbl** that you have modified.

Notice that the changed program is listed in the *Git Staging* view..

►| ② Double click on the **LGICDB01.cbl** that is listed under **Unstaged Changes** and noticed the comparison among the program that you updated versus the one that is in the Git repository.

You will need to commit the changes to the Git repository to make a final build later.

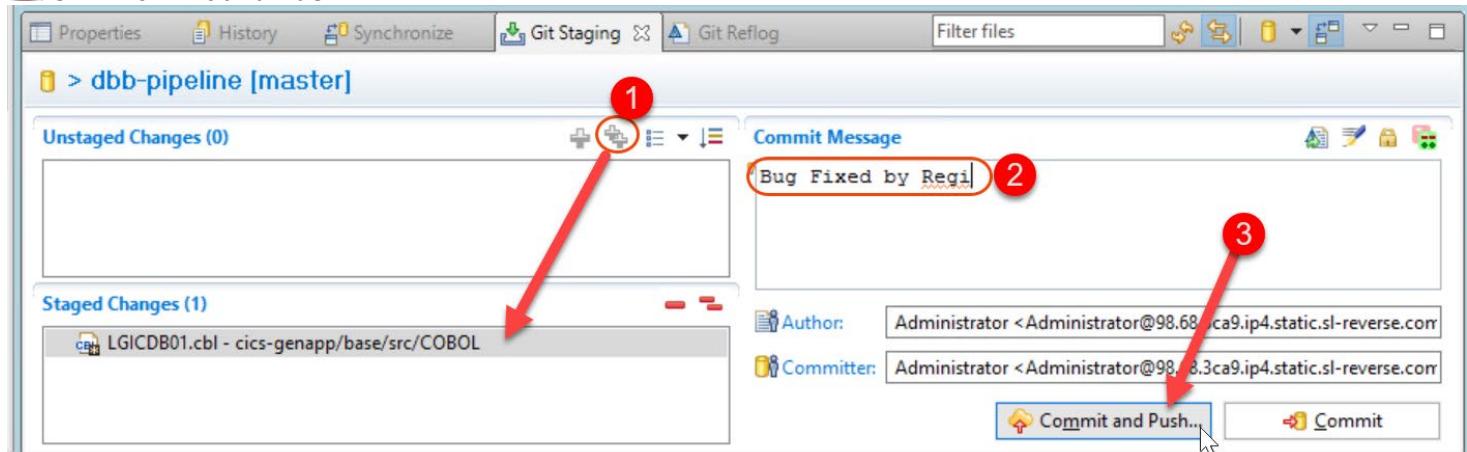


6.1.3 ►| Use **Ctrl + Shift + F4** to close the editors.

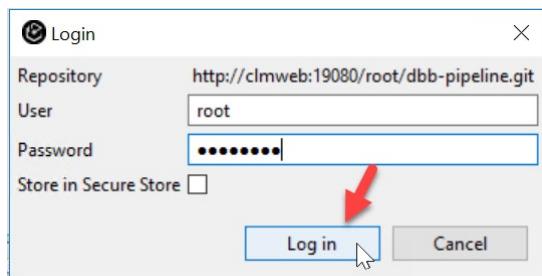
6.2 Push and Commit the changes to GitLab

6.2.1 ► Using Git Staging view, ,

- 1 Click on the icon 
- 2 Write a commit message like: **Bug Fixed by your name**
- 3 Click **Commit and Push ...**

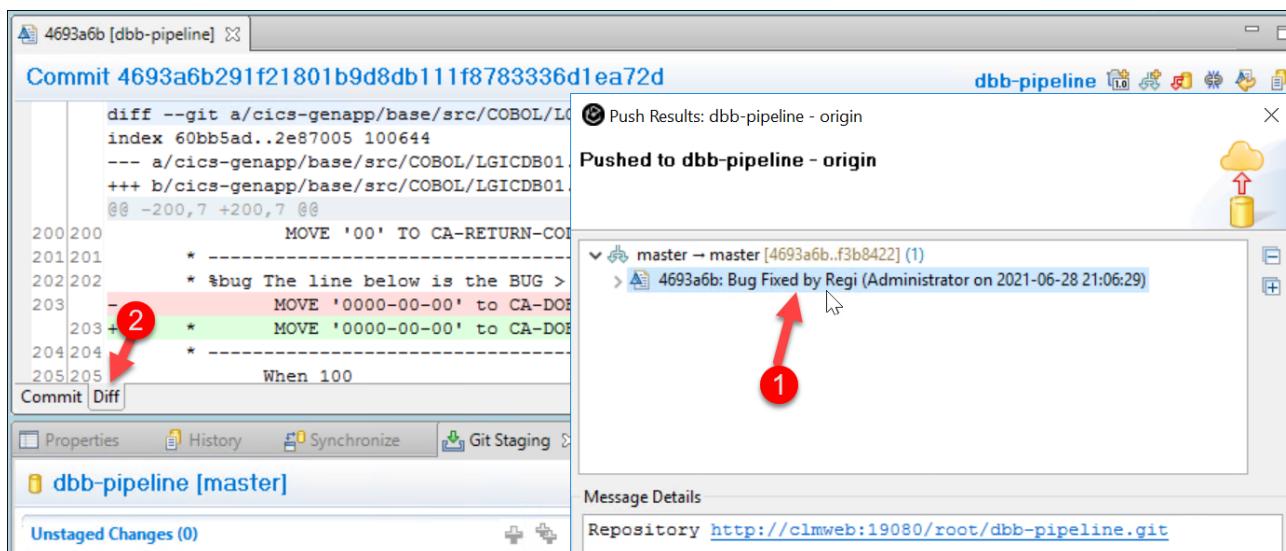


6.2.2 ► For login credentials use **root** and password **zdtlinux** and click **Log in**



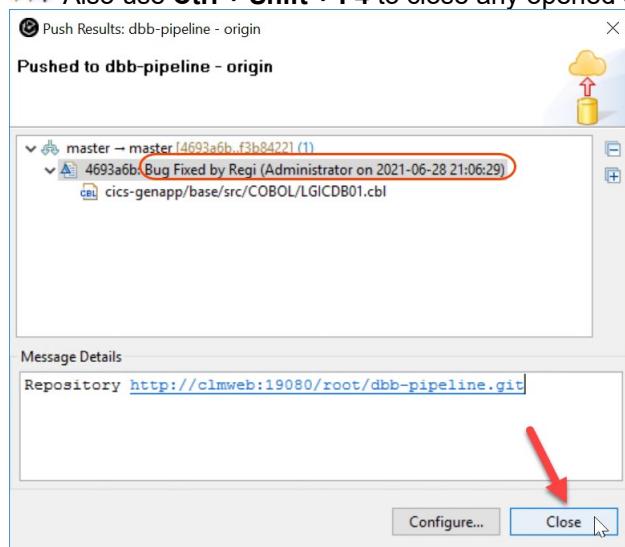
6.2.3 To see the changes you committed to Git:

- 1 Double click on **Git hash string** and 2 select **Diff** tab



6.2.4 ➡ Verify that the commit was successful and click **Close**

➡ Also use **Ctrl + Shift + F4** to close any opened editor



6.2.5 ➡ To see this changes in **GitLab** use the **Firefox browser** and do a **Refresh (F5)** (details how to get the link at 2.1.1).

The screenshot shows the GitLab project page for 'dbb-pipeline'. The project summary includes metrics: 153 Commits, 2 Branches, 0 Tags, 932 KB Files, and 1.5 MB Storage. Below the summary, a commit card for 'Bug Fixed by Regi' is shown, authored by Administrator 5 minutes ago. The commit hash is 4693a6b2. The page also features links for README, Apache License 2.0, CI/CD configuration, and buttons for adding CHANGELOG, CONTRIBUTING, and Auto DevOps enabled. A 'Add Kubernetes cluster' button is also present. At the bottom, a table lists contributors with columns for Name, Last commit, and Last update. The 'cics-genapp' row is highlighted with a red circle around the 'Last commit' column, which shows 'Bug Fixed by Regi'. The 'Clone' button is also highlighted with a red circle.

Section 7. Verify the GitLab CI Build execution..

At this point the changed code is delivered into the GitLab repository that is on Linux. You will use GitLab CI pipeline in action

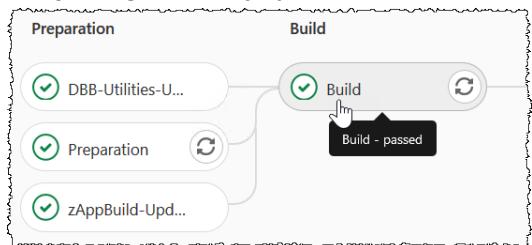
7.1 Verifying the Build

7.1.1 ➡ Click on CI / CD on left to verify the Pipeline in action

Status	Pipeline	Triggerer	Commit	Stages
running	#161 latest		master -> 4693a6b2 Bug Fixed by Regi	✓ ✓ ✓ ✓ ✓
passed	#158		master -> f3b84220	✓ ✓ ✓ ✓ ✓

7.1.2 ➡ Click on Running to see the results

7.1.3 ➡ Click on Build



The connection to z/OS fails with the message below?

There has been a runner system failure,

```
1 Running with gitlab-runner 13.12.0 (7a6612da)
2 on zos.dev runner RVUKgUne
3 Preparing the "ssh" executor
4 Using SSH executor...
5 ERROR: Preparation failed: ssh command Connect() error: ssh Dial() error: dial tcp 10.1.1.2:1022: connect: connection refused
```

Ask the instructor .. He needs to go to SDSF log and issue /S SSHD

7.1.4 Verify that the program **LGICDB01.cbl** is built and also a zUnit is executed since a zUnit test case for **LGICDB01** does exist

```

6 Preparing environment
7 Running on S0W1.DAL-EBIS.IHOST.COM via W-30830...
8 Getting source from Git repository
9 Skipping Git repository setup
10 Skipping Git checkout
11 Skipping Git submodules setup
12 Executing "step_script" stage of the job script
13 echo "*REGI* HLQ ${dbbHlq} DBBHome ${dbbHome}"
14 *REGI* HLQ IBMUSER.DBB DBBHome /var/dbb/1.1
15 echo "*REGI* GitLab workspace created at $CI_PROJECT_DIR"
16 *REGI* GitLab workspace created at /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline
17 $ dbbHome/bin/groovyz $dbbGroovyzOpts $ZAPPBUILD_REPO/build.groovy --logEncoding UTF-8 --workspace $APP_REPO --application cics-genapp --workDir $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID --hlq ${dbbHlq}.GENAPP --url $dbbUrl -pw ADMIN $dbbBuildType $buildVerbose $dbbBuildExtraOpts
18 ** Build start at 20210628.081053.010
19 ** Repository client created for https://clmweb:11043/dbb/
20 ** Build output located at /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010
21 ** Build result created for BuildGroup:cics-genapp-161 BuildLabel:build.20210628.081053.010 at https://clmweb:11043/dbb/rest/buildResult/3048
22 ** --impactBuild option selected. Building impacted programs for application cics-genapp
23 ** Writing build list file to /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/buildList.txt
24 ** Invoking build scripts according to build order: BMS.groovy,Cobol.groovy
25 ** Invoking test scripts according to test order: ZunitConfig.groovy
26 ** Building files mapped to Cobol.groovy script
27 *** Building file cics-genapp/base/src/COBOL/LGICDB01.cbl
28 ** Building files mapped to ZunitConfig.groovy script
29 *** Building file cics-genapp/zUnit/testcfg/LGICDB01.bzucfg
30 *** zUnit Test Job RUNZUNIT(JOB00109) completed with 0
31 **** zUnit Test Job RUNZUNIT(JOB00109) completed with 0
32 **** zUnit Test Job RUNZUNIT(JOB00109) completed with 0
33 **** Module [TLGICDB0] ****
34 Name: TLGICDB0
35 Status: pass
36 Test cases: 1 (1 passed, 0 failed, 0 errors)
37 Details:
38 TEST2 pass
39 **** Module [TLGICDB0] ****

```

7.1.5 Verify that the build was successful

```

1 **** Module [TLGICDB0] ****
2 ** Writing build report data to /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/BuildReport.json
3 ** Writing build report to /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/BuildReport.html
4 ** Updating build result BuildGroup:cics-genapp-161 BuildLabel:build.20210628.081053.010 at https://clmweb:11043/dbb/rest/buildResult/3048
5 ** Build ended at Mon Jun 28 20:12:28 CDT 2021
6 ** Build State : CLEAN
7 ** Total files processed : 2
8 ** Total build time : 1 minutes, 34.599 seconds
9 ** Build finished
10 cd $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID/build*
11 for f in `find . -name "*.zunit.report.log"`; do echo "Process zUnit $f"; Xalan -o $f.xml $f /tmp/AZUZ2J30.xml; done;
12 Process zUnit ./LGICDB01.zunit.report.log
13 Job succeeded

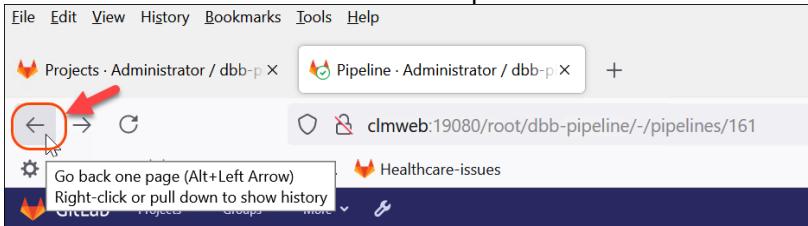
```

Section 8. Verify the GitLab CI Package and Deploy to UCD and test the CICS transaction again using 3270

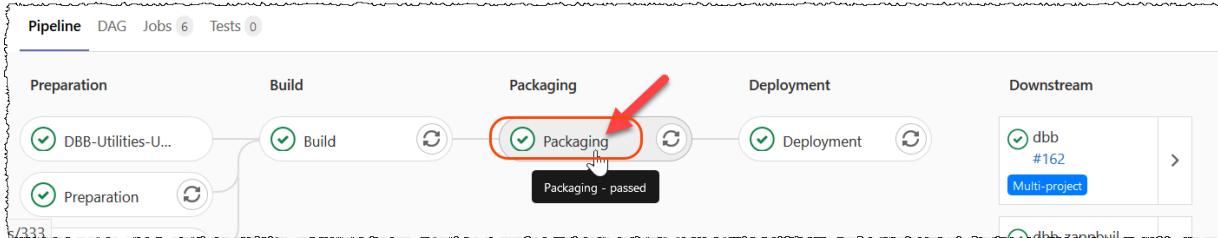
You will verify the results of the second stage (deploy using UCD) and also test the deployed application using CICS.

8.1 Checking the Packaging results (second stage)

8.1.1 Click on the Go back in the top left of the browser:



8.1.2 Click on Packaging



8.1.3 Verify the packaging done by the UCD buztool starting on line 23

```

1 Running with gitlab-runner 13.12.0 (7a6612da)
2 on zos.dev runner RVUKgUne
3 Preparing the "ssh" executor
4 Using SSH executor...
5 Preparing environment
6 Running on S0W1.DAL-EBIS.IHOST.COM via W-30830...
7 Getting source from Git repository
8 Skipping Git repository setup
9 Skipping Git checkout
10 Skipping Git submodules setup
11 Executing "step_script" stage of the job script
12
13 $ echo "*REGI* UCDBUZTOOL_PATH ${ucdbuztool}"
14 *REGI* UCDBUZTOOL_PATH /apps/ucd/v7/bin/buztool.sh
15 $ cd $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID
16 $ export buildListFile=find . -name "buildList.txt"
17 $ if [ -n "$buildListFile" ]; then iconv -f UTF-8 -t IBM-1047 $buildListFile > $buildListFile.IBM1047; chtag -tc IBM-1047 $buildListFile.IBM1047; export DBB_Processed_Files=`wc -l $buildListFile.IBM1047 | awk '{print $1}'`; rm $buildListFile.IBM1047; else export DBB_Processed_Files=0; fi
18 $ if [ "$DBB_Processed_Files" -gt 0 ]; then echo "Move the binaries created by build to UCD to be deployed"; echo "*REGI* Push to UCD Codestation"; cd $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID/build*; export BUILDPATH=$pwd; $dbbHome/bin/groovyz $dbbGroovyzOpts $DBB_REPO/Pipeline/CreateUCDComponentVersion/dbb-ucd-packaging.groovy --buztool ${ucdbuztool} --component ${ucdComponent} --workDir $BUILDPATH; fi
19 Move the binaries created by build to UCD to be deployed
20 *REGI* Push to UCD Codestation
21 ** Create version start at 20210628.081314.013
22 ** Properties at startup:
23 component -> GenApp
24 startTime -> 20210628.081314.013
25 workDir -> /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010
26 preview -> false
27 buztoolPath -> /apps/ucd/v7/bin/buztool.sh
28 ** Read build report data from /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/BuildReport.json
29 ** Find deployable outputs in the build report
30
31

```

```

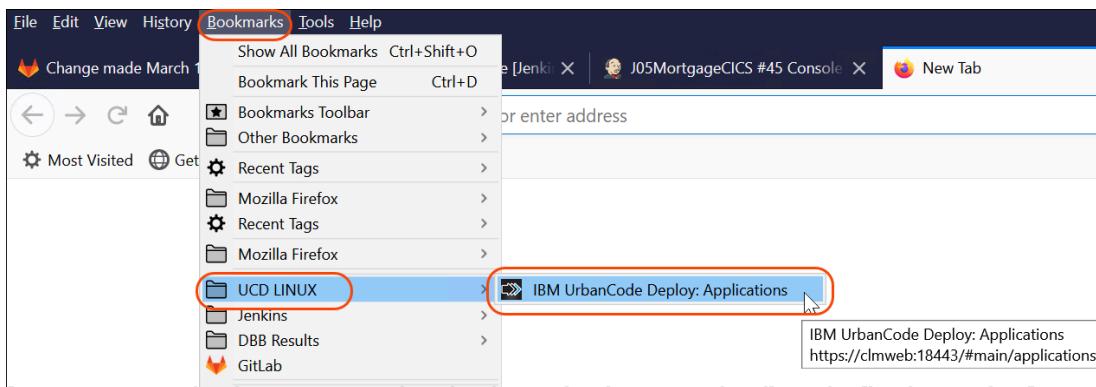
40 zOS toolkit config   : /apps/ucd/v7/ (7.0.2.0,20190131-0837)
41 zOS toolkit binary   : /apps/ucd/v7/ (7.0.2.0,20190131-0837)
42 zOS toolkit data set : BUZV7 (7.0.2.0,20190131-0837)
43 Reading parameters:
44 ....Command : createzosversion
45 ....Component : GenApp
46 ....Generate version name : 20210628-201345
47 ....Shiplist file : /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/shiplist.xml
48 ....Output File:/var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/buztool.output
49 Verifying version
50 ....Repository location : /apps/ucd/v7/var/repository/GenApp/20210628-201345
51 Pre-processing shiplist:
52 ....Shiplist after processing :/apps/ucd/v7/var/repository/GenApp/20210628-201345/shiplist.xml
53 Packaging data sets:
54 ....Location to store zip : /apps/ucd/v7/var/repository/GenApp/20210628-201345
55 ....Zip name : package.zip
56 ....IBMUSER.DBB.GENAPP.DRM.bin
57 ....IBMUSER.DBB.GENAPP.LOAD.bin
58 ....Elapsed time for data set package or deploy operation : 4.938874
59 Post-processing package:
60 PackageManifest file post-processing completed.
61 Create version and store package:
62 ....Version artifacts stored to UCD server CodeStation
63 ....Version:20210628-201345 created
64 Elapsed time 40.0 seconds.
65 ** buztool output properties
66 version.url -> https://ucdserver:18443/#version/d4e49392-8414-41a5-8327-322b23988d25
67 version.repository.type -> CODESTATION
68 version.name -> 20210628-201345
69 version.id -> d4e49392-8414-41a5-8327-322b23988d25
70 component.name -> GenApp
71 version.shiplist -> /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/shiplist.xml
72 ** Build finished
73 Job succeeded

```

8.2 Checking UrbanCode Created version

You can logon to UCD and verify the version created there

8.2.1 ➡ Open another browser tab and start UCD console using the Bookmarks below



8.2.2 ➡ Click Components and GenApp

UrbanCode Deploy

Dashboard Components Applications Configuration Processes Resources Calendar Work Items

Home / Components

Components Templates

Components

Name	Latest Import	Latest Version	Template
Name			
AccountMgmtCICS	20180627-1257160725	MVSCOMPONENT	
CICS	NO VERSION		
GenApp	20210628-201345	MVSCOMPONENT	
LGICDB01			

8.2.3 ➡ Click Versions and the last generated version (this name is also shown on the GitLab log)

UrbanCode Deploy

Dashboard Components Applications Configuration Processes Resources Calendar Work Items Reports

Home / Components / GenApp

Component: GenApp [Show details](#)

Dashboard Usage Configuration Calendar Versions Processes Changes

Version	Statuses	Type	Created By	Date
Filter	Statuses	Any		
20210628-201345		Incremental	admin	6/28/2021, 9:13 PM
20210624-140908		Incremental	admin	6/24/2021, 3:09 PM

8.2.4 ➡ Expand the artifacts and you will see the binaries created at the UCD Code station

Artifacts

Total add: 2 members in 2 data sets

Expand All [Collapse All](#)

Name	Artifact Type	Deploy Type	Inputs	Last Modified	Properties
Filter	Filter	Filter	Filter		
BMUSER.DB.BINAPP.DBRM	[PDS,ADD]				buildcommand=IGYCRCTL buildoptions=LIB,CICS,SQL
LGICDB01	DBRM	cics-genapp/base/src/COBOL/LGICDB01.cbl			
BMUSER.DB.BINAPP.LOAD	[PDS,ADD]				buildcommand=IEWBLINK buildoptions=XREF,LIST,RENT
LGICDB01	LOAD	cics-genapp/base/src/COBOL/LGICDB01.cbl			

8.3 Checking the Deployment results (third stage)

8.3.1 ► Open the GitLab Packaging that an click on the Go back in the top left of the browser:

The screenshot shows a browser window with the URL clmweb:19080/root/dbb-pipeline/-/jobs/333. The page title is "Packaging (#333) · Jobs · Admin". The top navigation bar includes "File Edit View History Bookmarks Tools Help". Below the title, there's a "Go back one page (Alt+Left Arrow)" button highlighted with a red oval and an arrow pointing to it. The main content area shows a pipeline step named "Deployment" with a green checkmark and the status "Deployment - passed".

8.3.2 ► Click on Packaging

The screenshot shows the UrbanCode Deploy Pipeline interface. It displays a flowchart of steps: Preparation, Build, Packaging, and Deployment. The "Deployment" step is highlighted with a red oval and an arrow pointing to it. The status for the Deployment step is "Deployment - passed".

8.3.3 ► The Deploy was successful as seem on lines 17-26

```

1  Running with gitlab-runner 13.12.0 (7a6612da)
2  on zos.dev runner RVUKgUne
3  Preparing the "ssh" executor
4  Using SSH executor...
5  Preparing environment
6  Running on S0W1.DAL-EBIS.IHOST.COM via W-30830...
7
8  Getting source from Git repository
9  Skipping Git repository setup
10 Skipping Git checkout
11 Skipping Git submodules setup
12 Executing "step_script" stage of the job script
13 $ cd $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID
14 $ export buildListFile=find . -name "buildList.txt"
15 $ if [ -n "$buildListFile" ]; then iconv -f UTF-8 -t IBM-1047 $buildListFile > $buildListFile.IBM1047; chtag -tc IBM-1047 $buildListFile.IBM1047; export DBB_Processed_Files=`wc -l $buildListFile.IBM1047 | awk '{print $1}'`; rm $buildListFile.IBM1047; else export DBB_Processed_Files=0; fi
16 $ if [ "$DBB_Processed_Files" -gt 0 ]; then echo "Invoke UCD and wait for the deployment to be completed"; echo "*REGI* Deploy Appl ${ucdApplication} Process ${ucdProcess} to Environment :${ucdEnv} on CICSTS53"; cd $CI_PROJECT_DIR/BUILD-$CI_PIPELINE_ID/build*; export BUILDPATH=`pwd`; $DBB_HOME/bin/groovyz $DBB_REPO/Pipeline/DeployUCDComponentVersion/ucd-deploy.groovy -a "${ucdApplication}" -e "${ucdEnv}" -U $ucdUser -P $ucdPassword -u $ucdSite -d "${ucdComponent:latest}" -p "${ucdProcess}" -k -t 3000000; fi
17
18 *REGI* Deploy Appl GenApp_CICS Process Deploy_to_CICS to Environment :DEV on CICSTS53
19 ** Deploying component versions: GenApp:latest
20 *** Starting deployment process 'Deploy_to_CICS' of application 'GenApp_CICS' in environment 'DEV'
21 *** Follow Process Request: https://clmweb:18443/#applicationProcessRequest/17a55554-6e1b-43b7-ecbd-385707b4df5f
22
23 *** The deployment result is SUCCEEDED. See the UrbanCode Deploy deployment logs for details.
24
25 ** Build finished
26 Job succeeded
27

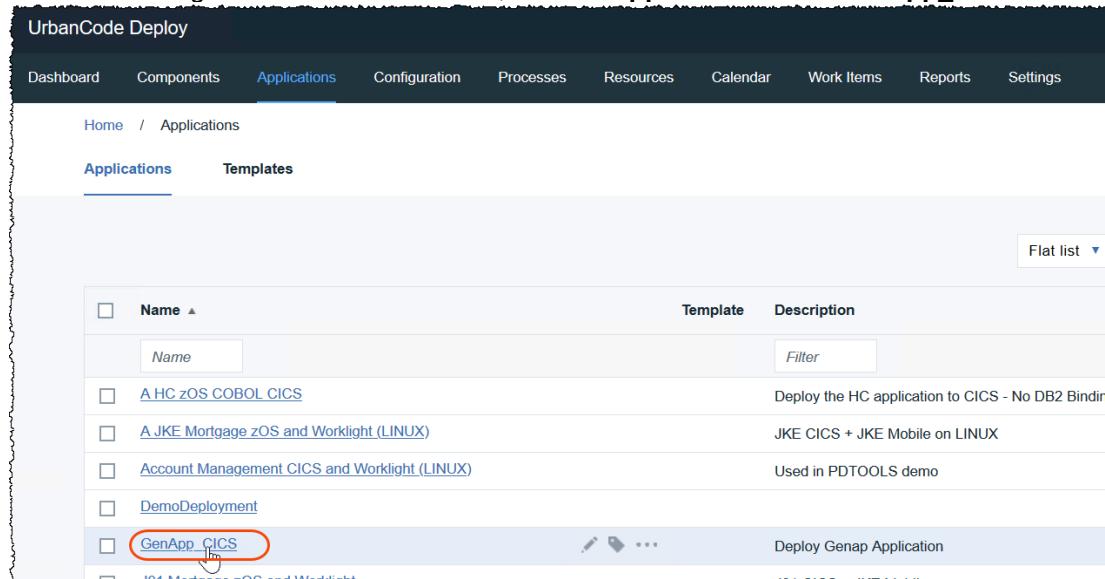
```

A red oval and an arrow point to the line "Job succeeded" at the bottom of the log output.

8.4 Checking UrbanCode Deploy results

You can logon to UCD and verify the deploy results created there

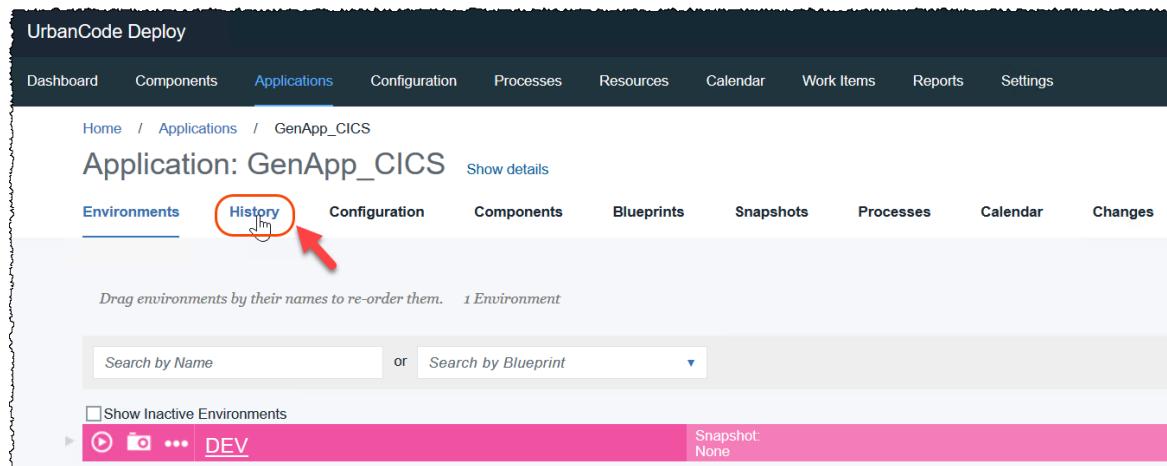
8.4.1  Using the UCD tab on browser, click on **Applications** find **GenApp_CICS** and click on it



The screenshot shows the UrbanCode Deploy interface. The top navigation bar has 'Applications' selected. Below it, a sub-navigation bar has 'Applications' also selected. The main content area displays a table of applications:

Name	Template	Description
A HC zOS COBOL CICS		Deploy the HC application to CICS - No DB2 Binding
A JKE Mortgage zOS and Worklight (LINUX)		JKE CICS + JKE Mobile on LINUX
Account Management CICS and Worklight (LINUX)		Used in PDTTOOLS demo
DemoDeployment		
GenApp_CICS		Deploy Genap Application

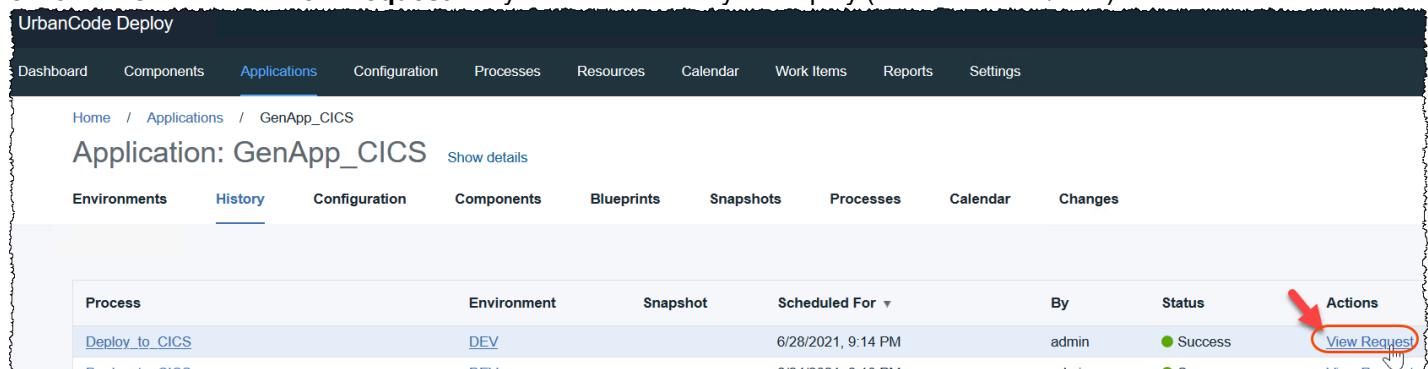
8.4.2  Click on the **History** tab



The screenshot shows the 'GenApp_CICS' application details page. The top navigation bar has 'Applications' selected. Below it, a sub-navigation bar has 'History' selected. The main content area shows the history of environments:

- Environments: DEV
- Search fields: Search by Name, Search by Blueprint
- Show Inactive Environments checkbox
- Snapshot: None

8.4.3  Click on the **View Request** entry that is related to your deploy (check the date/time)



The screenshot shows the 'GenApp_CICS' application details page with the 'History' tab selected. The main content area displays a table of processes:

Process	Environment	Snapshot	Scheduled For	By	Status	Actions
Deploy_to_CICS	DEV		6/28/2021, 9:14 PM	admin	Success	View Request

8.4.4 ► Click **Expand All** and **expand the node** as show in the #2 below
Once the step is green means that it is completed.

Step	Progress	Start Time	Duration	Status
v 1. Install: "GenApp"	1 / 1	9:14:51 PM	0:04:10	Success
v GenApp	1 / 1	9:14:51 PM	0:04:10	Success
Deploy GenApp to CICS (GenApp 20210628-201345)		9:14:51 PM	0:04:10	Success
1. Download Artifacts for zOS		9:14:53 PM	0:01:08	Success
2. Deploy Data Sets		9:16:00 PM	0:01:04	Success
3. GenBndCard		9:17:04 PM	0:00:21	Success
4. Generate Program List		9:17:25 PM	0:00:40	Success
5. Bind Package & Plan		9:17:25 PM	0:00:50	Success
6. NEWCOPY Programs		9:18:15 PM	0:00:46	Success
Total Execution	1 / 1	9:14:51 PM	0:04:10	Success

8.4.5 ► Once the **Bind Package & Plan** is green, move the mouse as below and click **output log**

8.4.6 Notice that The **bind** was successful.

```

Working Directory /apps/ucd/v7/var/work/GenApp
257 DSNT255I -DB2 DSNTBCM2 BIND OPTIONS FOR
258 PACKAGE = DALLASB.GENASA2.LGICDB01.()
259 SQLERROR NOPACKAGE
260 CURRENTDATA NO
261 DEGREE 1
262 DYNAMICRULES BIND
263 DEFER
264 NOREOPT VARS
265 KEEPDYNAMIC NO
266 IMMEDWRITE INHERITFROMPLAN
267 DBPROTOCOL DRDA
268 OPTHINT
269 ENCODING UNICODE (01208)
270 PLAMGMT OFF
271 PLAMMGMTSCOPE STATIC
272 CONCURRENTACCESSRESOLUTION
273 EXTENDEDINDICATOR
274 PATH
275 DSNT275I -DB2 DSNTBCM2 BIND OPTIONS FOR
276 PACKAGE = DALLASB.GENASA2.LGICDB01.()
277 QUERYACCELERATION
278 GETACCELARCHIVE
279 DSNT232I -DB2 SUCCESSFUL BIND FOR
280 PACKAGE = DALLASB.GENASA2.LGICDB01.()
281 DSN
282 DSN
283 DSN
284 END
285 1 job(s) completed.
286

```

8.4.7 ► Once the **NEWCOPY Programs** is green, move the mouse as below and click **output log**

1 / 1 9:14:51 PM
Deploy GenApp to CICS (GenApp 20210628-201345)
1. Download Artifacts for zOS 9:14:53 PM
2. Deploy Data Sets 9:16:00 PM
3. GenBndCard 9:17:04 PM
4. Generate Program List 9:17:25 PM
5. Bind Package & Plan 9:17:25 PM
6. NEWCOPY Programs 9:18:15 PM
(CICS TS v. 41.20181207-0633)
Total Executed 1 / 1 9:14:51 PM
Output Log

8.4.8 Notice that **LGICDB01** program was deployed to CICS Dev environment. and a CICS NEWCOPY was succeeded.

```
Working Directory /apps/ucd/v7/var/work/GenApp
17 resourceNameList=LGICDB01,
18 resourceType=PROGRAM
19 retryInterval=
20 sel=
21 ts_location=
22 ts_password=
23 ts_type=
24 username=
25 environment:
26 AGENT_HOME=/apps/ucd/v7
27 AH_AUTH_TOKEN=*****
28 AH_WEB_URL=https://ucdserver:18443
29 AUTH_TOKEN=*****
30 DS_AUTH_TOKEN=*****
31 DS_SYSTEM_ENCODING=IBM-1047
32 GROOVY_HOME=/apps/ucd/v7/opt/groovy-2.4.15
33 JAVA_OPTS=-Dfile.encoding=IBM-1047 -Dconsole.encoding=IBM-1047
34 PLUGIN_HOME=/apps/ucd/v7/var/plugins/com.ibm.ucd.plugin.cics_41_5a9aaf832db74606b0e690aecbcdc462781b92dbedf355bc800df440bd516e2d
35 US_DIALOGUE_ID=762377e6-f0f2-4f76-a6e2-b226f25b1324
36 WE_ACTIVITY_ID=17a55555-3663-1403-34b5-89fd05c3a581
37 =====
38 2021/06/29 01:19:20.028 GMT BUZCP0006I Connected to "10.1.1.2:1490". CICS TS version: 050300.
39 2021/06/29 01:19:25.633 GMT BUZCP0037I Perform NEWCOPY Operation.
40 2021/06/29 01:19:26.277 GMT BUZCP0024I NEWCOPY PROGRAM "LGICDB01" succeeded. RED HIGHLIGHT
41 2021/06/29 01:19:26.489 GMT BUZCP0029I Summary: 1 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.
42
43 =====
44 Post Processing Script Execution Console Output:
45
46
```

8.5 Testing the CICS code deployed to Dev environment

8.5.1 ► Double click on the **3270 terminal emulator** that is on the Windows desktop



8.5.2 ► Type **I cicsts53.** (I as logon) and press **Enter** key.

```

Session A - [24 x 80]
File Edit Settings View Communication Actions Window Help
Print 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 = SC0TCP03
Application Developer System
      // 0000000 SSSSS
      // 00 00 SS
zzzzzz // 00 00 SSSS
zz // 00 00 SSSS
zz // 00 00 SS
zzzzzz // 0000000 SSSS
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"
1 cicsts53
24/01/2023 24/01/2023
Connected to remote server/host zos-dev using lu/pool SC0TCP03 and port 23
  
```

8.5.3 ► Logon using your z/OS user id and password (**ibmuser/sys1**) and press **Enter**
Type your userid and password, then press ENTER:

```

Userid . . . ibmuser Groupid . . .
Password . . . ibmuser
Language . . .
New Password . . .
  
```

8.5.4 The sign-on message is displayed

```
DFHCE3549 Sign-on is complete (Language ENU).
```

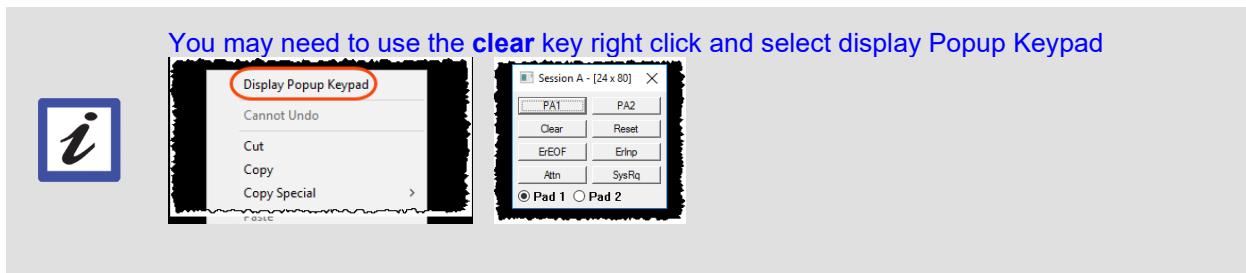
8.5.5 ► Type the CICS transaction **ssc1** and press the **Enter** key.

```

zosdev.hce X
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
ssc1_
  
```

8.5.6 ► Move the mouse to last **0** and type **1**, use the tab key and type **1** as **Select option** and press **Enter**

SSC1 General Insurance Customer Menu	
1. Cust Inquiry	Cust Number
2. Cust Add	Cust Name :First :Last
4. Cust Update	DOB House Name House Number Postcode Phone: Home Phone: Mob Email Addr
	(yyyy-mm-dd)
Select Option 1	



8.5.7 ► The data below is retrieved from a DB2 table .
Notice that **DOB** (*Date Of Birthday*) is now correct. **The bug is fixed.**

```
SSC1          General Insurance Customer Menu

1. Cust Inquiry      Cust Number      0000000001
2. Cust Add          Cust Name :First  Andrew
4. Cust Update       :Last           Pandy
                                DOB             1950-07-11   (yyyy-mm-dd)
                                House Name
                                House Number
                                Postcode
                                Phone: Home    01962 811234
                                Phone: Mob     07799 123456
                                Email Addr    A.Pandy@beebhouse.com

Select Option  1
```

8.5.8 ► Press **F3** to end the application.

```
zosdev.hce
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
Transaction ended
```

Congratulations! You have completed the Lab 10. .

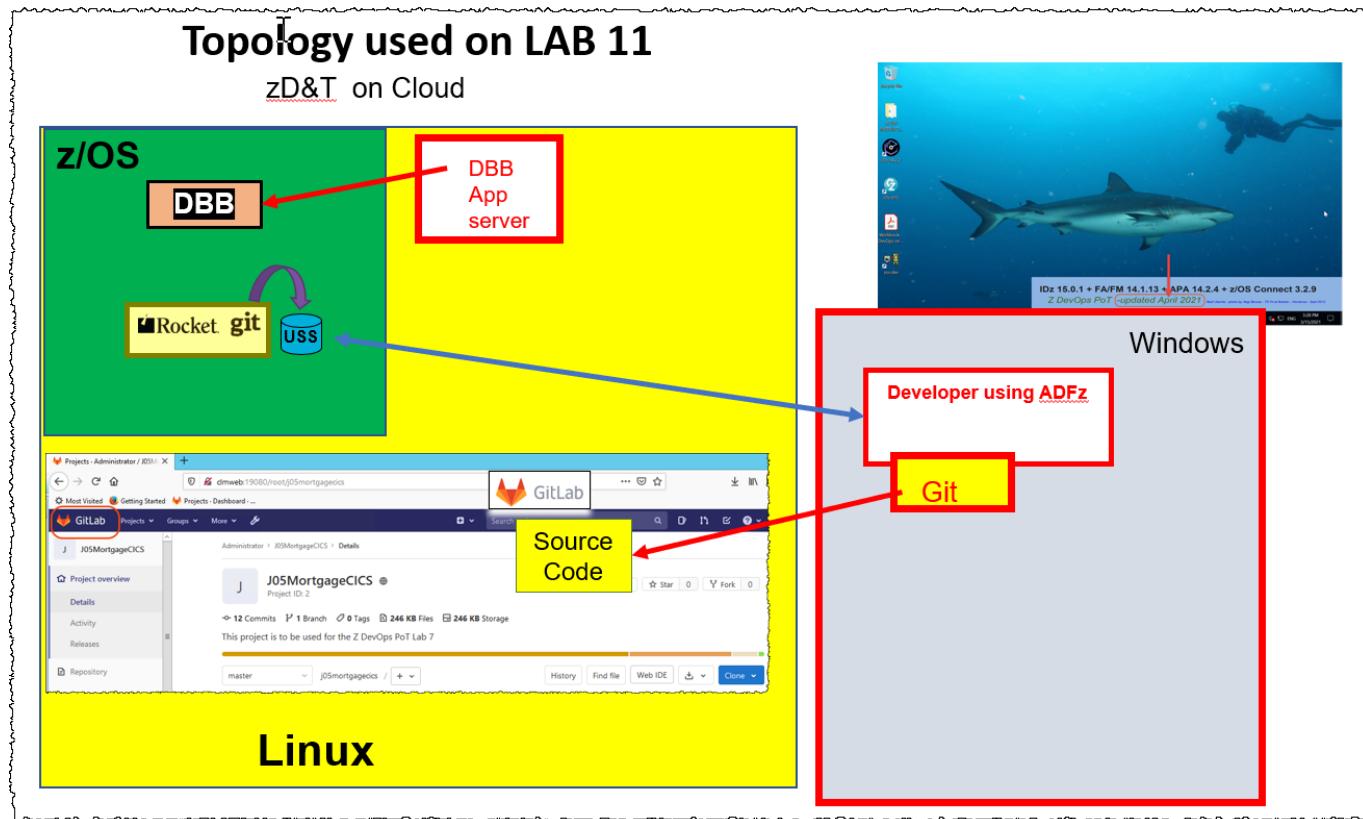
LAB 11 – (OPTIONAL) Migrating partitioned data sets (PDS) to a distributed Git repository. (50 minutes)

Updated Aug-17-2021 by Regi Barosa. Created by Z DevOps Transformation Team (DAT) and reviewed by Wilbert Kho and Mathieu Dalbin.

Even though Git is not a prerequisite of DBB, the existence of a Git client on z/OS was the driving factor for the creation of DBB.

Most DBB users are assumed to install the Rocket Software port of the Git client to z/OS as part of their DevOps solution. To facilitate the move to Git as an SCM for z/OS development, **you can use a DBB migration tool** that copies source code from partitioned data sets (PDS) to a local Git repository on the z/OS File System (zFS) to be committed and pushed to a distributed Git server.

More details at <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>



Overview of development tasks

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

1. **Logon to IDz and prepare the necessary directories and script migration file on zFS**

You will start IDz and prepare some folders and the script migration file required for the migration on z/OS UNIX Files.

2. **Migrating using DBB migration tool to a local Rocket Git client repository on zFS**

→ You will use the DBB migration tool, and the Rocket Git client installed on USS system, and create a Git repository.

3. **Push the migrated files to Git server (Gitlab)**

When all the source files have been copied to the local Rocket Git repository, you are ready to push the files to the Git server as follows,

4. **Load the source code from Git to the local IDz workspace and start working with the migrated assets**

Once the migration is complete on Git server you may clone using IDz and start working with the migrated assets.

What is Git and DBB?

IBM Dependency Based Build (DBB) is an intelligent build system for traditional z/OS applications written in languages such as COBOL and PL/I that allows the analysis of build dependencies between objects. The goal of DBB is to provide automation capabilities that can be used on z/OS

IBM DBB is a standalone framework (it does not require a specific source code manager or automation tool) to simplify the process of building code on z/OS based on a modern scripting language.

z/OS development teams have the freedom to choose a modern software configuration management (SCM) tool, such as Git, and continuous integration tools, such as Jenkins or GitLab, to build traditional z/OS applications written in COBOL or PL/I.

DBB allows you to standardize DevOps processes and practices across multiple platforms



Git is an open-source Code Management tool that is very popular in the distributed world.

In early 2017, Rocket Software ported Git into the mainframe – with the necessary checks to handle EBCDIC to UTF-8 conversions and vice-versa

GitHub vs. Bitbucket vs. GitLab ?

More at: <https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab>

GitHub, **Bitbucket**, and **GitLab** are code collaboration and version control tools offering repository management. They each have their share of fans, though **GitHub** is by far the most used of the three.

Of the three, only **GitLab** is open source, though all three support open source projects.

GitHub offers free public repositories; **Bitbucket** also offers free private repositories; **GitLab** offers a Community Edition which is entirely free

Section 1. Logon to IDz and prepare the necessary directories and files on zFS

For the migration of the PDS to distributed Git server you will use the Rocket Git Client and USS files also known as zFS files. IDz can help creating those folders and files there.

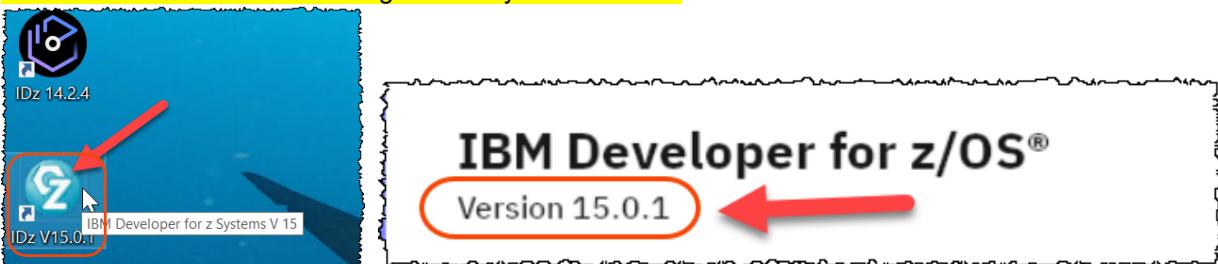
1.1 Connect to z/OS using IDz

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

► Using the Windows desktop double click on **IDz V15** icon.

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

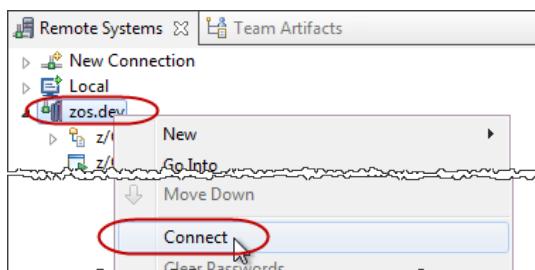
IMPORTANT -> This icon will start an IDz workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.



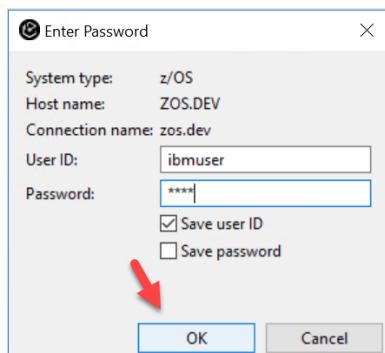
1.1.2 ► Open the **z/OS Projects** perspective by selecting

Window > Perspective > Open Perspective > z/OS Projects

1.1.3 ► Right click on **zos.dev** and select **Connect**.

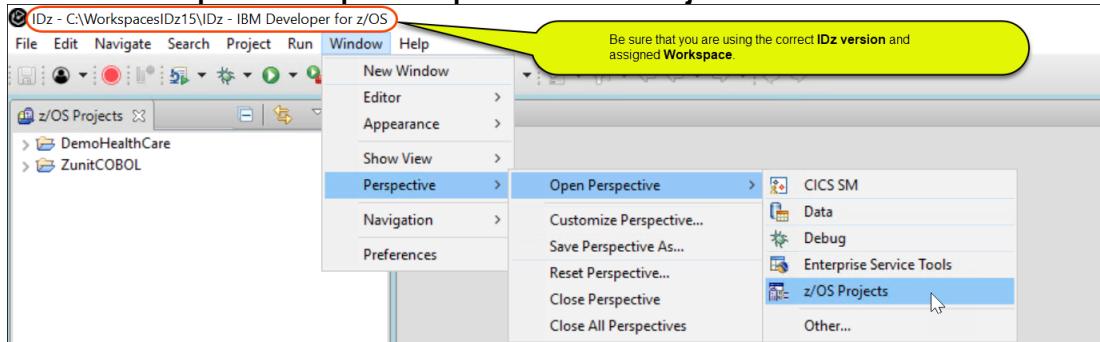


1.1.4 ► Type **ibmuser** as userid and **sys1** as password. Click **OK** to connect to z/OS.



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

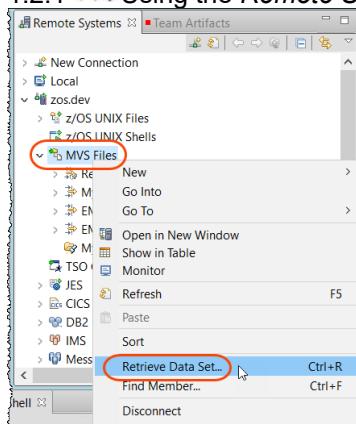
**1.1.6 ➡ Open the z/OS Projects perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects**



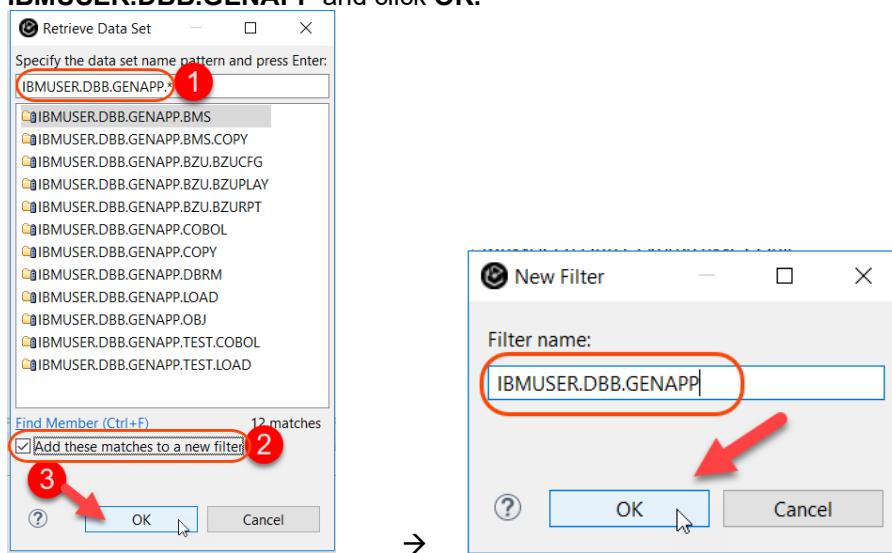
1.2 Verify the Z/OS datasets to be migrated

The application to be migrated is the GENAPP application and the source code is on the HLQ named **IBMUSER.DBB.GENAPP.***. You can create a filter that lists all datasets with this HLQ.

1.2.1 ➡ Using the Remote Systems perspective (on top right) right click **zos.dev and select **Retrieve Data Set...****

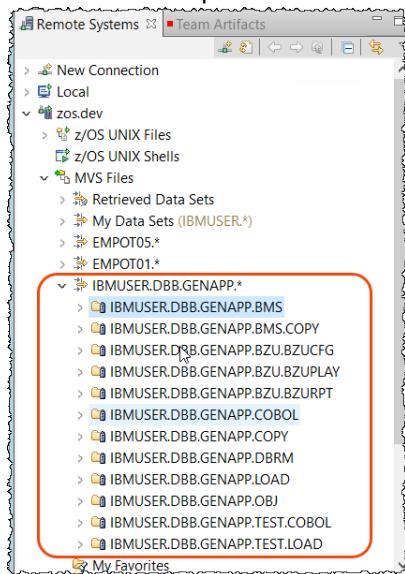


1.2.2 ➡ Type **IBMUSER.DBB.GENAPP.* and press **Enter**. Select **Add these matches to a new filter** type **IBMUSER.DBB.GENAPP** and click **OK**.**



1.2.3 You will see the new filter created with some of the datasets that will be migrated to Git Server. The files stored in ***.COBOL**, ***.BMS** and ***.COPY** are the files which we will migrate.

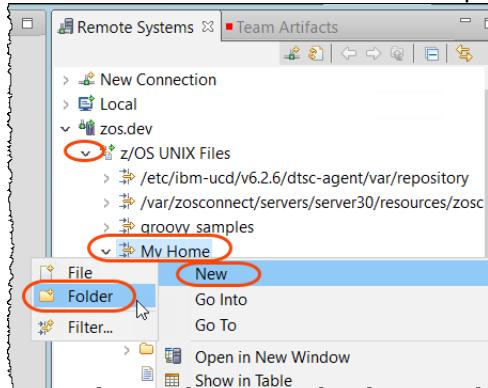
► You can expand those PDS to see the content if you want.



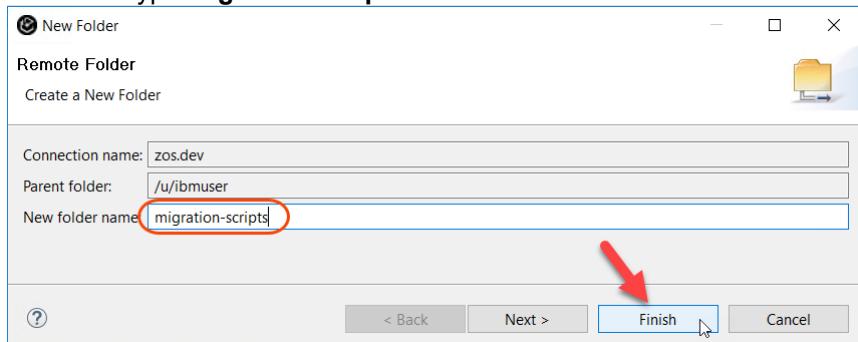
1.3 Create directories and script migration file at zFS

You will create folders named ***migration-scripts*** and ***migration-repository*** required for the migration on z/OS UNIX. One will hold the migration scripts and the other will hold the *Rocket Git* repository.

1.3.1 To create folders on ZFs. ► Expand the **z/OS Unix Files** and right click **My Home > New > Folder**

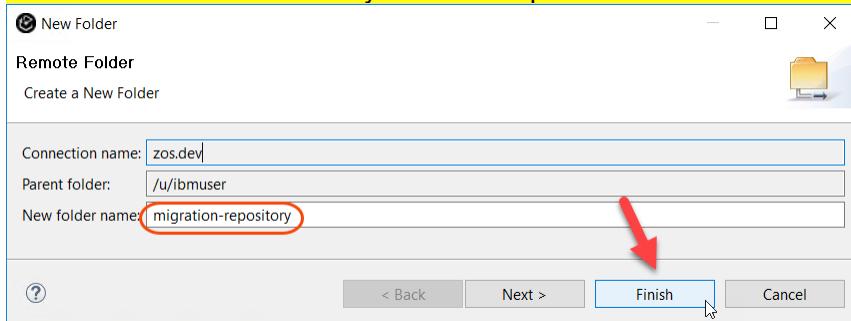


1.3.2 ► Type ***migration-scripts*** and click **Finish**

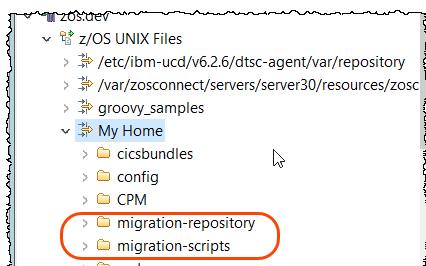


1.3.3 ► Repeat for the folder **migration-repository** and click **Finish**

IMPORTANT → Be sure that you have no spaces after the folder name.

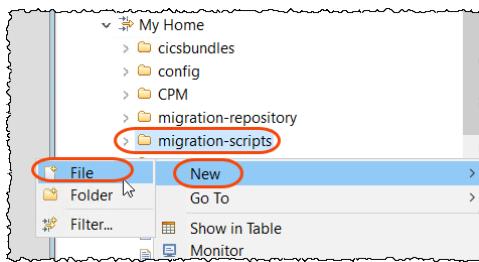


1.3.4 The folders are created under **My Home (u/ibmuser)**



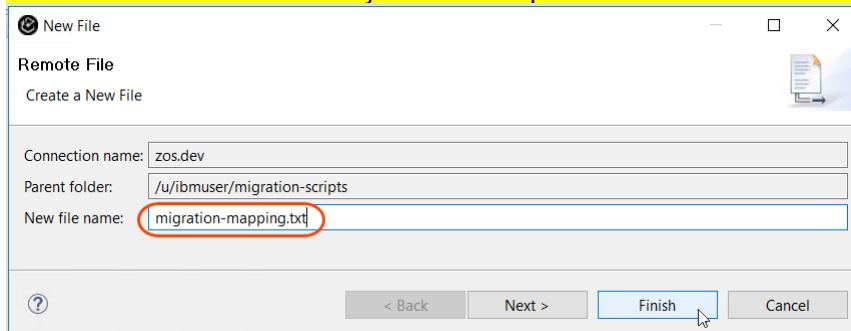
1.3.5 To create the migration scripts..

► Right click **migration-scripts** folder and select **New > File**

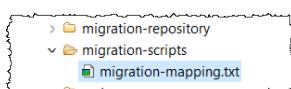


1.3.6 ► Type **migration-mapping.txt** and click **Finish**

IMPORTANT → Be sure that you have no spaces after the folder name.



1.3.7 Verify the file created. You will now add content to it.



1.3.8 ► Double click **migration-mapping.txt** (take a while to open it..)



1.3.9 ► You will need to copy all data from the box below and paste to the file being edit ..

You can use **Ctrl + C** to copy the data.

Notice that for each PDS member you specify the location where the code will be moved to in the USS directory named **migration-repository**.

```

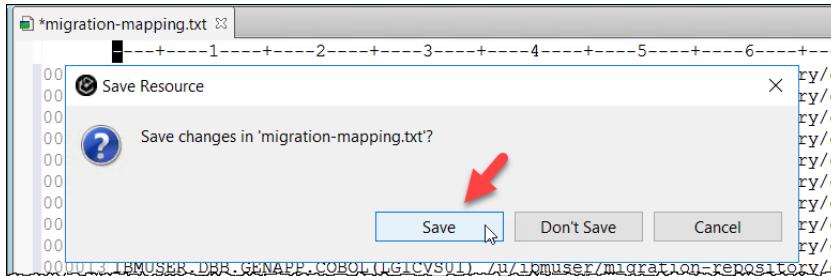
IBMUSER.DBB.GENAPP.COBOL(LGACDB01) /u/ibmuser/migration-repository/cobol/lgacdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGACDB02) /u/ibmuser/migration-repository/cobol/lgacdb02.cbl
IBMUSER.DBB.GENAPP.COBOL(LGACUS01) /u/ibmuser/migration-repository/cobol/lgacus01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGACVS01) /u/ibmuser/migration-repository/cobol/lgacvs01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGAPDB01) /u/ibmuser/migration-repository/cobol/lgapdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGAPOL01) /u/ibmuser/migration-repository/cobol/lgapol01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGAPVS01) /u/ibmuser/migration-repository/cobol/lgapvs01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGDPDB01) /u/ibmuser/migration-repository/cobol/lgdpdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGDPOL01) /u/ibmuser/migration-repository/cobol/lgdpol01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGDPVS01) /u/ibmuser/migration-repository/cobol/lgdpv01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGICDB01) /u/ibmuser/migration-repository/cobol/lgicdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGICUS01) /u/ibmuser/migration-repository/cobol/lgicus01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGICVS01) /u/ibmuser/migration-repository/cobol/lgicvs01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGIPDB01) /u/ibmuser/migration-repository/cobol/lgipdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGIPOL01) /u/ibmuser/migration-repository/cobol/lgipol01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGIPVS01) /u/ibmuser/migration-repository/cobol/lgipvs01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGSETUP) /u/ibmuser/migration-repository/cobol/lgsetup.cbl
IBMUSER.DBB.GENAPP.COBOL(LGSTSQ) /u/ibmuser/migration-repository/cobol/lgstsq.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUUCDB01) /u/ibmuser/migration-repository/cobol/lguucdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUCUS01) /u/ibmuser/migration-repository/cobol/lgucus01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUCVS01) /u/ibmuser/migration-repository/cobol/lgucvs01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUPDB01) /u/ibmuser/migration-repository/cobol/lgupdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUPOL01) /u/ibmuser/migration-repository/cobol/lgupo101.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUPVS01) /u/ibmuser/migration-repository/cobol/lgupvs01.cbl
IBMUSER.DBB.GENAPP.COPY(LGCMAREA) /u/ibmuser/migration-repository/copy/lgcarea.cpy
IBMUSER.DBB.GENAPP.COPY(LGPOLICY) /u/ibmuser/migration-repository/copy/lgpolicy.cpy
IBMUSER.DBB.GENAPP.COPY(SSMAP) /u/ibmuser/migration-repository/copy/ssmap.cpy
IBMUSER.DBB.GENAPP.BMS(SSMAP) /u/ibmuser/migration-repository/map/ssmap.bms

```

1.3.10 ➡ Paste to the first position of the first line the scripts copied

```
*migration-mapping.txt
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8-----+
000004 IBMUSER.DBB.GENAPP.COBOL(LGACVS01) /u/ibmuser/migration-repository/cobol/lgacvs01.cbl
000005 IBMUSER.DBB.GENAPP.COBOL(LGAPDB01) /u/ibmuser/migration-repository/cobol/lgapdb01.cbl
000006 IBMUSER.DBB.GENAPP.COBOL(LGAPOL01) /u/ibmuser/migration-repository/cobol/lgapol01.cbl
000007 IBMUSER.DBB.GENAPP.COBOL(LGAPVS01) /u/ibmuser/migration-repository/cobol/lgapvs01.cbl
000008 IBMUSER.DBB.GENAPP.COBOL(LGDDB01) /u/ibmuser/migration-repository/cobol/lgddb01.cbl
000009 IBMUSER.DBB.GENAPP.COBOL(LGDPOL01) /u/ibmuser/migration-repository/cobol/lgdpol01.cbl
000010 IBMUSER.DBB.GENAPP.COBOL(LGDPVS01) /u/ibmuser/migration-repository/cobol/lgdps01.cbl
000011 IBMUSER.DBB.GENAPP.COBOL(LGICDB01) /u/ibmuser/migration-repository/cobol/lgicdb01.cbl
000012 IBMUSER.DBB.GENAPP.COBOL(LGICUS01) /u/ibmuser/migration-repository/cobol/lgicus01.cbl
000013 IBMUSER.DBB.GENAPP.COBOL(LGICVS01) /u/ibmuser/migration-repository/cobol/lgicvs01.cbl
000014 IBMUSER.DBB.GENAPP.COBOL(LGIPDB01) /u/ibmuser/migration-repository/cobol/lgipdb01.cbl
000015 IBMUSER.DBB.GENAPP.COBOL(LGIPOL01) /u/ibmuser/migration-repository/cobol/lgipol01.cbl
000016 IBMUSER.DBB.GENAPP.COBOL(LGIPVS01) /u/ibmuser/migration-repository/cobol/lgipvs01.cbl
000017 IBMUSER.DBB.GENAPP.COBOL(LGSETUP) /u/ibmuser/migration-repository/cobol/lgsetup.cbl
000018 IBMUSER.DBB.GENAPP.COBOL(LGSTSQ) /u/ibmuser/migration-repository/cobol/lgstsq.cbl
000019 IBMUSER.DBB.GENAPP.COBOL(LGUCDB01) /u/ibmuser/migration-repository/cobol/lgucdb01.cbl
000020 IBMUSER.DBB.GENAPP.COBOL(LGUCUS01) /u/ibmuser/migration-repository/cobol/lgucus01.cbl
000021 IBMUSER.DBB.GENAPP.COBOL(LGUCVS01) /u/ibmuser/migration-repository/cobol/lgucvs01.cbl
000022 IBMUSER.DBB.GENAPP.COBOL(LGUDB01) /u/ibmuser/migration-repository/cobol/lgudb01.cbl
000023 IBMUSER.DBB.GENAPP.COBOL(LGUPOL01) /u/ibmuser/migration-repository/cobol/lgupol01.cbl
000024 IBMUSER.DBB.GENAPP.COBOL(LGUPVS01) /u/ibmuser/migration-repository/cobol/lgupvs01.cbl
000025 IBMUSER.DBB.GENAPP.COPY(LGMAREA) /u/ibmuser/migration-repository/copy/lgmarea.cpy
000026 IBMUSER.DBB.GENAPP.COPY(LGPOLICY) /u/ibmuser/migration-repository/copy/lgpolicy.cpy
000027 IBMUSER.DBB.GENAPP.COPY(SSMAP) /u/ibmuser/migration-repository/copy/ssmap.cpy
000028 IBMUSER.DBB.GENAPP.BMS(SSMAP) /u/ibmuser/migration-repository/map/ssmap.bms
000029 |
```

1.3.11 ➡ Close the editor and click **Save** to save the copied content..



What have you done so far?

You used IDz to verify the datasets to be migrated to Git. Also using IDz you created two folders on Z/OS UNIX Files.
One to hold the Rocket Git repository and the other with the migration scripts.
You are now ready to start the migrations using the DBB toolkit provided.

Section 2 . Migrating using DBB migration tool to a local Rocket Git repository on zFS

You can use the migration tool to copy source files from data set libraries to the local Rocket Git repository. In addition to copying the source from PDS to zFS, the migration tool also creates and updates the .gitattributes file. The .gitattributes file is required by Rocket's Git client to perform automatic codepage conversion between the Git server and the local Git repository during the migration.

We will run the migration tool by executing a shell script called migrate.sh which is located in **/var/dbb/1.1/migration/bin**. The tool has two modes of operation:

- Run migration using a *mapping file*
- Run migration using a *mapping rule*

We will run using the mapping file.

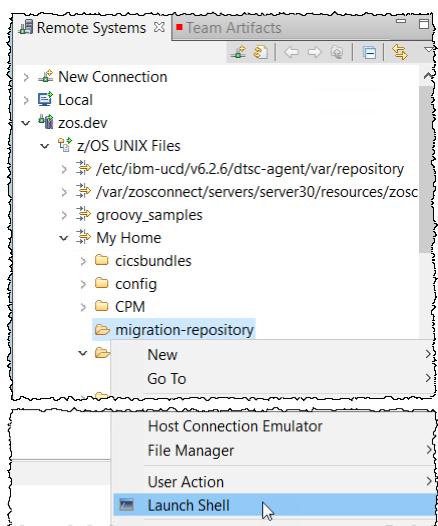
More details at <https://www.ibm.com/docs/en/ad fz/dbb/1.0.0?topic=migrating-data-sets-git>

2.1 Setting up a local Rocket Git repository on zFS

On this step you will initialize the Rocket git repository using the *migration-repository* folder on Unix System Services. You will need to have a UNIX Shell to perform this activity and IDz can also help you here.

2.1.1 To create a UNIX shell.

► Right click **migration-repository** and select **Launch Shell**



2.1.2 Notice that you are already on the `/u/ibmuser/migration-repository` when the UNIX shell was launched.

► Initialize a new git repository by issuing **git init**

```
zos.dev % sh
cd /u/ibmuser/migration-repository
/u/ibmuser/migration-repository>
Command git init
```

What is **Git init** ?

The `git init` command creates a new Git repository.

It can be used to convert an existing, un-versioned project to a Git repository or initialize a new, empty repository.

Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.



You will create a Rocket client git repository using the folder created at the z/OS Unix folder named **migration-repository**

2.1.3 ► Double click on the title “Remote Shell” to maximize this view

```
zos.dev ~
sh

cd /u/ibmuser/migration-repository
/u/ibmuser/migration-repository>
```

2.1.4 See the message displayed:

```
IDz - z/OS Projects - IBM Developer for z/OS - C:\WorkspacesADF3.1\IDz
File Edit Navigate Search Project Data Run Window Help
File View Insert Tools Options Window Help
Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote System Details Remote Shell
zos.dev ~
sh

cd /u/ibmuser/migration-repository
/u/ibmuser/migration-repository>
git init
Initialized empty Git repository in /u/ibmuser/migration-repository/.git/
/u/ibmuser/migration-repository>
```

2.2 Run migration using a mapping file

After the local Git repository has been created on zFS, you can start using the migration tool to copy source files from data set libraries to the local repository. In addition to copying the source from PDS to zFS, the migration tool also creates and updates the `.gitattributes` file.

The `.gitattributes` file is required by Rocket's Git client to perform automatic codepage conversion between the Git server and the local Git repository during the migration.

2.2.1 To run the migration using the DBB tool into the initialized git repository on USS:

► cd /var/dbb/1.1/migration/bin

```
Command cd /var/dbb/1.1/migration/bin
```

► migrate.sh -r /u/ibmuser/migration-repository/ /u/ibmuser/migration-scripts/migration-mapping.txt

```
Command migrate.sh -r /u/ibmuser/migration-repository/ /u/ibmuser/migration-scripts/migration-mapping.txt
```

IMPORTANT:

Notice that after pressing enter nothing seems to happen..

But in fact the command is running..

Be patient..

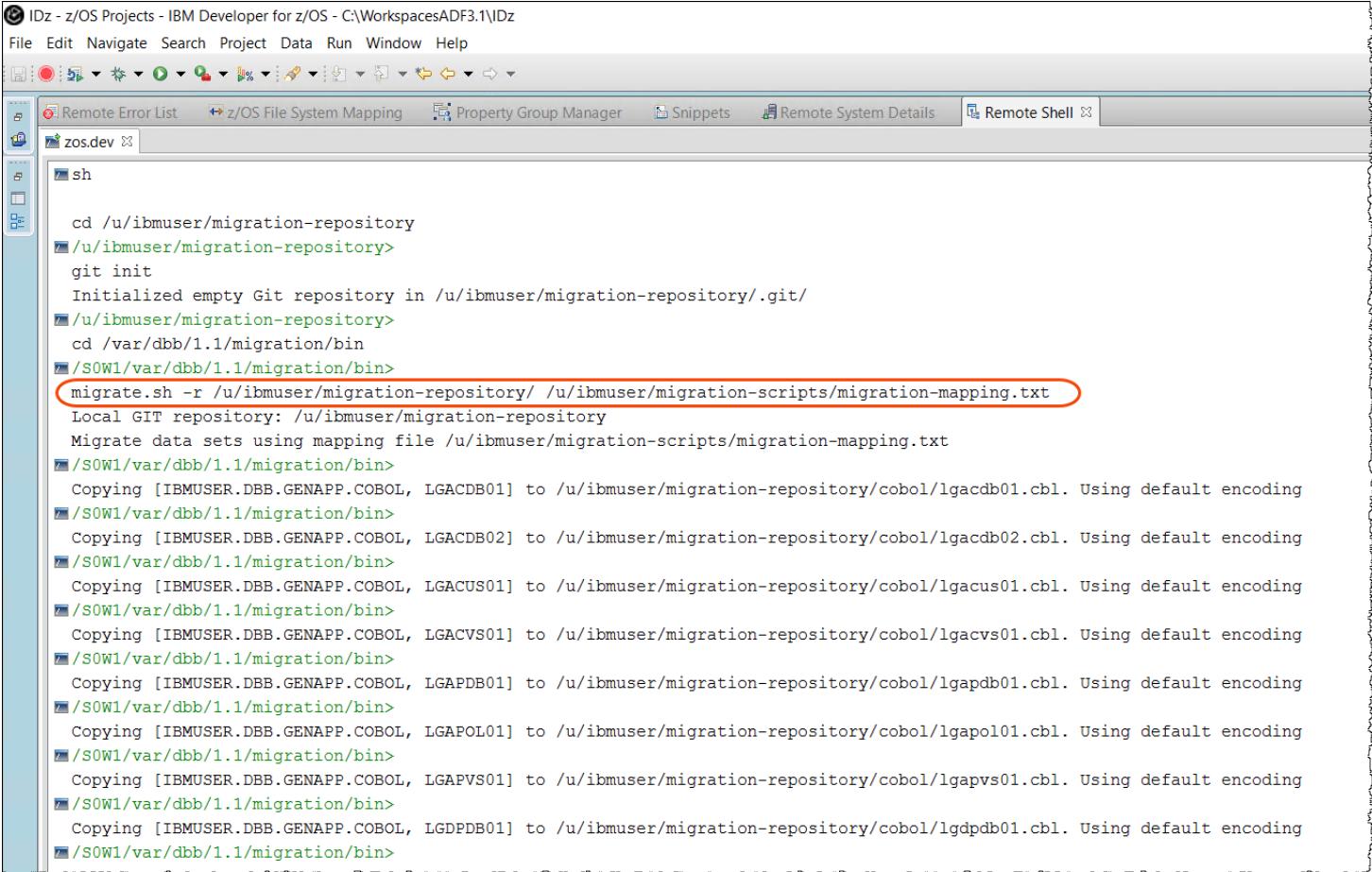
DO NOT enter this command again. Just wait....

2.2.2 Please be patient it will take a few minutes to complete on ZDT.

Please review the log.

Notice that each line in the mapping file maps a fully qualified PDS member to the absolute path of the zFS file it is to be copied.

Also each line can have an optional PDS encoding value if the encoding of the PDS member is not the default IBM-1047



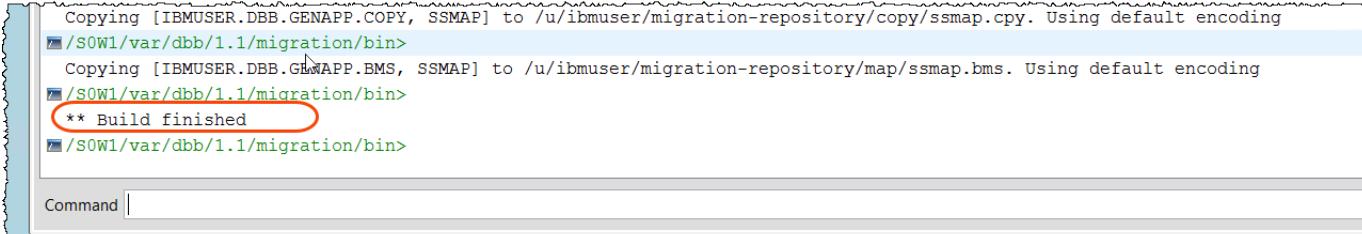
```

IDz - z/OS Projects - IBM Developer for z/OS - C:\WorkspacesADF3.1\IDz
File Edit Navigate Search Project Data Run Window Help
Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote System Details Remote Shell
zos.dev ✘
sh
cd /u/ibmuser/migration-repository
/u/ibmuser/migration-repository>
git init
Initialized empty Git repository in /u/ibmuser/migration-repository/.git/
/u/ibmuser/migration-repository>
cd /var/dbb/1.1/migration/bin
/S0W1/var/dbb/1.1/migration/bin>
migrate.sh -r /u/ibmuser/migration-repository/ /u/ibmuser/migration-scripts/migration-mapping.txt
Local GIT repository: /u/ibmuser/migration-repository
Migrate data sets using mapping file /u/ibmuser/migration-scripts/migration-mapping.txt
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGACDB01] to /u/ibmuser/migration-repository/cobol/lgacdb01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGACDB02] to /u/ibmuser/migration-repository/cobol/lgacdb02.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGACUS01] to /u/ibmuser/migration-repository/cobol/lgacus01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGACVS01] to /u/ibmuser/migration-repository/cobol/lgacvs01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGAPDB01] to /u/ibmuser/migration-repository/cobol/lgapdb01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGAPOL01] to /u/ibmuser/migration-repository/cobol/lgapol01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGAPVS01] to /u/ibmuser/migration-repository/cobol/lgapvs01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GENAPP.COBOL, LGDPDB01] to /u/ibmuser/migration-repository/cobol/lgdadb01.cbl. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>

```

2.2.3 When the process is completed, you will see the prompt along with the **** Build finished** message.

Notice that this process populate the local Rocket git repository and also verifies if the members contain "non-roundtripable" characters



```

Copying [IBMUSER.DBB.GENAPP.COPY, SSMAP] to /u/ibmuser/migration-repository/copy/ssmap.cpy. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
Copying [IBMUSER.DBB.GWAPP.BMS, SSMAP] to /u/ibmuser/migration-repository/map/ssmap.bms. Using default encoding
/S0W1/var/dbb/1.1/migration/bin>
** Build finished
/S0W1/var/dbb/1.1/migration/bin>

```

2.2.4 Verify in the Remote Shell view, that the files arrived at the /u/ibmuser/migration-repository:

Issue:

```
▶▶ cd /u/ibmuser/migration-repository
▶▶ ls -al
```

```
/S0W1/var/dbb/1.1/migration/bin>
cd /u/ibmuser/migration-repository
/u/ibmuser/migration-repository>
ls -al
total 112
drwx----- 6 IBMUSER SYS1 8192 Aug 13 10:11 .
drwxr-xr-x 13 IBMUSER SYS1 8192 Aug 12 17:35 ..
drwxr-xr-x 7 IBMUSER SYS1 8192 Aug 13 09:48 .git
-rw-r--r-- 1 IBMUSER SYS1 195 Aug 13 10:11 .gitattributes
drwxr-xr-x 2 IBMUSER SYS1 8192 Aug 13 10:10 cobol
drwxr-xr-x 2 IBMUSER SYS1 8192 Aug 13 10:11 copy
drwxr-xr-x 2 IBMUSER SYS1 8192 Aug 13 10:11 map
/u/ibmuser/migration-repository>
```

At this point, we have populated the **Rocket git repository** on USS.

2.3 Setting the file tag after the migration

The migration tool on the DBB 1.1.0 being used does not set the file tag, which is important.

This is fixed on newer DBB versions like DBB 1.1.1.

2.3.1 We need to set them with the below commands. Please issue commands

```
▶▶ chtag -c ISO8859-1 -t .gitattributes
▶▶ chtag -c IBM-1047 -t cobol/*
▶▶ chtag -c IBM-1047 -t copy/*
▶▶ chtag -c IBM-1047 -t map/*
```

```
/u/ibmuser/migration-repository>
ctag -c ISO8859-1 -t .gitattributes
/u/ibmuser/migration-repository>
ctag -c IBM-1047 -t cobol/*
/u/ibmuser/migration-repository>
ctag -c IBM-1047 -t copy/*
/u/ibmuser/migration-repository>
ctag -c IBM-1047 -t map/*
/u/ibmuser/migration-repository>
```

2.3.2 Verify the .gitattributes content created in /u/ibmuser/migration-repository/

This was created by the migration tool.

When moving to or from Rocket Git client to the Git server the files will be converted as specified here.

```
▶▶ cat .gitattributes
```

```
/u/ibmuser/migration-repository>
cat .gitattributes
cobol/*.cbl zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
copy/*.cpy zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
map/*.bms zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
```

Managing non-roundtripable or non-printable characters?

The migration tool does not do any encoding conversion when migrating members to ZFS, so if members are encoded in EBCDIC, the files copied to HFS are also encoded in EBCDIC.

However, source files that are stored in distributed Git repositories are usually encoded in UTF-8. When round-trip conversion is mentioned in this documentation, it refers to the following process:



Convert character set from EBCDIC to UTF-8 when it is committed to Git, and

Convert it back from UTF-8 to EBCDIC when it is loaded to HFS from the Git repository

There are some situations where this round-trip conversion does not preserve the original content of the source files. This is often referred to as non-roundtripable character situations. Files that contain non-roundtripable characters are typically transferred as binary to Git, so no codepage conversion occurs.

Also, non-printable characters can be detected by the migration script, by specifying the `-np` parameter. Depending on the value of this parameter, the migration script will issue a message and will flag the files as text (performing codepage conversion) or binaries. This non-printable scanning capability is available in DBB 1.1.1 (which is not yet installed in the current Lab image).

Details at: <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>

2.3.3 Verify that the file tags are correctly set, so that the rocket git client can correctly convert the files between EBCDIC and UTF-8. Please issue command

▶▶ ls -lT cobol/*

```
/u/ibmuser/migration-repository> ls -lT cobol/*
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      14003 Aug 13 10:06 cobol/lgacdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      11106 Aug 13 10:07 cobol/lgacdb02.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9286 Aug 13 10:07 cobol/lgacus01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      6076 Aug 13 10:07 cobol/lgacvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      25475 Aug 13 10:07 cobol/lgapdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      10243 Aug 13 10:07 cobol/lgapo101.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9071 Aug 13 10:07 cobol/lgapvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      11643 Aug 13 10:08 cobol/lgdpdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9872 Aug 13 10:08 cobol/lgdpo101.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      6671 Aug 13 10:08 cobol/lgdpv01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      10875 Aug 13 10:08 cobol/lgicdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      8320 Aug 13 10:08 cobol/lgicus01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9747 Aug 13 10:08 cobol/lgicvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      48242 Aug 13 10:09 cobol/lgipdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      8283 Aug 13 10:09 cobol/lgipo101.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      7226 Aug 13 10:09 cobol/lgipvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      22222 Aug 13 10:09 cobol/lgsetup.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      5768 Aug 13 10:09 cobol/lgstsq.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9070 Aug 13 10:10 cobol/lgucdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9281 Aug 13 10:10 cobol/lgucus01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      6812 Aug 13 10:10 cobol/lgucvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      24889 Aug 13 10:10 cobol/lgupdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      11048 Aug 13 10:10 cobol/lgupo101.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9855 Aug 13 10:10 cobol/lgupvs01.cbl
/u/ibmuser/migration-repository>
```

2.3.4 Use the git status command, to check the current state of the repository.

►| **git status**

```
/u/ibmuser/migration-repository>
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

.gitattributes
cobol/
copy/
map/

nothing added to commit but untracked files present (use "git add" to track)
/u/ibmuser/migration-repository>
```

2.4 Finishing migration process

Before pushing to the Git server, you need to add the newly copied files to the local Rocket git and also commit the changes.

2.4.1 Add the newly copied files to the local Git repository database by navigating to the local Rocket git and perform a Git 'add'.

►| **git add .**

```
/u/ibmuser/migration-repository>
git add .
/u/ibmuser/migration-repository>
```

2.4.2 Commit the changes specifying an appropriate commit message.

►| **git commit -m "Migrate source libraries to Git"**

```
/u/ibmuser/migration-repository>
git commit -m "Migrate source libraries to Git"
[master (root-commit) be098a6] Migrate source libraries to Git
 29 files changed, 8775 insertions(+)
 create mode 100644 .gitattributes
 create mode 100644 cobol/lgacdb01.cbl
 create mode 100644 cobol/lgacdb02.cbl
 create mode 100644 cobol/lgacus01.cbl
 create mode 100644 cobol/lgacvs01.cbl
 create mode 100644 cobol/lgapdb01.cbl
 create mode 100644 cobol/lgapo101.cbl
 create mode 100644 cobol/lgapvs01.cbl
 create mode 100644 cobol/lgdpdb01.cbl
 create mode 100644 cobol/lgdpo101.cbl
 create mode 100644 cobol/lgdrys01.cbl
 create mode 100644 cobol/lgupvs01.cbl
 create mode 100644 copy/lgcarea.cpy
 create mode 100644 copy/lgpolicy.cpy
 create mode 100644 copy/ssmap.cpy
 create mode 100644 map/ssmap.bms
/u/ibmuser/migration-repository>
```

2.4.3 Use the git status command, to check the current state of the repository.

```
git status
On branch master
nothing to commit, working tree clean
/u/ibmuser/migration-repository>
```

Section 3. Push the migrated files to Git server (Gitlab)

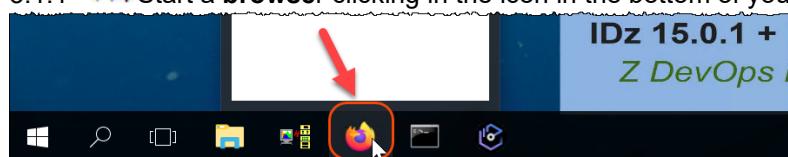
When all of the source files have been copied to the local Git repository, you are ready to push the files to the Git server.

In our Lab we will create a new Gitlab repository and push the migrated code to this newly created Git server repository.

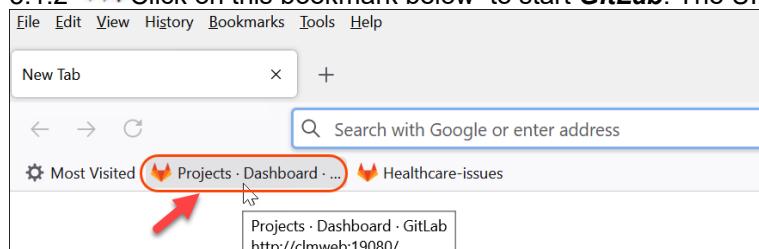
3.1 Creating a new Gitlab repository

Usually, the Git server repository is already created, but we will create a new one from scratch. Once done you will push to this repository the Rocket Git client that you created

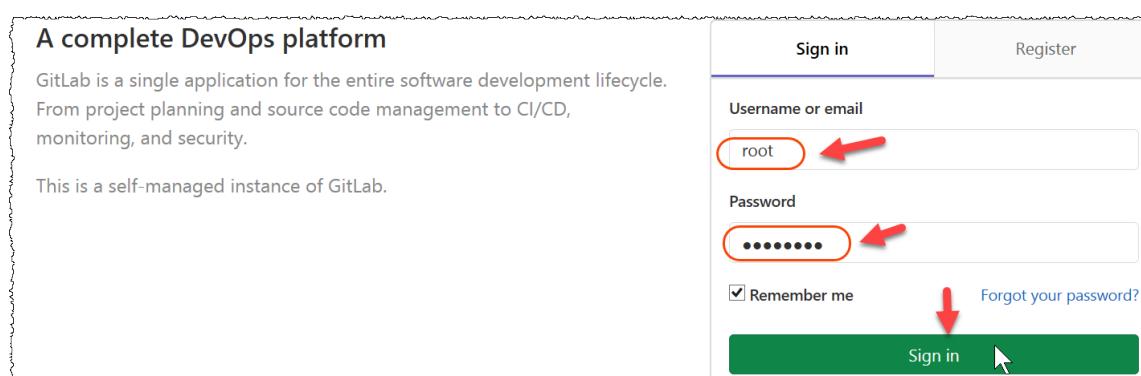
3.1.1 ► Start a browser clicking in the icon in the bottom of your screen



3.1.2 ► Click on this bookmark below to start **GitLab**. The URL used is <http://clmweb:19080/>



3.1.3 ► If ask for credentials use **root** and password: **zdtlinux**



3.1.4 ➡ To create a new repository click **New project**

The screenshot shows the GitLab 'Projects' dashboard. At the top right, there is a green button labeled 'New project'. A red arrow points to this button, and a red circle highlights it. The dashboard includes a header with 'File Edit View History Bookmarks Tools Help', a navigation bar with 'Projects - Dashboard - GitLab', and a search bar at the top right.

3.1.5 ➡ Type **Migrated from Z** as Project name and click **Create project**

The screenshot shows the 'New project' creation page. The 'Project name' field contains the text 'Migrated from Z' (circled with a red number 1). At the bottom left, there is a green 'Create project' button (circled with a red number 2). A red arrow points to this button. The page also includes sections for 'Blank project', 'Create from template', 'Import project', 'Project URL', 'Project slug', 'Project description (optional)', and 'Visibility Level' (with 'Private' selected).

3.1.6 The new repository is created but empty. Notice on the command instructions ways to populate it. You will use the instructions named “Push an existing Git repository”

Project 'Migrated from Z' was successfully created.

Migrated from Z

Project ID: 15

The repository for this project is empty

You can get started by cloning the repository or start adding files to it with one of the following options.

Clone

Command line instructions

You can also upload existing files from your computer using the instructions below.

Git global setup

```
git config --global user.name "Administrator"
git config --global user.email "admin@example.com"
```

Create a new repository

```
git clone git@clmweb:root/migrated-from-z.git
cd migrated-from-z
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Push an existing folder

```
cd existing_folder
git init
git remote add origin git@clmweb:root/migrated-from-z.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@clmweb:root/migrated-from-z.git
git push -u origin --all
git push -u origin --tags
```

3.1.6 You will need to execute that git commands on the UNIX shell.
A suggested way is to open a notepad and copy/paste those commands.
But we also provided that commands below

```
git remote add origin git@clmweb:root/migrated-from-z.git
git push -u origin --all
git push -u origin --tags
```

3.2 Connect the local Rocket Git repository with the Gitlab repository

You will connect the local Rocket Git repository with the Gitlab server repository that you created, by adding the remote origin. Once done you can push to Gitlab server

3.2.1 ► Go back to IDz using the icon that is on the base of your screen:



3.2.2 Using the IDz UNIX shell execute the commands as below

► `git remote add origin git@clmweb:root/migrated-from-z.git`

```
/u/ibmuser/migration-repository> git remote add origin git@clmweb:root/migrated-from-z.git
/u/ibmuser/migration-repository>
```

3.3 Push the existing Rocket Git client repository to the Git server repository

You will now push to Gitlab server the Rocket Git repository that you created.

3.3.1 Using the IDz UNIX shell execute the commands as below

► `git push -u origin --all`

```
/u/ibmuser/migration-repository> To clmweb:root/migrated-from-z.git
  git push -u origin --all
    * [new branch] master -> master
      Branch master set up to track remote branch master from origin.
/u/ibmuser/migration-repository>
```

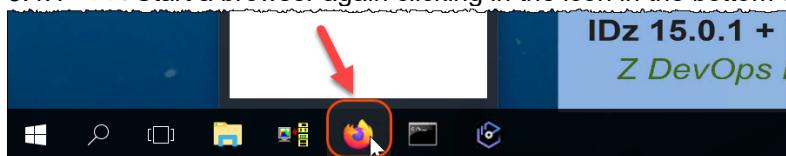
► `git push -u origin --tags`

```
/u/ibmuser/migration-repository> Everything up-to-date
  git push -u origin --tags
/u/ibmuser/migration-repository>
```

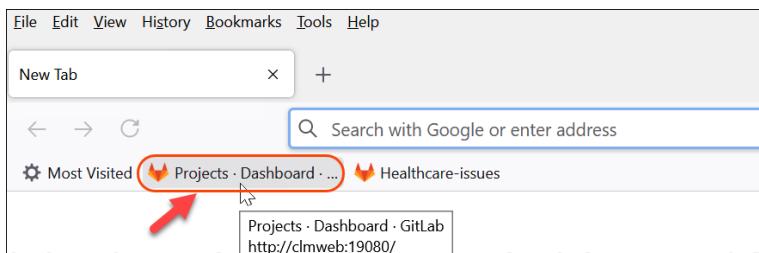
3.4 Access Gitlab repository to verify the code migrated

You can now access the Gitlab server repository to verify the migrated code.

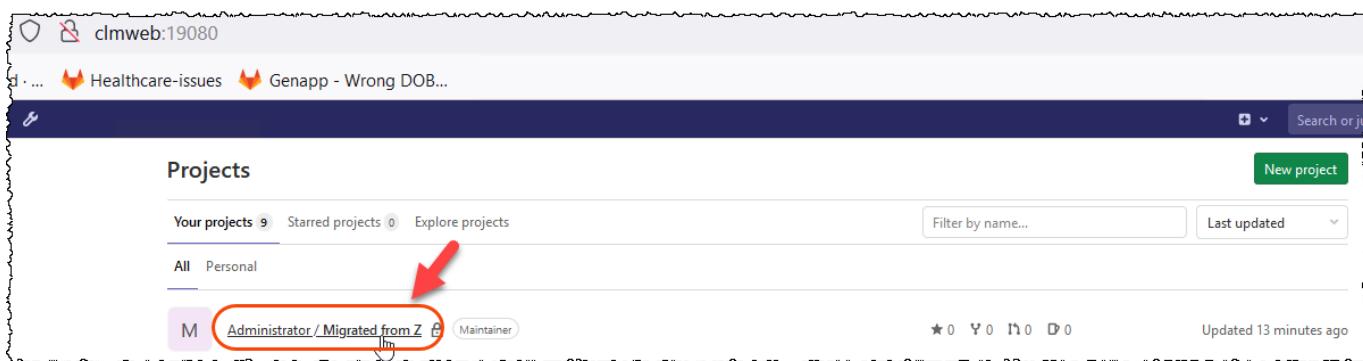
3.4.1 ► Start a browser again clicking in the icon in the bottom of your screen



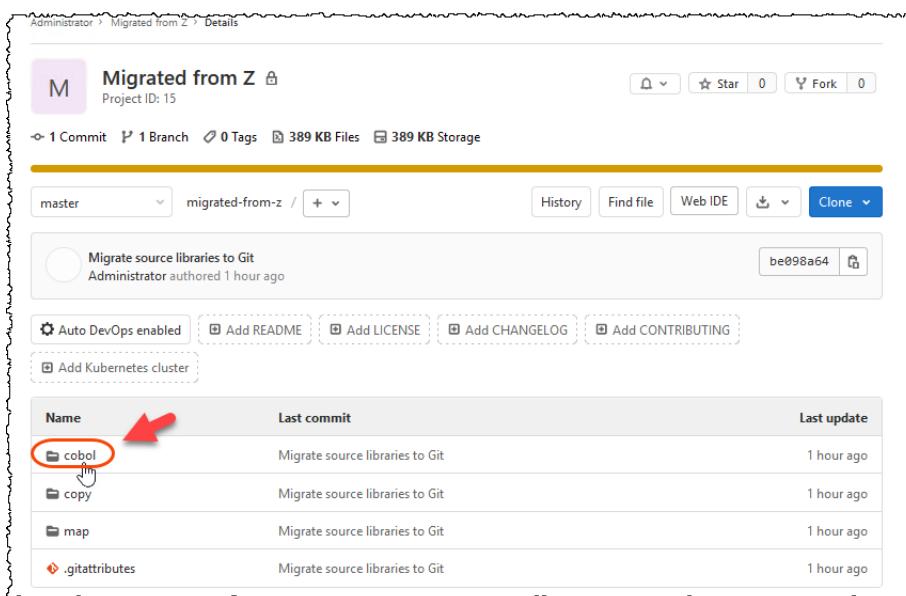
3.4.2 ➡ Click on this bookmark below



3.4.3 ➡ Click on the repository that you created



3.4.4 ➡ Click on cobol folder



3.4.5 ➡ Click on **lgacdb01.cbl** file

Administrator > Migrated from Z > Details

master migrated-from-z / cobol / +

History Find file Web IDE Clone

Migrate source libraries to Git
Administrator authored 1 hour ago
be098a64

Name	Last commit	Last update
..		
lgacdb01.cbl	Migrate source libraries to Git	1 hour ago
lgacdb02.cbl	Migrate source libraries to Git	1 hour ago

3.4.6 And verify the content.

Administrator > Migrated from Z > Repository

master migrated-from-z / cobol / **lgacdb01.cbl**

Adm Migrate source libraries to Git
Administrator authored 1 hour ago

```

lgacdb01.cbl 13.7 KB Edit
1 ****
2 * Changed LGACDB01 Dec 18, 2020 15:13
3 *
4 * ADD Customer Details
5 *
6 * To add customer's name, address and date of birth to the
7 * DB2 customer table creating a new customer entry.
8 *
9 * @c3
10 ****
11 IDENTIFICATION DIVISION.
12 PROGRAM-ID. LGACDB01.
13 ENVIRONMENT DIVISION.
14 CONFIGURATION SECTION.
15 *
16 DATA DIVISION.
17 *
18 WORKING-STORAGE SECTION.
19 
```

At this point the migration to Git is complete...

What have you done so far?

Using the DBB migration utilities you migrated a complete application from z/OS PDS members to Gitlab.



Notice that you first created the Rocket Git client and pushed to **Gitlab**. Other way would be cloning an existing Git to Rocket Git.

For details see the documentation:
<https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>

On the next section we show how to load the migrated code from Gitlab to your local IDz workspace. Notice that this is not part of the migration but the next logical step to start working with your code.

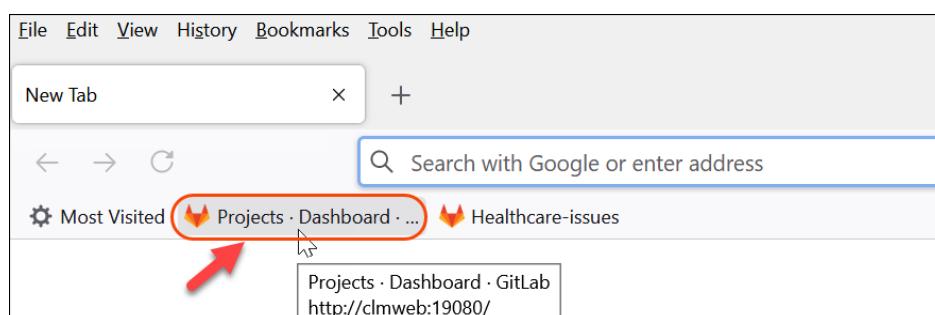
Section 4. Load the source code from Git to the local IDz workspace and start working with the migrated assets

On this section we show how to load the migrated code from Gitlab to your local IDz workspace. Notice that this is not part of the migration but the next logical step to start working with your code.

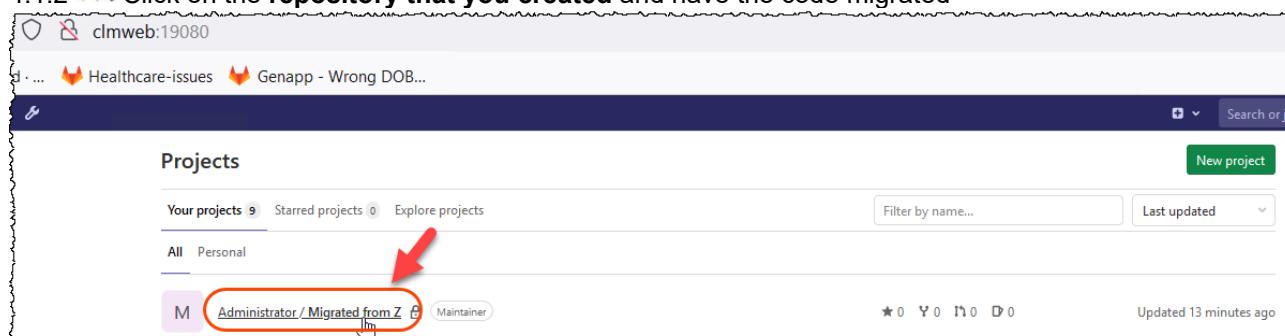
4.1 Cloning the Git server repository using IDz

Usually, the Git server repository that you migrated the code you will clone to your IDz client windows.

4.1.1 ► Using the web browser click on **this bookmark** below



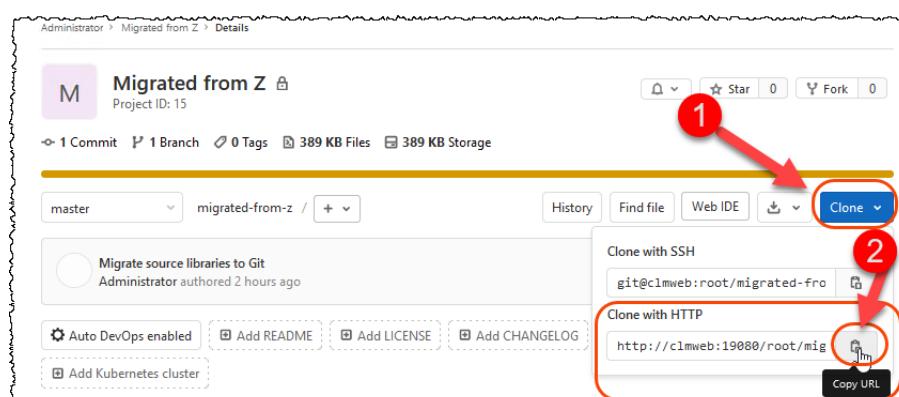
4.1.2 ► Click on the **repository that you created** and have the code migrated



4.1.3 In order to clone it at your window desktop you could use **HTTP** or **SSH**.

Since **SSH** is blocked in our environment we need to use **HTTP**.

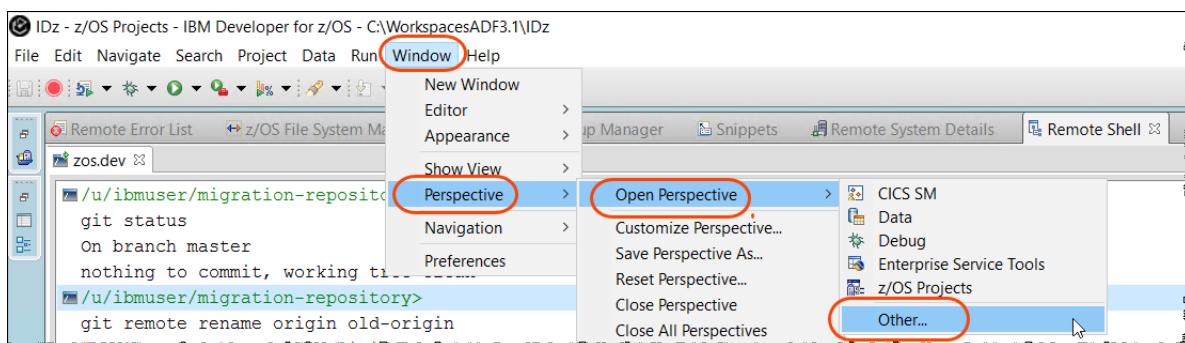
► ① Click on **Clone** (the blue button) and ② in the icon to **copy the URL**. This value will be kept in the windows clipboard.



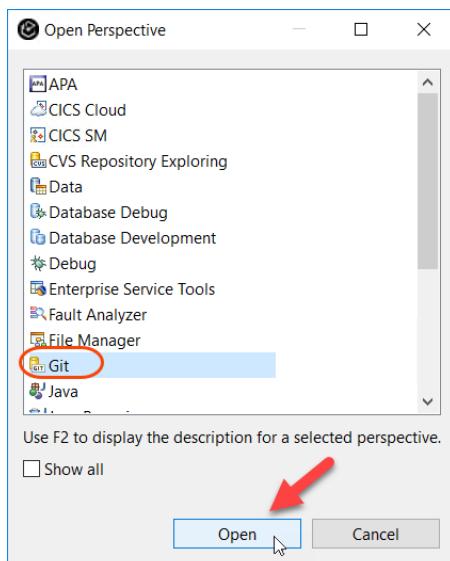
4.1.4 ► Go back to IDz using the icon  that is on the base of your screen:



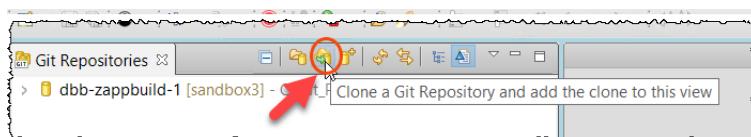
4.1.5 ► Open the Git perspective by selecting
Window > Perspective > Open Perspective > Other...



4.1.6 ► Select Git and click Open



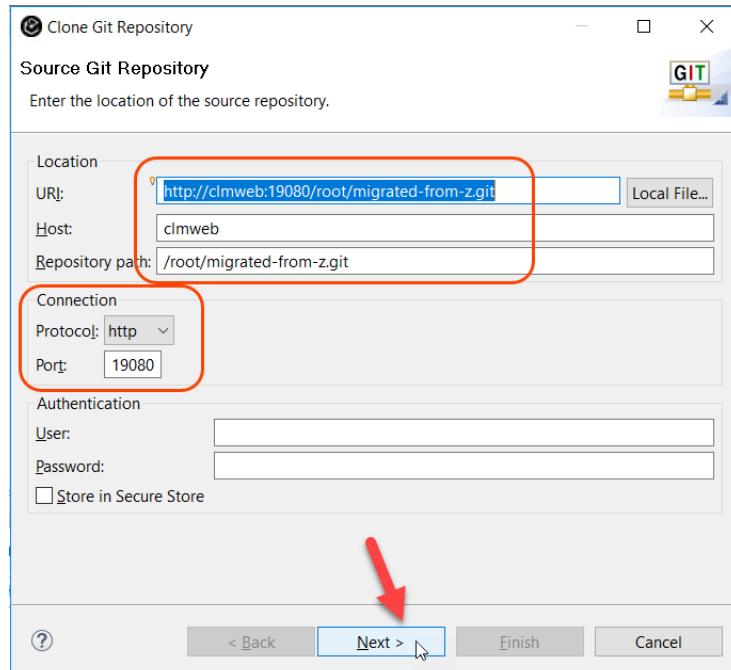
4.1.7 ► In the *Git Repositories* tab, click the  to Clone a Git Repository



4.1.8 ► The values copied from the web page (*copy URL*) will be shown in the Clone Git Repository.

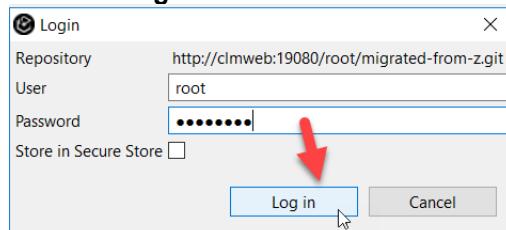
Tip: In case you don't see it, got back to the page and copy it again (steps 4.1.3 above).

► Click **Next**



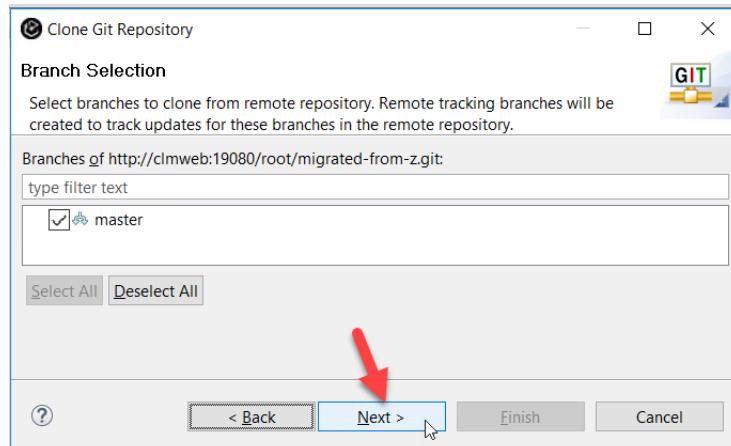
4.1.9 For credentials use: **root** and password **zdtlinux**

► Click **Log in**



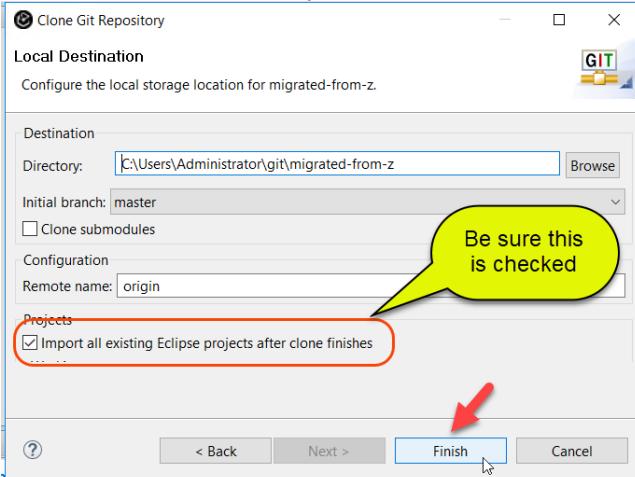
4.1.10 We have only one branch named **master**.

► Click **Next**



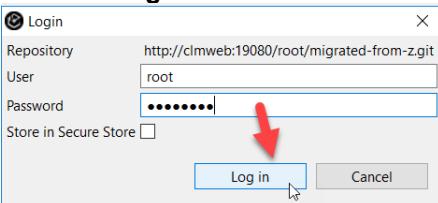
4.1.11 We will clone the repository on your local windows and import to be shown on IDz
You could find a specific folder to hold the cloned repository.

► Click **Finish** and accept the default location.

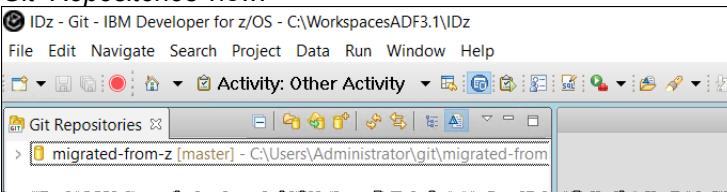


4.1.12 If credentials are asked again use: **root** and password **zdtlinux**

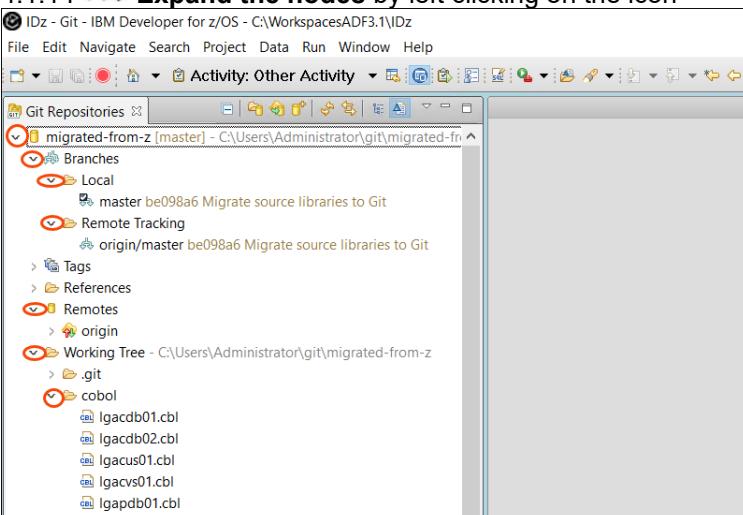
► Click **Log in**



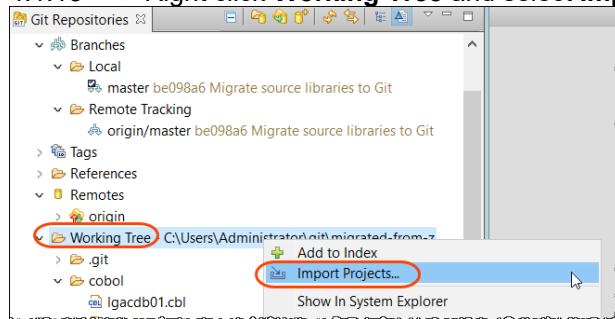
4.1.13 The migrated repository was cloned from the Remote master repository and will appear in the **Git Repositories** view.



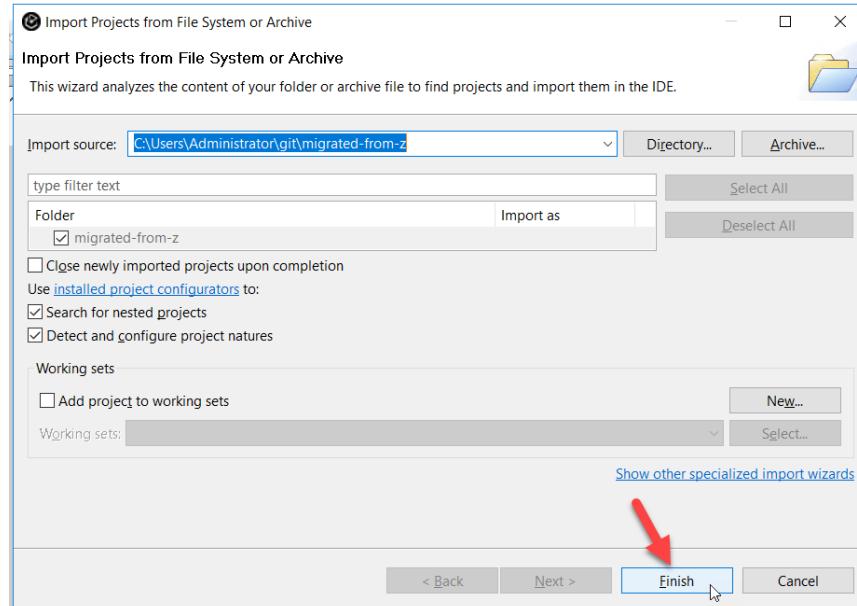
4.1.14 ► Expand the nodes by left clicking on the icon ▶ as shown below:



4.1.15 ➡ Right click Working Tree and select Import Projects...



4.1.16 ➡ Click Finish



Why the imported project will not show in the z/OS Projects Perspective?

IDz projects may have different functionalities. The project that is loaded from Git must have specific functionality to be seen on **z/OS Projects perspective**. This functionality is called "**Project Natures**".

There are at least 2 ways to add the required "Project Natures":

1. Using IDz as described on this exercise (step 4.2.4).
2. Manually creating a file name ".project" with some specific content.

In this lab we will use the #1 method above. But if you prefer adding **.project** manually just create a file with this name and content as below.

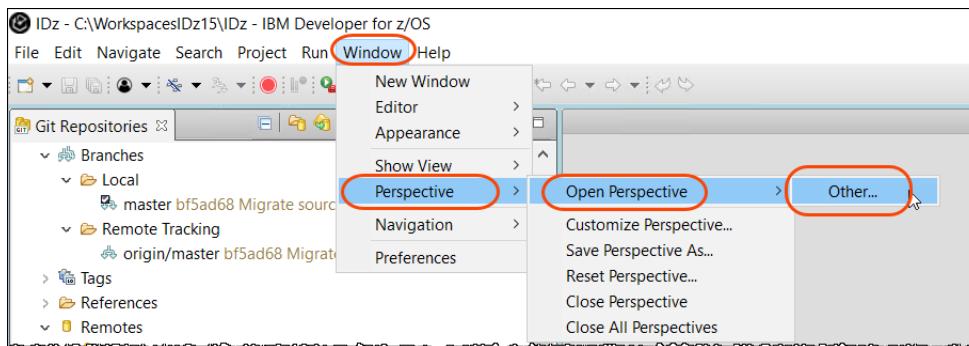
```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>genapp-migration</name>
    <comment></comment>
    <projects>
        </projects>
        <natures>
            <nature>com.ibm.ftt.dbbz.integration.dbbzprojectnature</nature>
            <nature>com.ibm.ftt.ui.views.project.navigator.local</nature>
        </natures>
    </projectDescription>
Details at https://www.ibm.com/docs/en/adfz/developer-for-zos/15.0.0?topic=zos-setting-up-developer-dbb-integration
```

4.2 Verify the code cloned using z/OS projects perspective

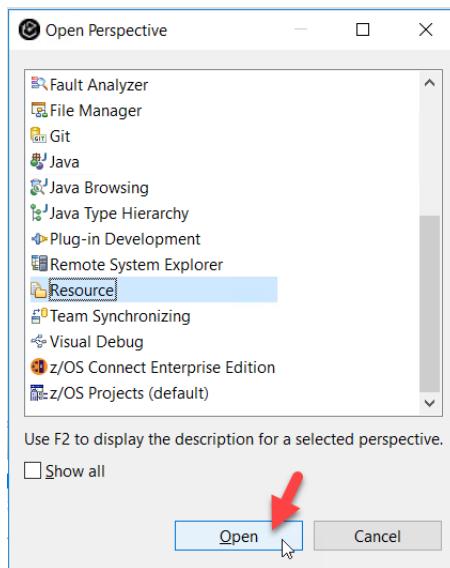
The z/OS projects perspective is the IDz perspective that developers use to work with the source code. Notice that you have cloned the project at your local workstation, but this will not be automatically moved to the Z/OS Projects perspective. You need to create a file named .project with the correct contents..

The easy way to do that is described below..

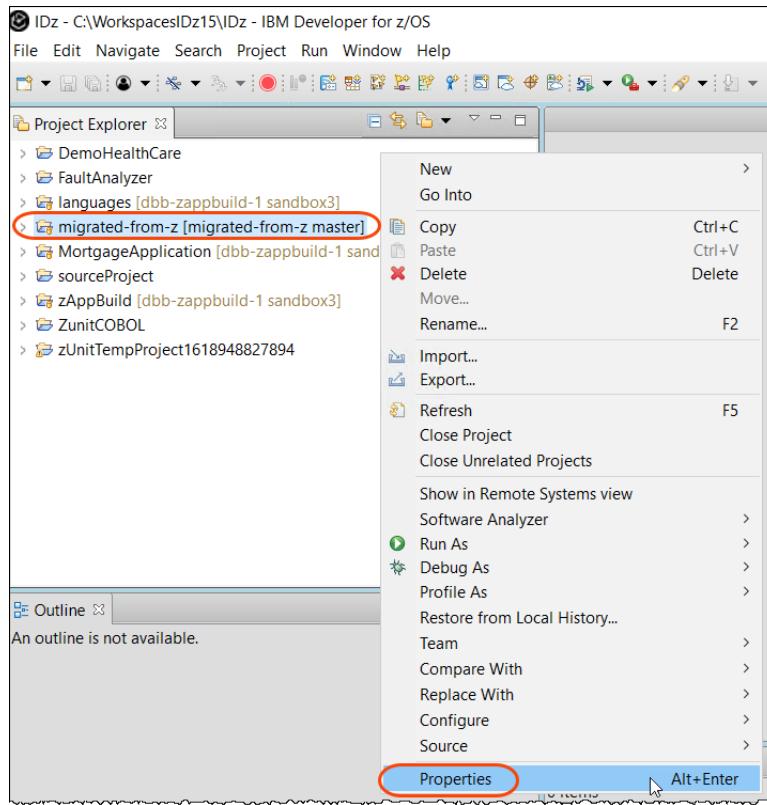
- 4.2.1 ► Switch to the **Resource Perspective** by selecting
Window > Perspective > Open Perspective > Other...



- 4.2.2 ► Scroll down, select **Resource** and click **Open**

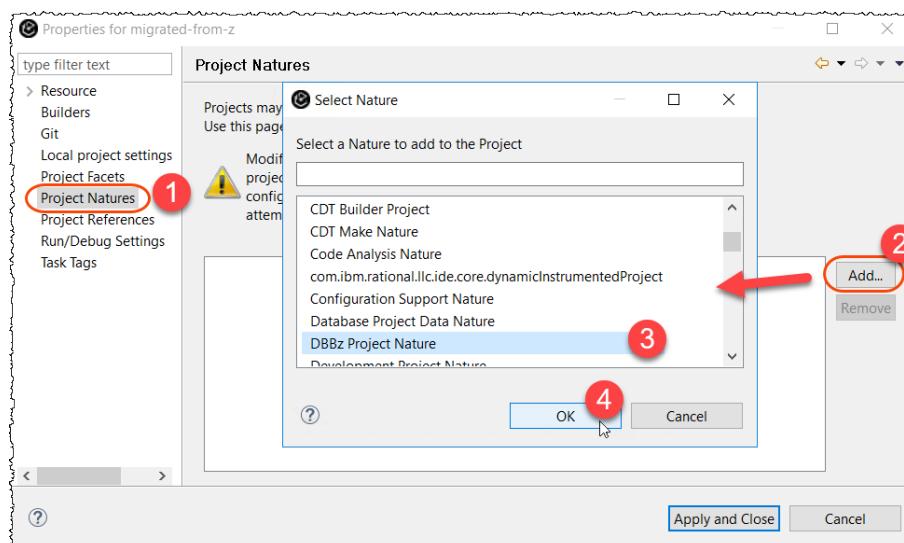


4.2.3 ► Right click **migrated-from-z** and select **Properties**

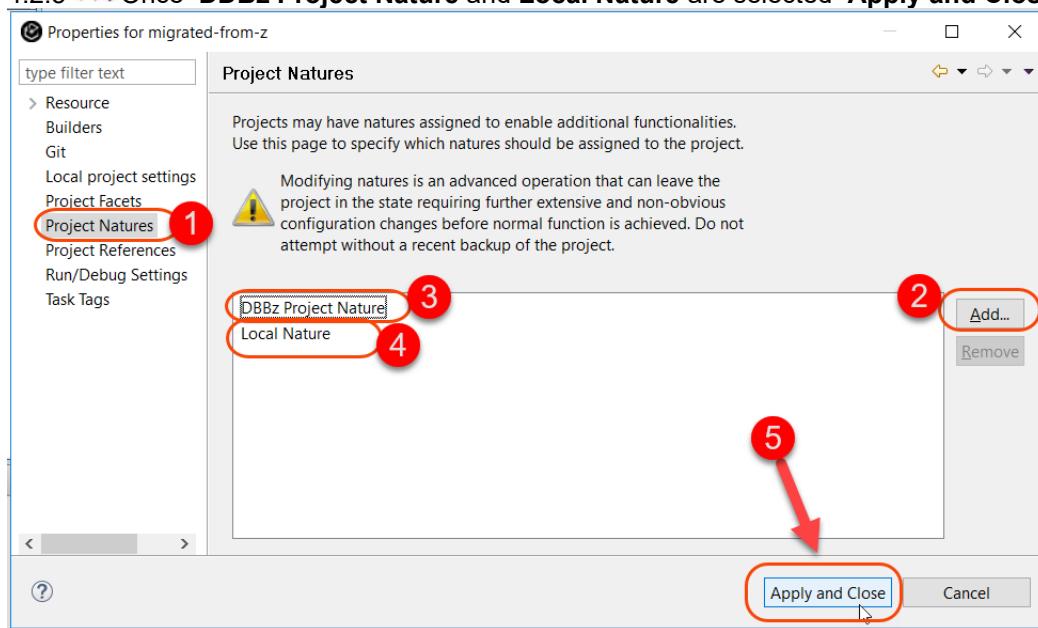


4.2.4 ► Select **Project Natures**, click **Add...** and **OK**

► This brings up a *Select Nature* dialog where you have to add the **DBBz Project Nature** and **Local Nature** at 2 different times.



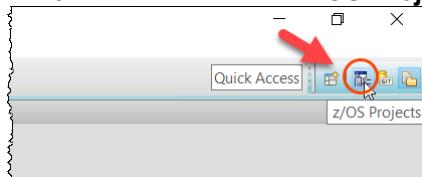
4.2.5 ► Once DDBz Project Nature and Local Nature are selected **Apply and Close**



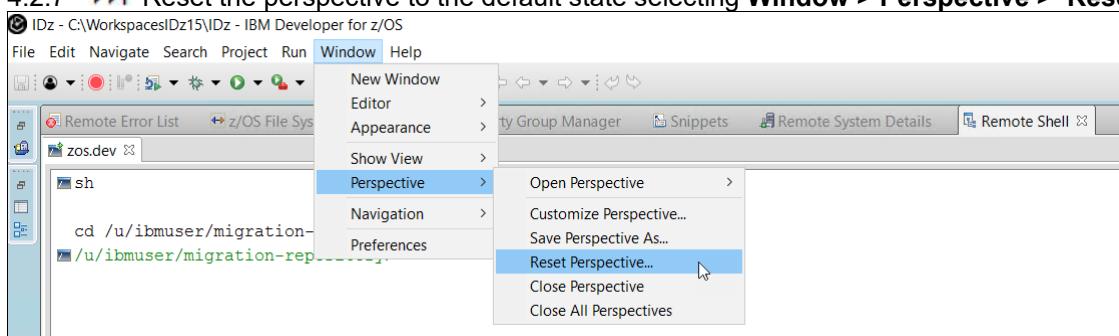
This will create the file named `.project` under the Project where the content will be as below.. Notice that this file is hidden in the project

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>genapp-migration</name>
    <comment></comment>
    <projects>
    </projects>
    <natures>
        <nature>com.ibm.ftt.dbbz.integration.dbbzprojectnature</nature>
        <nature>com.ibm.ftt.ui.views.project.navigator.local</nature>
    </natures>
</projectDescription>
```

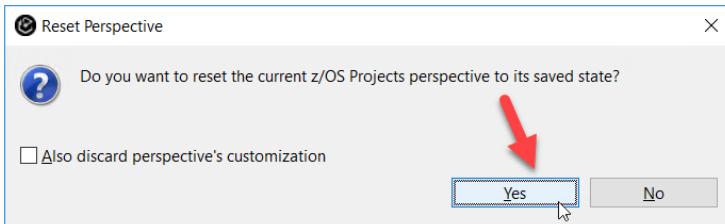
4.2.6 ► Return to the z/OS Projects Perspective



4.2.7 ► Reset the perspective to the default state selecting **Window > Perspective > Reset Perspective**

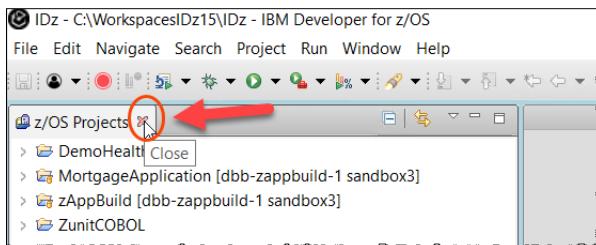


4.2.8 ➡ Click **Yes** to reset the perspective to the saved state

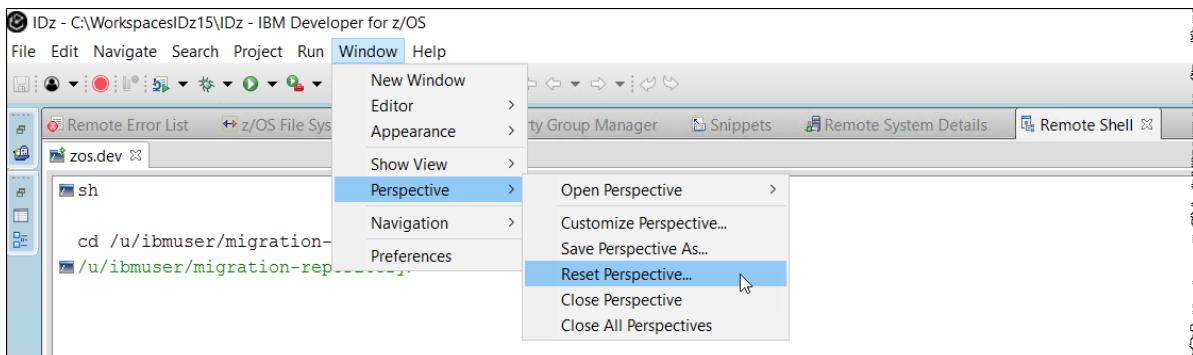


4.2.9 Notice that the project is still not showing there.

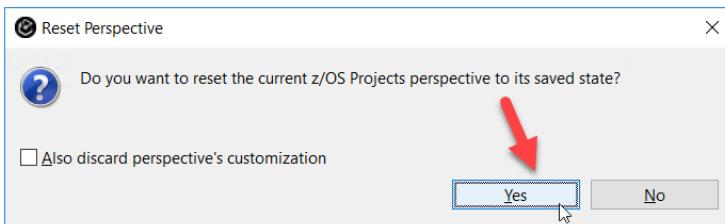
➡ Close the z/OS Projects



4.2.10 ➡ Perform the Reset Perspective again



4.2.11 ➡ Click **Yes** to reset the perspective to the saved state.



4.2.12 You will have the Project loaded now

▶▶ Expand the project and double click on **lgacdb01.cbl** to see the migrated content.

IDz - migrated-from-z/cobol/lgacdb01.cbl - C:\Workspaces\IDz15\IDz - IBM Developer for z/OS

File Edit Source Refactor Navigate Search Project Run Window Help

z/OS Projects

- > DemoHealthCare
- > migrated-from-z [migrated-from-z master]
 - .gitattributes
 - .project
 - cobol
 - lgacdb01.cbl
 - lgacdb02.cbl
 - lgacus01.cbl
 - lgacvs01.cbl
 - lgapdb01.cbl
 - lgapol01.cbl
 - lgapvps01.cbl
 - lgdpdb01.cbl
 - lgdpol01.cbl
 - lgdpvps01.cbl
 - lgicdb01.cbl
 - lgicus01.cbl

lgacdb01.cbl

```

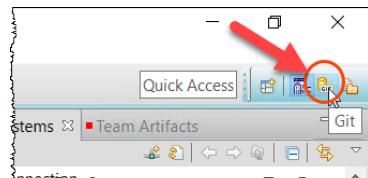
-----+*A-1-B-----2-----3-----4-----5-----+
***** Changed LGACDB01 Dec 18, 2020 15:13
*****
* ADD Customer Details
*
* To add customer's name, address and date of b
* DB2 customer table creating a new customer ent
*
* @c3
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. LGACDB01.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
*
DATA DIVISION.
*
WORKING-STORAGE SECTION.

```

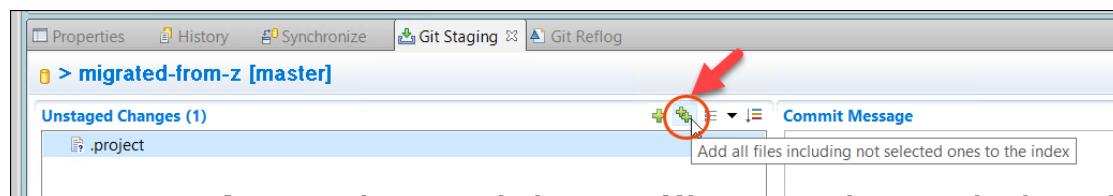
4.3 Commit the update to the Git server repository

The file ***.project** created on IDz workspace must be staged and committed to the Git server.

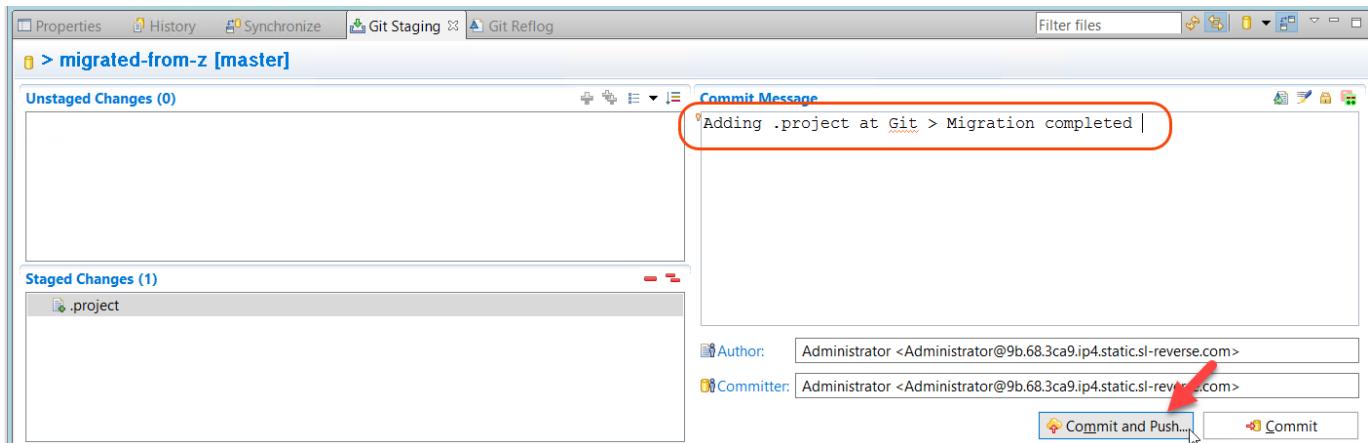
4.3.1 ▶▶ Switch to the **Git Perspective** by selecting the icon below



4.3.2 ▶▶ Using the **Git Staging** view click **+** to stage the update.

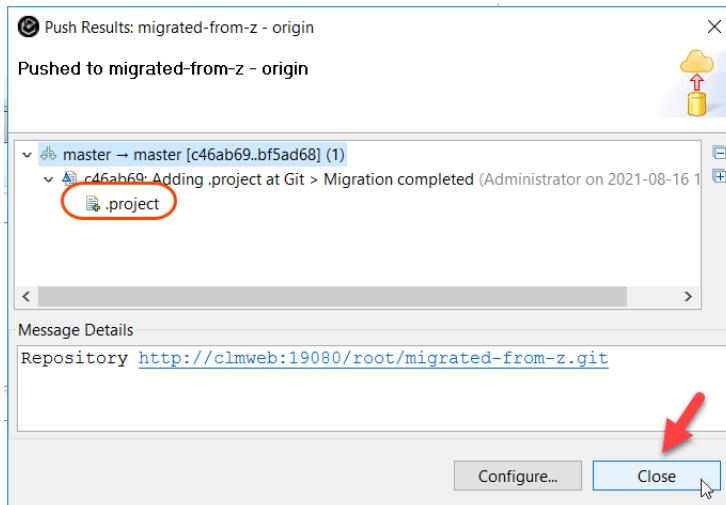


4.3.3  Add a message like “Adding .project at Git > Migration completed” and click **Commit and Push...**...



4.3.4 If asks for login use **root** and **zdtlinux**

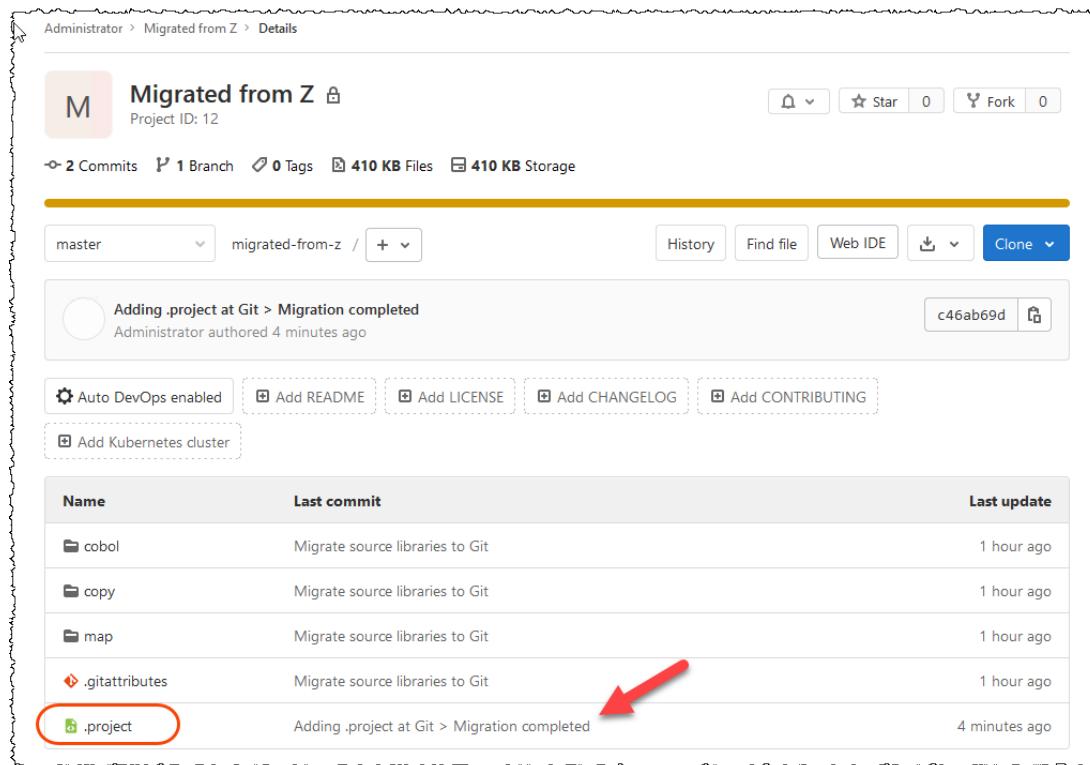
 Click **Close**



4.3.5  Start a browser again clicking in the icon  in the bottom of your screen



4.3.6  Using the browser press **F5** to refresh the browser and verify that the file **.project** is now at the Git repository



The screenshot shows a Git repository named "Migrated from Z" with Project ID 12. It has 2 commits, 1 branch, 0 tags, 410 KB files, and 410 KB storage. The master branch is selected. A recent commit is shown:

Name	Last commit	Last update
cobol	Migrate source libraries to Git	1 hour ago
copy	Migrate source libraries to Git	1 hour ago
map	Migrate source libraries to Git	1 hour ago
.gitattributes	Migrate source libraries to Git	1 hour ago
project	Adding .project at Git > Migration completed	4 minutes ago

A red arrow points to the "Adding .project at Git > Migration completed" commit message. The "project" row is highlighted with a red circle.

Congratulations! You have completed the Lab 11.

LAB 12 -(OPTIONAL) Application Discovery : Find a candidate function in a CICS application in order to create an API

Updated August 17, 2021 by David H Reviewed by Regi

Overview of development tasks

Use IBM® Application Discovery and Delivery Intelligence (ADDI) to discover CICS® programs that can be converted in to APIs.

With a rising demand for more and expanded customer accessibility and functionality, exposing and leveraging APIs has become a critical component of the modern IT business model. The first step in this process is discovering what potential API candidates exist in your codebase.

This scenario guides you through the discovery process in roughly 30 minutes. By the end of the session, you will know how to perform the following tasks:

- Use IBM ADDI to understand the architecture of a CICS application.
- Create graphs to find individual components of the application that would be good API candidates.
- View a copybook to understand parameters for mapping the API.

No previous knowledge of application architecture or programming is needed, but some awareness of CICS and API-specific terminology might help.

The main tasks that you will perform are:

1. **Make sure that you have registered for an ADDI Instance**
2. **Open your ADDI instance**
3. **Start the ADDI Discover API Scenario**
4. **Follow the workshop instructions provided within the z Trial instance**

The instructions in this document are just related to starting your ADDI zTrial instance.

Once the ADDI zTrial workshop is started you will be following the instructions provided within the workshop Wizard.



Each time you see a symbol (in this document) it means that you have to “do” something on your computer – not merely read the document.

Tips!

- 1) The labs will perform on your personal IBM's ADDI zTrial Cloud instance
- 2) This zTrial instance will only be available for 3 days, starting on (including) the day that you received the email that your zTrial was ready.
- 3) If your access expires before you complete this workshop, you can request another instance from the following link <https://www.ibm.com/it-infrastructure/z/resources/trial>. You can not extend the 3 day limit.

Section 1 Request the ADDI Z Trial Lab

(Assumed that you have completed this on day 1)

– Be sure that you have registered for an ADDI zTrial instance and have received your credentials to log in via an email from “IBM Z Trial”

Because the provisioning of an ADDI zTrial instance can take up to 3 hours, the instruction on how to register and get your ADDI zTrial environment were provided on day 1 introduction. If you have not requested your zTrial instance yet, here are the instructions

- Request an ADDI zTrial instance

1. Open IBM zTrial registration page <https://www.ibm.com/it-infrastructure/z/resources/trial>
2. Click on Register for the trial of the
3. Select the Updated! IBM Application Discovery and Delivery Intelligence

The screenshot shows the "IBM Z software trials" landing page. At the top, there's a dark header with the title "IBM Z software trials". Below it, a sub-header says "Try IBM Z® software at no charge and with no installation. Your hands-on trial is available within 2 hours for 5 days (including weekends)." The main content area is divided into four categories:

- Cloud, DevOps and API integration software trials** (with a yellow box highlighting "Updated! IBM z/OS® Connect Enterprise Edition")
- New! IBM Z Development and Test Environment** (with a yellow box highlighting "New! IBM Application Performance Analyzer for z/OS®")
- New! Bring Your Own (BYO) IDE for Cloud Native Development** (with a yellow box highlighting "Updated! IBM Application Discovery and Delivery Intelligence")
- Updated! IBM Cloud Provisioning and Management for z/OS**

Each category has a brief description, a "Learn how to:" section with bullet points, and two blue links: "Register for the trial" and "Explore the product first".

Category	Description	Learn how to:	Action Links
Updated! IBM z/OS® Connect Enterprise Edition	Create efficient and scalable RESTful APIs for mobile and cloud applications.	<ul style="list-style-type: none"> * Create REST APIs to: CICS® programs IIMS® transactions Db2® for z/OS data IBM® MQ Queues. * Call a REST API from a COBOL application. 	<ul style="list-style-type: none"> → Register for the trial → Explore the product first
New! IBM Application Performance Analyzer for z/OS®	Quickly identify and act upon areas of low performance, high CPU consumption, low response time, and low throughput in your most critical z/OS applications.	<ul style="list-style-type: none"> * Create, initiate, and analyze performance observations for a Java and COBOL batch program. * Exploit source-program mapping and then modify source code statements in a Java and COBOL program. * Create and review a Variance Report to compare the performance of an original and modified program. 	<ul style="list-style-type: none"> → Register for the trial → Explore the product first
New! Bring Your Own (BYO) IDE for Cloud Native Development	With this trial of IBM Wazi Developer or IBM Developer for z/OS® Enterprise Edition you can choose your preferred IDE (Eclipse® or Microsoft® VS Code™) integrated with familiar DevOps tools such as Git and Jenkins to develop a z/OS application. Learn how to:	<ul style="list-style-type: none"> * Use GitLab to see your assigned tasks and access application source code stored in Git. * Code, debug and build the sample COBOL application using Eclipse or VS Code. * Commit and push modifications to Git and request a Jenkins pipeline. 	<ul style="list-style-type: none"> → Register for the trial → Explore the product first
Updated! IBM Cloud Provisioning and Management for z/OS	Provision z/OS software subsystems with automated processes. Learn how to:	<ul style="list-style-type: none"> * Provision and deprovision a CICS® region with a cloud provisioning marketplace. * Create and publish a CICS region service to facilitate self-service provisioning. * Provision, deprovision an MQ queue with a cloud provisioning marketplace. 	<ul style="list-style-type: none"> → Register for the trial → Explore the product first
IBM SDK for Node.js - z/OS	Modernize apps, orchestrate services with Node.js and connect to your z/OS assets. Learn how to:		
IBM Application Delivery Foundation for z/OS	Integrated tools that help teams design, build, test, and debug z/OS software. Learn how to:		

4. Login or Create a new IBM ID

Fill in registration page

NOTE: **If you already have** an IBM account ID – please click log in here
And use your IBM account ID

OR

**If you do not have a free IBM account ID,
please fill in this info in order to create one.**

Start your free trial

Already have an IBM account? Log in

Email *

Email is required

First name *

Last name *

Company

Job title

Country or region * (?)

United States

After registration you, will see a registration confirmation then receive 2 emails over the next few hours

5. Look for the following 2 emails - Depending on system demand this could take anywhere from 15 minutes up to 3 hours.

- You will receive 1 of 2 emails

[EXTERNAL] Thank you for registering
IBM Z Trial to: dhawrel
Cc: IBM Z Trial

From: "IBM Z Trial" <ztrial@uk.ibm.com>
To: dhawrel@us.ibm.com,
Cc: IBM Z Trial <ztrial@uk.ibm.com>

05/24/2019 09:19 AM Hide Details

IBM

Thank you for registering

Hi David,

We're delighted that you've decided to trial Application Discovery and Delivery Intelligence.

We're preparing your trial right now and will email you again once it's ready (usually within two hours).

Happy Trailng

— The IBM Z Trial team

Questions or feedback? Just reply to this email.

IBM

You will receive 2 of 2 emails to log into zTrial either through a Web browser or Remote desktop



To: dhawrel@us.ibm.com,
 Cc: IBM Z Trial <ztrial@uk.ibm.com>.
 Bcc:
 Subject: [EXTERNAL] Your trial of Application Discovery and Delivery Intelligence is ready
 From: "IBM Z Trial" <ztrial@uk.ibm.com> - Friday 05/24/2019 10:01 AM

Your trial is ready

Hi David,

We are happy to inform you that your Application Discovery and Delivery Intelligence trial is now ready for your use. The trial will be available for 3 days from the date of this email.

Accessing your trial

There are 2 ways to access your trial.

1. Via a web browser

You may connect to the trial environment via a web browser (we recommend Chrome or Firefox) using the sign-in details below.

Web browser sign-in details

URL: <https://169-60-90-66-T-4695.ibmztrialmachines.com/> 
 User name: Administrator
 Password: ?yourpswd?

We recommend viewing your trial in full screen mode.

2. Via remote desktop

To access your Windows-based environment, you must be able to connect to a remote system over a network connection.

- Follow the instructions in your "ZTrial is Ready" email to access your unique ADDI zTrial instance



IBM Application Delivery Foundation for Z - Your trial is ready

IBM Z Trial to: dhawrel

Cc: "IBM Z Trial"

From:

"IBM Z Trial" <ztrial@uk.ibm.com>

To:

dhawrel@us.ibm.com

Cc:

"IBM Z Trial" <ztrial@uk.ibm.com>

Security:

To ensure privacy, images from remote sites were prevented from downloading. [Show Images](#)

IBM Z Trial

Your trial is ready

Section 2 – Lab : Discover a candidate API and find its interface

In this section you will use IBM Application Discovery to discover the candidate API and its interface, identify the copybook that would be needed to make the API.

2.1 Open your ADDI zTrial

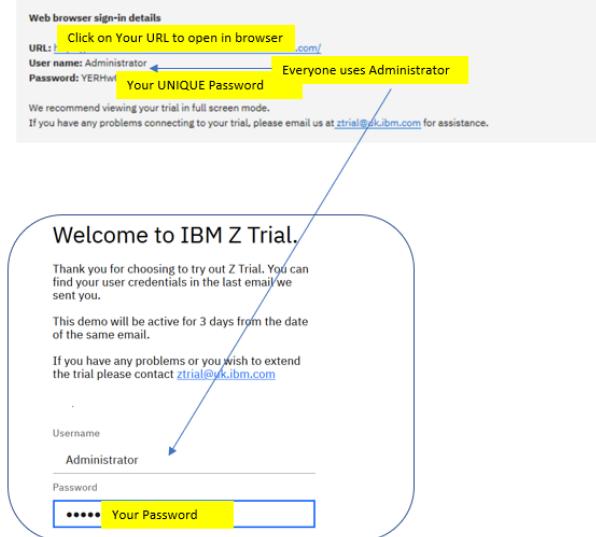
2.1.1 ► Open your ADDI zTrial instance

If cannot open your zTrial instance, call the instructor.

► As the instructions state in the email you received saying that your zTrial is ready
You can start your zTrial instance 1 of 2 ways

1 - Thru your Web browser

You may connect to the trial environment via a web browser (we recommend Chrome or Firefox) using the sign-in details below.



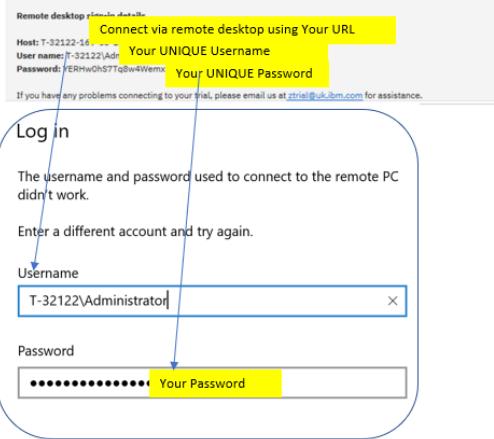
2 - Thru your Remote Desktop

To access your Windows-based environment, you must be able to connect to a remote system over a network connection.

Windows users should use the built-in [Remote Desktop Connection](#)

Mac users should use the [Microsoft Remote Desktop](#) app available from the App Store.

Linux users have several choices, which might vary between distribution.

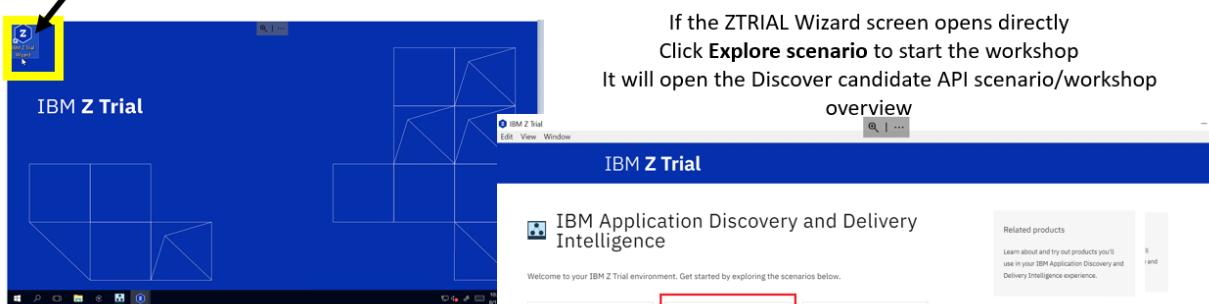


2.1.2 ► Start the ADDI Discover API Candidate wizard

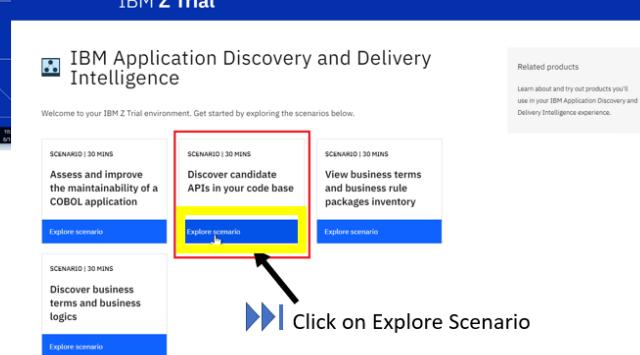
Depending on how you start your zTrial instance
One of the following screens will open

If this screen opens 1st

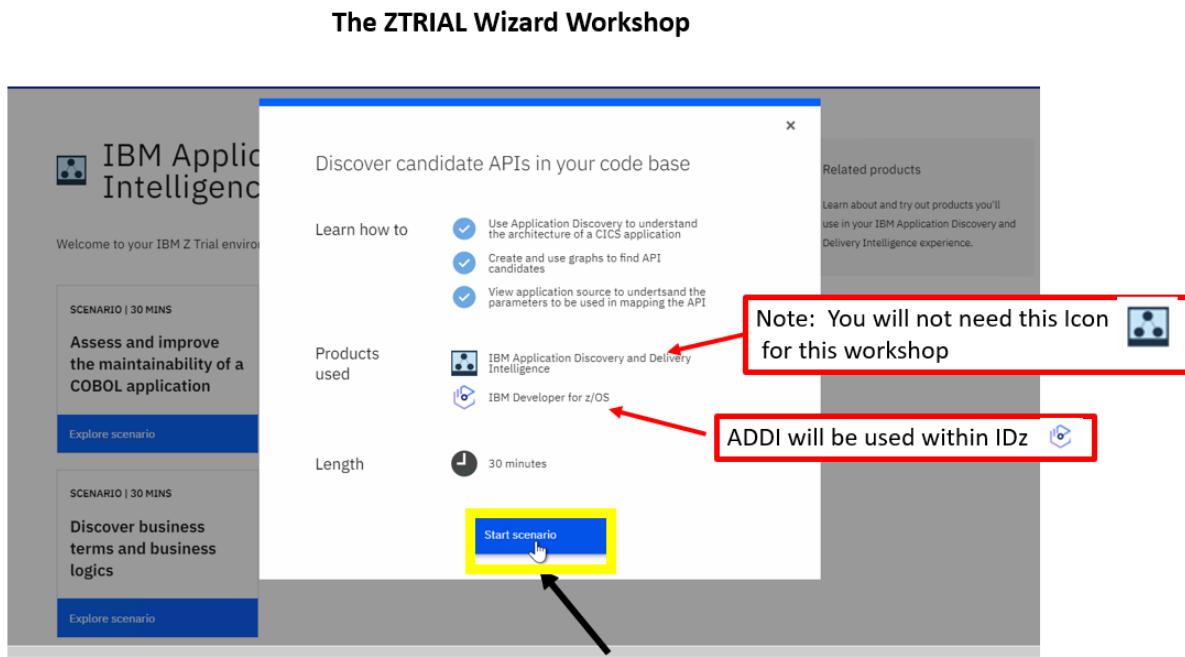
Click on the IBM Z Trial Wizard Icon to open the Z Trial Wizard screen



If the ZTRIAL Wizard screen opens directly
Click **Explore scenario** to start the workshop
It will open the Discover candidate API scenario/workshop overview



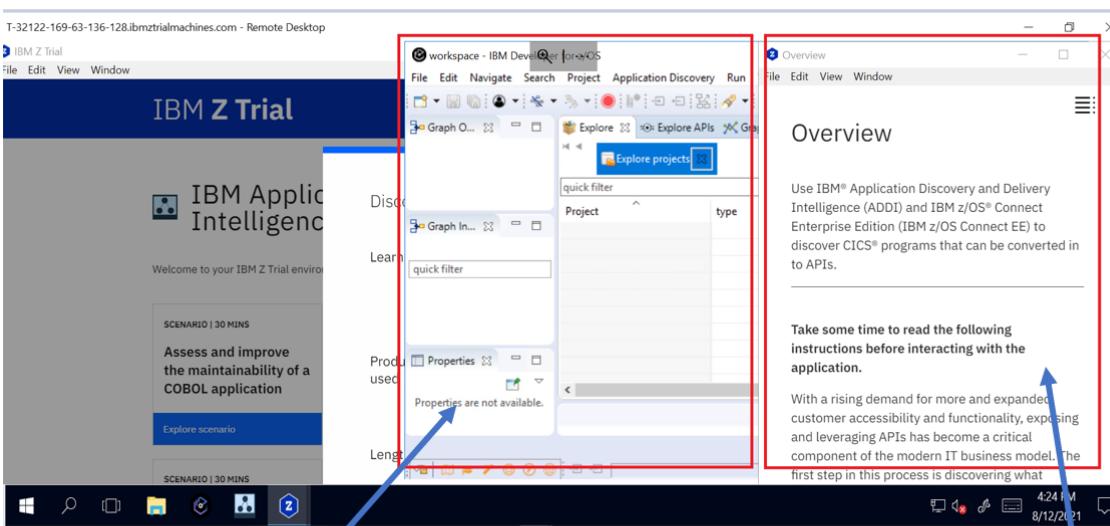
2.1.3 ➡ Start the Workshop scenario



➡ Click on Start Scenario
to bring up the instructions and start IDZ/ADDI

2.2 The Discover scenario will open the workshop instructions and start IDZ in the ADDI Perspective

2.2.1 Please take note of the ICONS for IDZ/ADDI and the workshop instructions

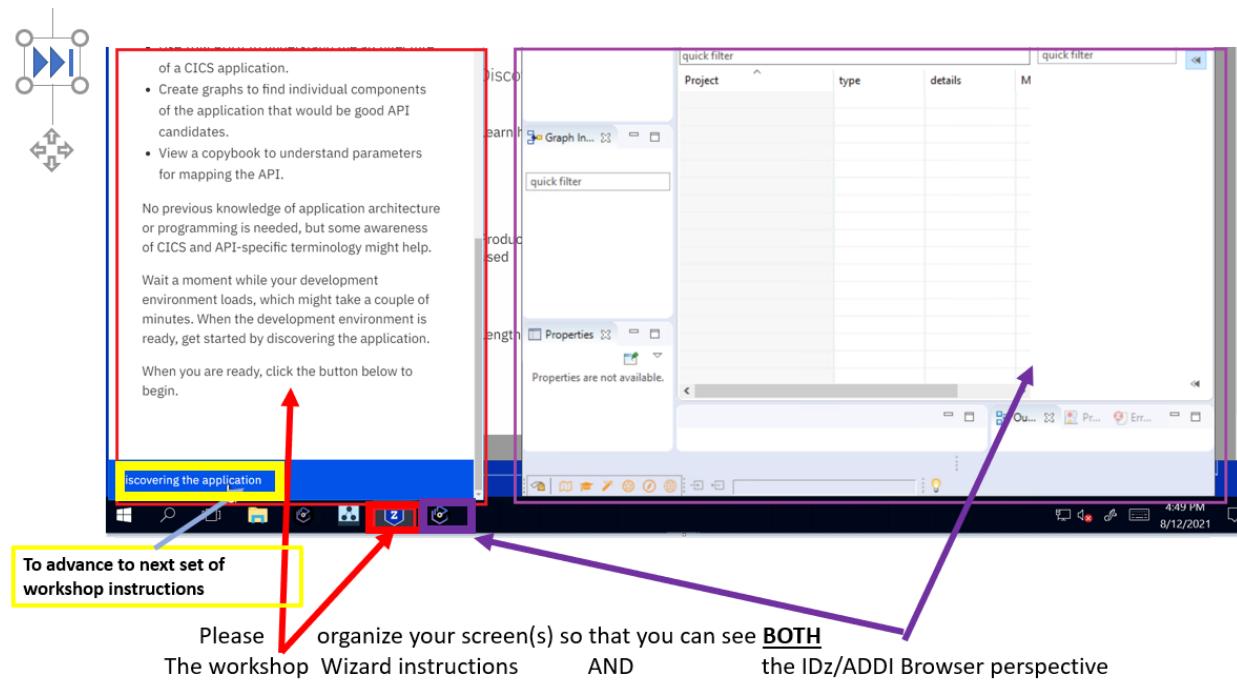


Start scenario will start IDZ  and OPEN the ADDI Perspective where you will be doing all of your ADDI Lab work

Please have patience. It may take a few seconds to open

These are your workshop instructions 

2.2.2 ► Reorganize your windows so that you can see BOTH the IDZ/ADDI perspective and the workshop instruction



2.2.3 ► From this point on you will be following the workshop instruction in the zTrial instance.

The workshop should take approximately 30 minutes+

These are all the ADDI functions that you will be using through this workshop
There will be a set of instruction for each Topic

Discovering the application

Use the AD Explore view to begin the discovery process on the application.

In this step, you will access and utilize the Explore view to begin understanding what candidate APIs exist in your codebase that can be enabled by using IBM® z/OS® Connect EE.

- Click the IBM Developer for z/OS icon in the taskbar.

The Application Discovery Browser perspective opens.

Overview

Creating a Program Call Graph

Filtering the Call Graph

Creating a Transaction Callgraph

Verifying processes within the code

Creating a Program Flow Graph

Accessing Analysis Source

Discovering the API Parameters

Next steps

Go back to prior topic/ set of workshop instruction

To advance to next topic/set of workshop instruction

IF you encounter any issues or have questions, contact the instructor.

You may also try any of the other scenario/workshops.

LAB 13 – (OPTIONAL) Run Integration Tests using Galasa (60 minutes)

Created by Will Yates. Modified by Regi Barosa on September 10, 2021 – Reviewed by Wilbert Kho.

This lab will take you through the steps needed to execute some sample Integration tests using **Galasa** against a sample mainframe application using **IDz** as IDE.

Galasa is an open-source test automation test framework that allows you to test both z/OS and Hybrid Cloud Applications. This capability allows you to test a system of engagement such as a web UI as well as validating the test within the system of record. Galasa is available from <https://galasa.dev>.

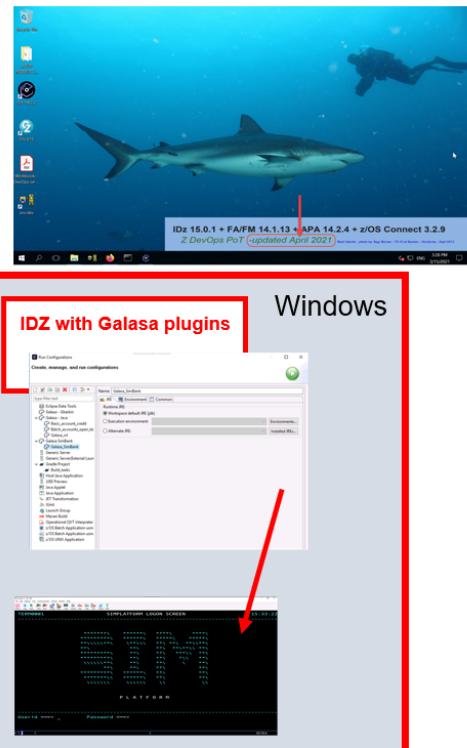
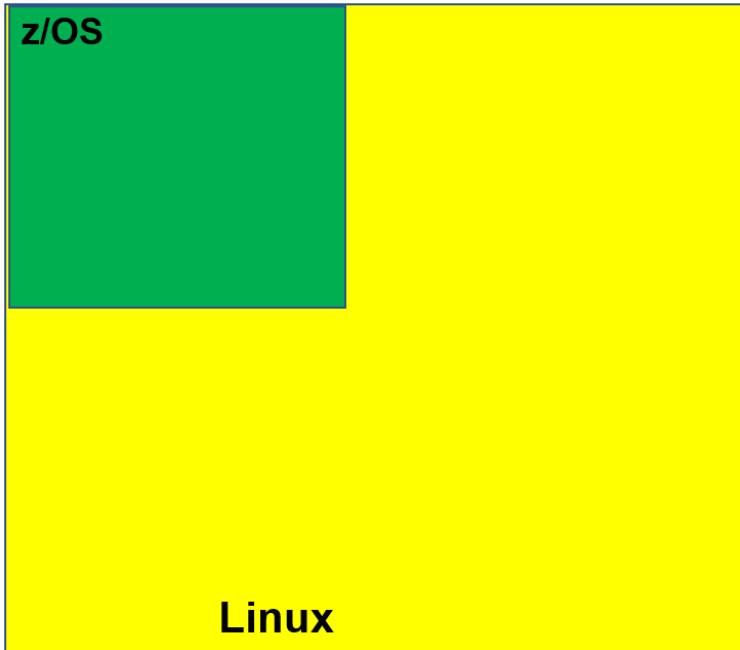
The IBM Distribution for Galasa is supported through [IBM Z Virtual Test Platform](#).

In this lab we will use the **SimBank** example that is provided through the Galasa eclipse plugin. This example provides a sample mainframe style application as well as a set of tests that can be executed to test the application. To simplify the demo process, the installation of the Galasa framework has already been completed within the virtual lab environment. However, you can download the same sample direct from <https://galasa.dev>.

Notice that even your environment has a Z/OS and Linux for this exercise you are using only the Windows client environment.

Topology used on this Lab

zD&T on Cloud



/ © 2021 IBM Corporation

Overview of development tasks

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

PART #1 –

Explore the **SimBank** application and execute the Installation Verification Test (IVT)

7. Start the example application SimBank from within IBM Developer for zOS
8. Use IBM Personal Communications to log onto the sample application
9. Run the IVT test to validate that Galasa can connect to the SimBank Application
10. Open the Test Result Editor

PART #2 –

Execute a web service and 3270 integration test

11. Execute a mixed web service and 3270 test within the Galasa framework
12. Examine the source code to see how the Galasa test works
13. Examine the Run Editor to see the stored artifacts
14. Change the test to log the request and response for the web service

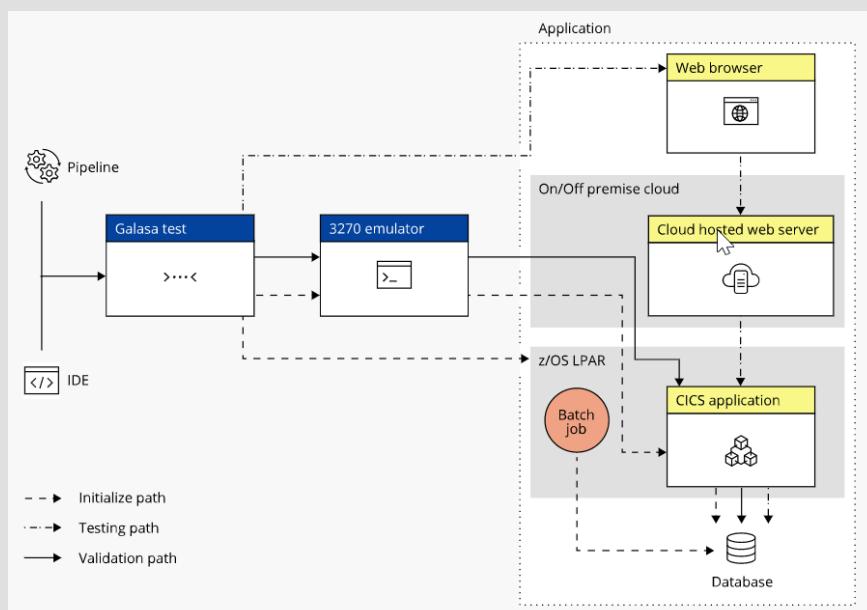
PART #3 –

Execute a Batch test

15. Execute a batch test within the Galasa framework
16. Examine the source code to see how the Galasa test works
17. Examine the Run Editor to see the stored artifacts

What is Galasa ?

Galasa simplifies testing in such an environment. The following diagram shows an example of how you can use Galasa to test a hybrid cloud application:



PART #1 – Explore the **SimBank** Application and execute the Installation Verification Test (IVT)

Galasa provides a sample application named **SimBank** that emulates a CICS environment and runs under Windows. This sample has been installed in our environment.

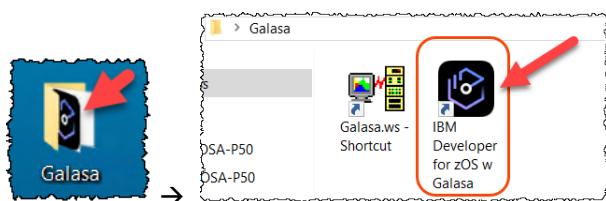
Section 1. Start the sample provided **SimBank** Application

Using the eclipse plugin from within the **IBM Developer for z/OS** you will start the **SimBank** application and use the console to check that the Application has started

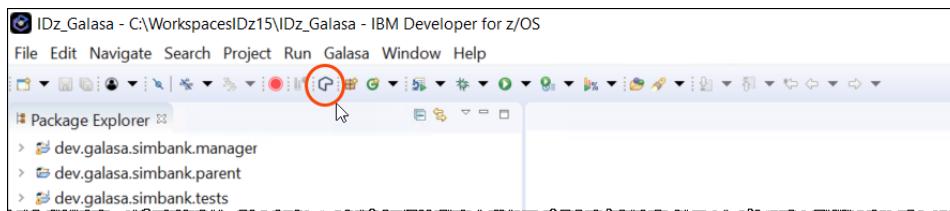
1.1 Start IBM Developer for z/OS

1.1.1 ► Double click the **Galasa** folder that is located on the Desktop

► Double click the icon “**IBM Developer for z/OS w Galasa**”.

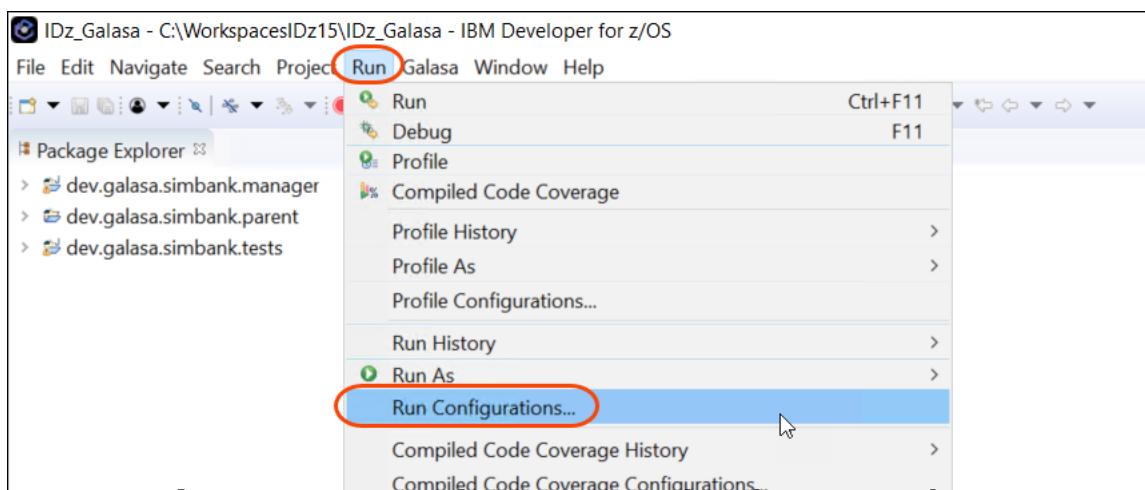


This will start the IDz environment. To speed this lab the Galasa eclipse plugin has already been installed into IDz and windows environment. You can see that the plugin is available by the inclusion of a ‘Galasa’ menu item and the **Galasa logo** within the tool bar seen below:



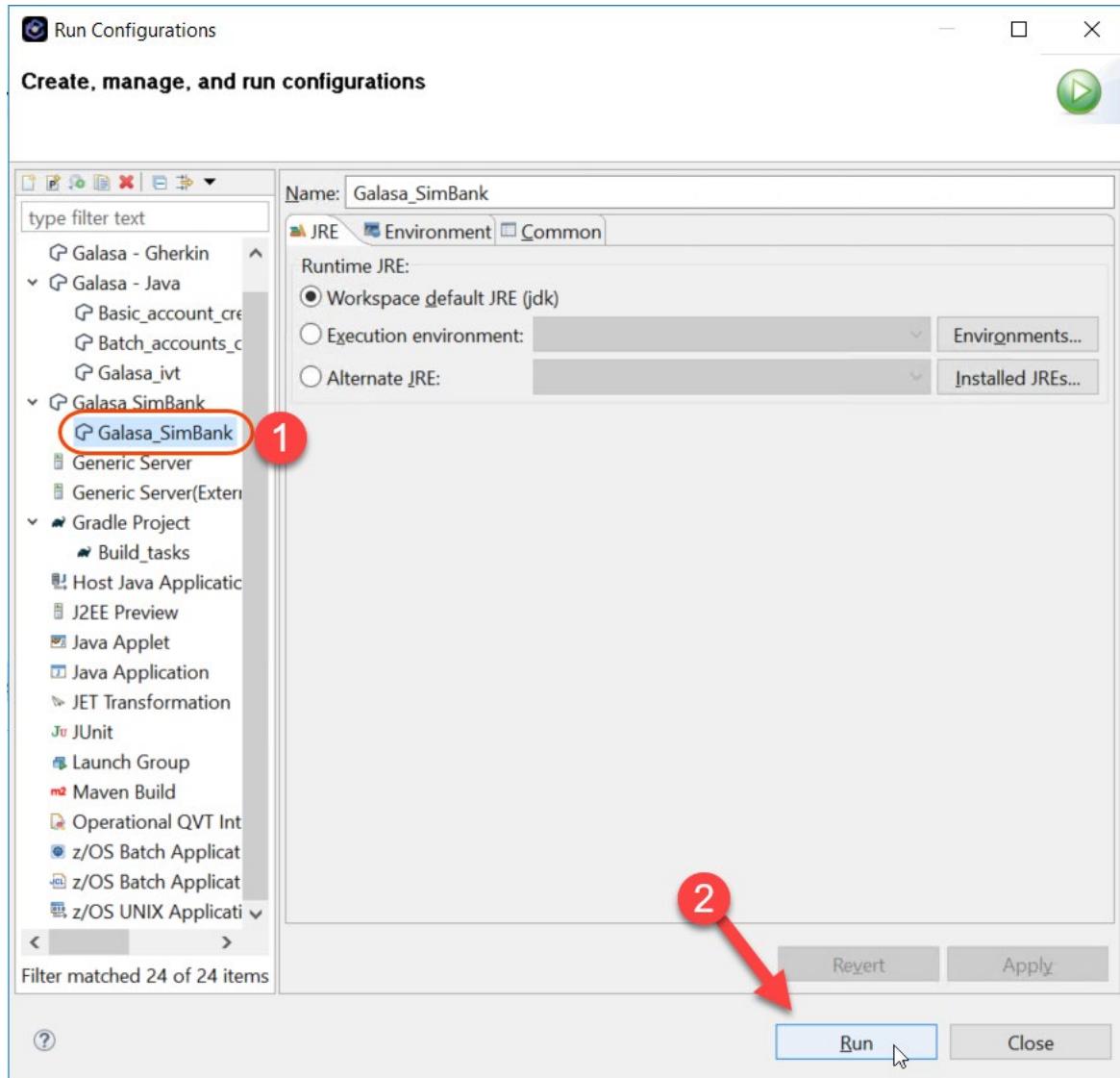
1.2 Run Galasa SimBank

1.2.1 ► Select **Run** and **Run Configurations...** from the menu within IDz.



1.2.2 This will open the run configurations dialog:

► Select **Galasa_SimBank** and click **Run**.



1.2.3 The *Run Configurations* dialog should disappear.

The *Console* view should open automatically displaying “**Simplatform started**”

```
Problems @ Javadoc Declaration Console
Galasa_SimBank [Galasa SimBank] C:\Program Files\IBM\IDZ_Galasa\jdk\bin\javaw.exe (Sep 9, 2021, 1:48:30 PM)
09/09/2021 13:48:35.334 INFO dev.galasa.simplatform.listener.Listener <init> Loading service: dev.galasa.simplatform.listener.ManagementFacilit
09/09/2021 13:48:35.336 INFO dev.galasa.simplatform.listener.Listener <init> Service: dev.galasa.simplatform.listener.ManagementFacilityListe
09/09/2021 13:48:35.337 INFO dev.galasa.simplatform.main.Simplatform main ... services loaded
09/09/2021 13:48:35.339 INFO dev.galasa.simplatform.main.Simplatform main Starting Derby Network server....
09/09/2021 13:48:35.953 INFO dev.galasa.simplatform.main.Simplatform main ... Derby Network server started on port 2027
09/09/2021 13:48:35.954 INFO dev.galasa.simplatform.main.Simplatform main ... Simplatform started
```

This validates that the simplatform application has correctly started within the IDE and is ready for execution.

Please note that closing IDz will terminate the SimBank application.

What is Simbank?

Distributed with Galasa, **SimBank** is a component that simulates a mainframe application. **SimBank** implements a sample banking application against which you can configure and run a set of provided tests in preparation for running your own tests against an actual mainframe application. You can also practice writing some new tests to run against the SimBank banking application.



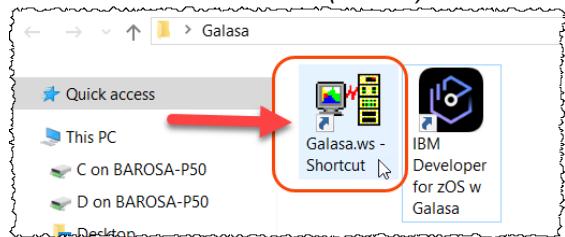
By exercising the Galasa framework against SimBank, you can pre-empt a lot (but not all) of the work and learning necessary to eventually hook your own tests up with a genuine mainframe environment. If the provided SimBank tests do not work, then it is unlikely that you will be able to run your own tests on a mainframe application. In summary, SimBank helps you to learn Galasa's basic principles of operation before you need to learn how to connect Galasa to your own mainframe application-under-test.

Section 2. Use IBM Personal Communications (PCOM) to log onto the sample application

As pointed out before we are emulating a CICS environment, but you can use a 3270 emulator like PCOM to execute the transaction that emulates the SimBank application.

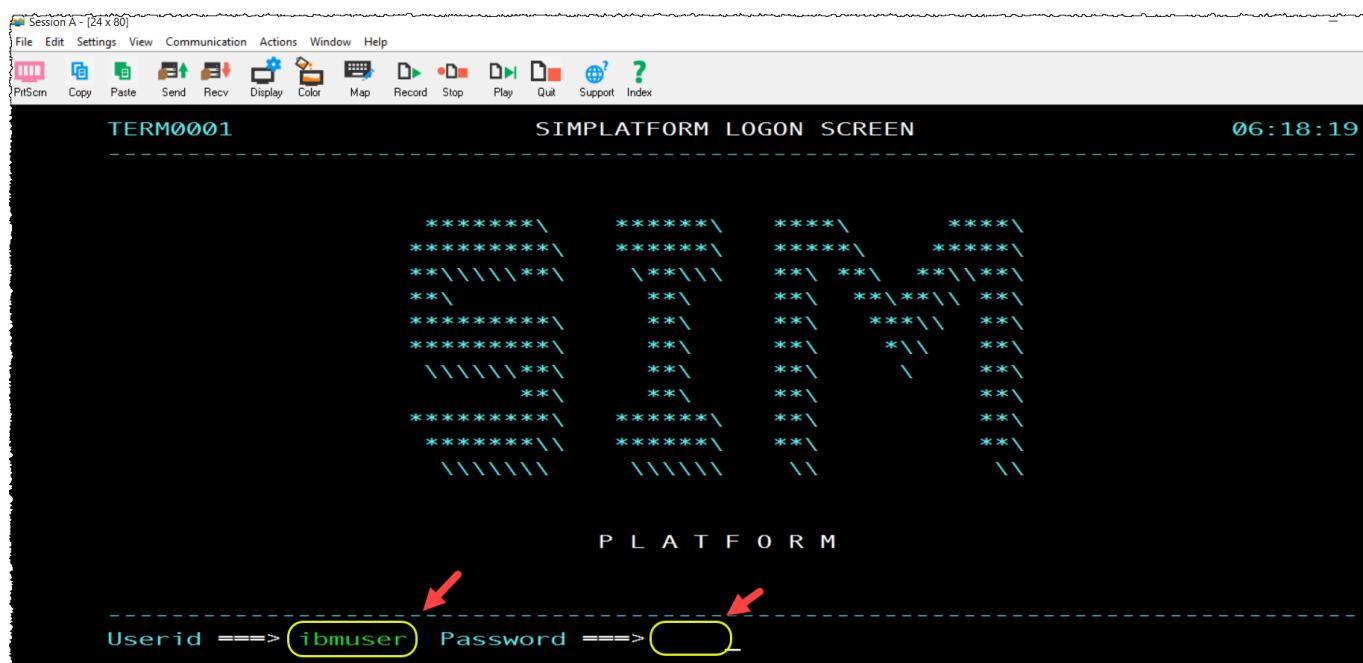
2.1 ► Minimize IDz.

2.2 ► From the Galasa folder on the desktop double click on the **Galasa.ws – Shortcut** icon to start IBM Personal Communications (PCOM).

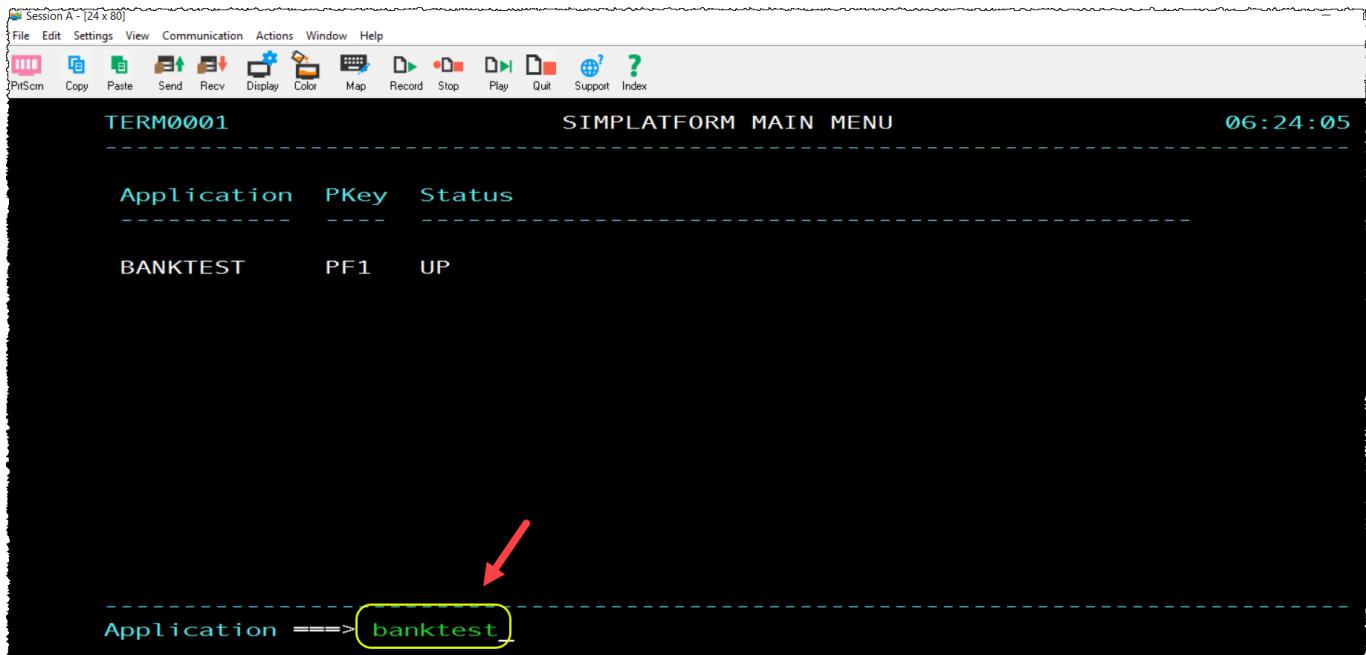


2.3 This is pre-configured to connect to the SimBank Application that is running within the virtual Environment.

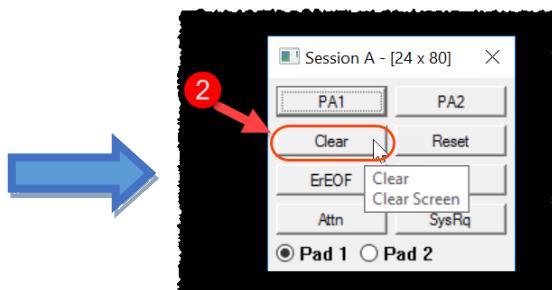
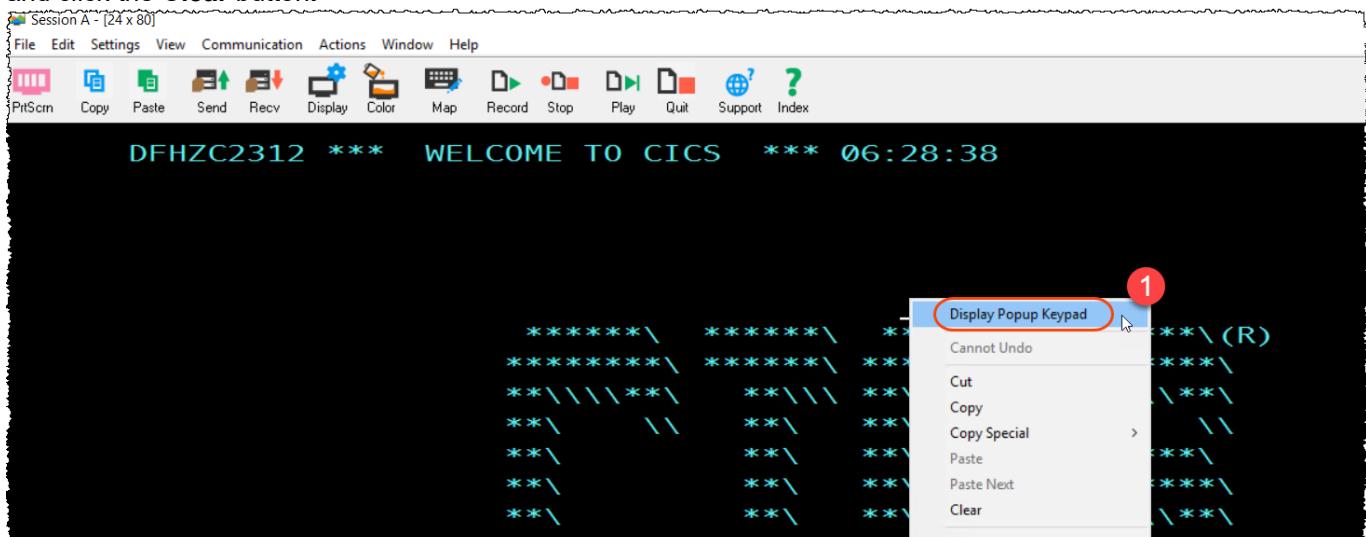
► Logon onto the application using **ibmuser** and **sys1** and press **enter**



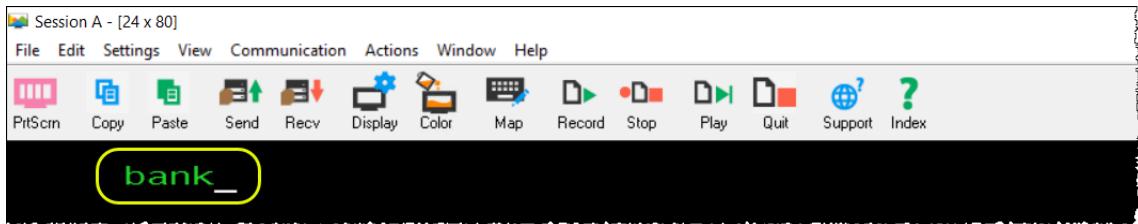
2.4 ► Enter the application by either typing **banktest** and pressing **enter** or by pressing **PF1**



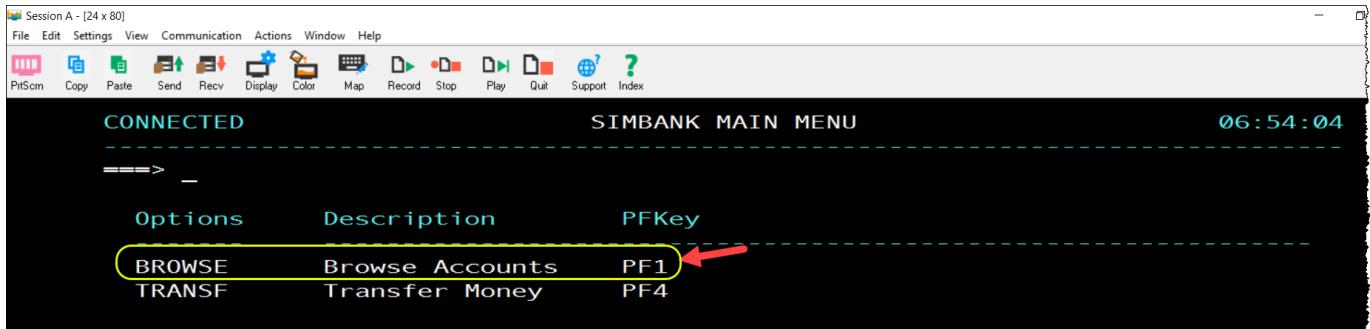
2.5 ► To clear the screen, right click on the dark space and select **Display Popup Keypad** and click the **Clear** button.



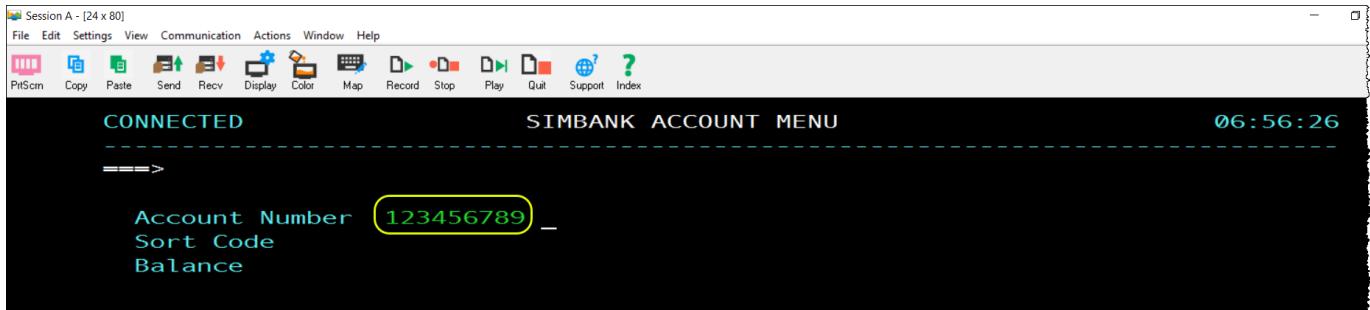
2.5 ► Run the bank transaction by typing **bank** and pressing **enter**



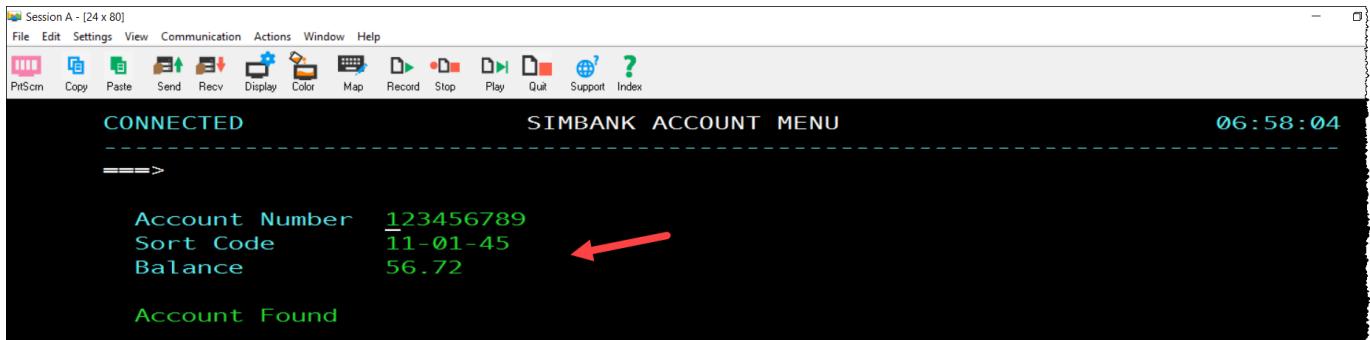
2.6 ► Press **PF1** to execute the **browse** option



2.7 ► Type the account number **123456789** and press **enter** to retrieve the customer details.

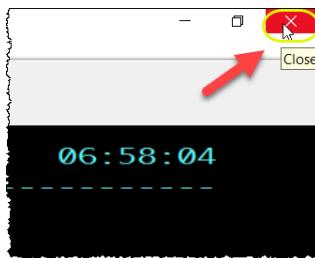


2.8 The customer data will be displayed.



The customer **987654321** also exists and can be retrieved.

2.9 ►► The Personal Communications (PCOM) application can now be **closed**.

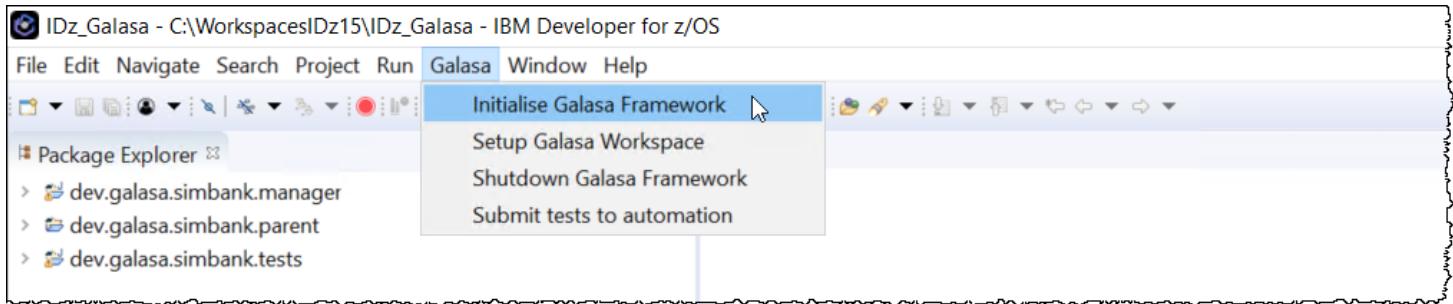


Section 3. Run the IVT test to validate that Galasa can connect to the SimBank Application

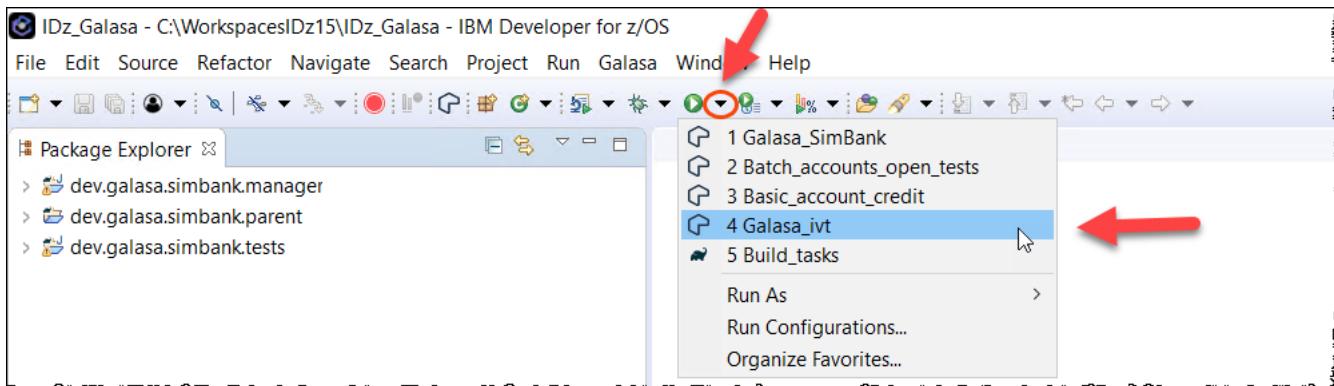
This basic test logs on to Galasa SimBank and examines an account.

3.1 ►► Ensure that IBM Developer for z/OS is running and **has the focus**.

3.2 ►► Start the Galasa Framework by selecting **Galasa -> Initialise Galasa Framework** from the menu bar.



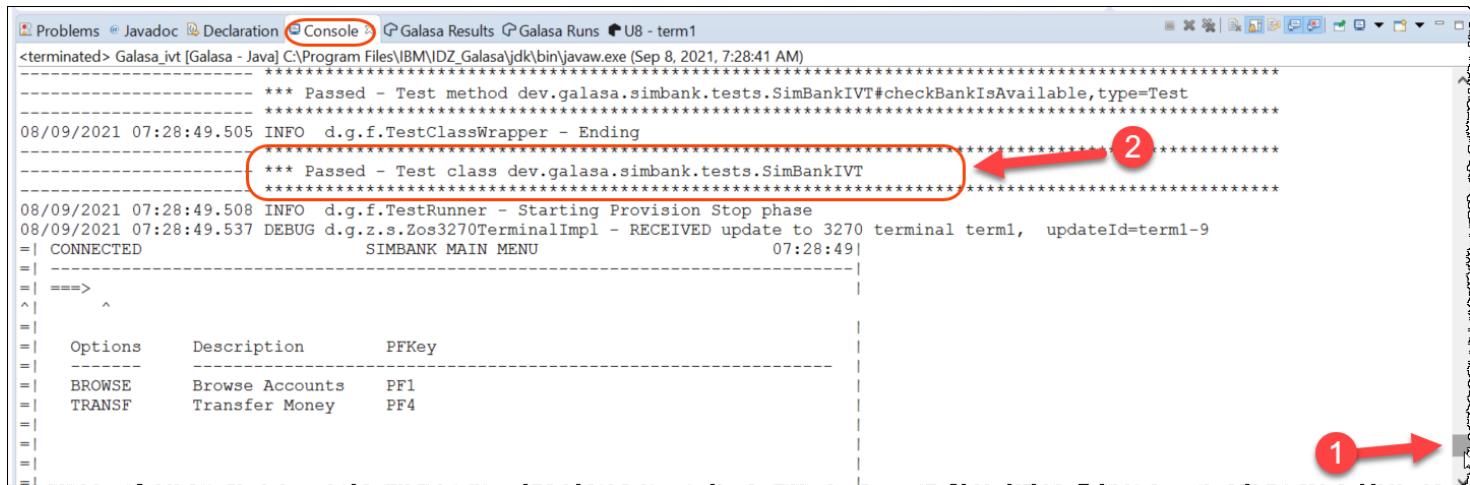
3.3 ►► Click on the small downward pointing arrow to the right of the green 'play' icon. From the dropdown that appears select **Galasa_ivt**
Notice that the order could be different from the screen capture below.



3.4 This will launch the Galasa IVT test which we will examine later. The console window should open and scroll a lot of text.

►► Scroll back a bit and should see the **message below**

This confirms that the test has run successfully and Galasa was able to connect to the SimBank application and execute.

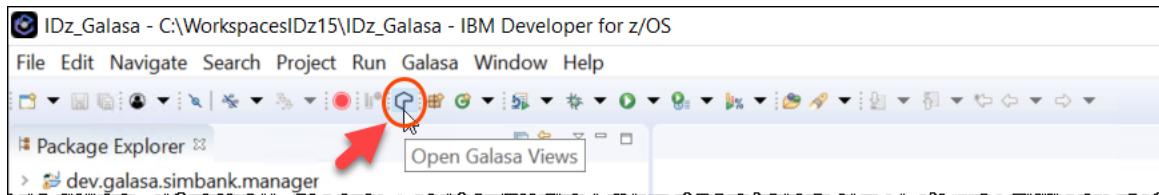


```
Problems Javadoc Declaration Console Galasa Results Galasa Runs U8 - term1
<terminated> Galasa_ivt [Galasa - Java] C:\Program Files\IBM\IDZ_Galasa\jdk\bin\javaw.exe (Sep 8, 2021, 7:28:41 AM)
=====
**** Passed - Test method dev.galasa.simbank.tests.SimBankIVT#checkBankIsAvailable,type=Test
*****
08/09/2021 07:28:49.505 INFO d.g.f.TestClassWrapper - Ending
*****
**** Passed - Test class dev.galasa.simbank.tests.SimBankIVT
*****
08/09/2021 07:28:49.508 INFO d.g.f.TestRunner - Starting Provision Stop phase
08/09/2021 07:28:49.537 DEBUG d.g.z.s.Zos3270TerminalImpl - RECEIVED update to 3270 terminal term1, updateId=term1-9
=| CONNECTED SIMBANK MAIN MENU 07:28:49|
=|
=|
=| ===>
^|
=|
=| Options Description PFKey
=|
=| BROWSE Browse Accounts PF1
=| TRANSF Transfer Money PF4
=|
=|
=|
=|
```

Section 4. Open the Test Result Editor

You can view the results of your test runs in Eclipse.

4.1 ►► Open the Galasa runs view by selecting the **Galasa logo**  on the menu bar.



4.2 This will open the **Galasa Results** and the **Galasa Runs** view.

►► Click on the **Galasa Results** view to give it focus.

►► You can now double click on **U1-SimBankIVT**



4.3 This will open the Run Editor for the test you just executed

The screenshot shows the 'Run U1' editor window. The title bar says 'Run U1'. The main area has a header 'U1' and a section titled 'Overview'. It displays the following information:

Run Name:	U1
Status:	finished
Result:	Passed
Requestor:	administrator
Bundle:	dev.galasa.simbank.tests
Test Name:	dev.galasa.simbank.tests.SimBankIVT
Queued:	Today, 15:39:04.697
Started:	Today, 15:39:04.809
Finished:	Today, 15:39:10.835

To the right of the overview, there is a table titled 'SimBankIVT - Passed' with two rows:

Test Method	Result
testNotNull	Passed
checkBankIsAvailable	Passed

At the bottom of the editor, there are tabs: General, Run Log, and Stored Artifacts. The General tab is selected.

From this editor, you can see that the run has the result '**Passed**' and executed correctly. Feel free to examine other parts of this editor in detail. However, we will look at the run editor in more detail in the next part of the lab.

4.4 ► Close the RUN U1 editor



PART #2 – Execute a web service and 3270 integration test

In this section we will take a look at another of the sample tests that run against the SimBank Application. In this case we will examine a test that:

- uses the 3270 interface to check an accounts balance
- executes a web service request against the application to credit an account with a set amount of credit
- uses the same logic as in 'a' again to validate that the web service updated the account balance successfully

Section 5. Execute a mixed web service and 3270 test within the Galasa framework

This test builds on the same application we have just examined and shows how Galasa can use a range of tools and technologies to drive the system under test..

5.1 ► Ensure that IDz is open and the SimBank application is running as per the instructions in the previous section. In the *Package Explorer* view you will note that three projects are available:

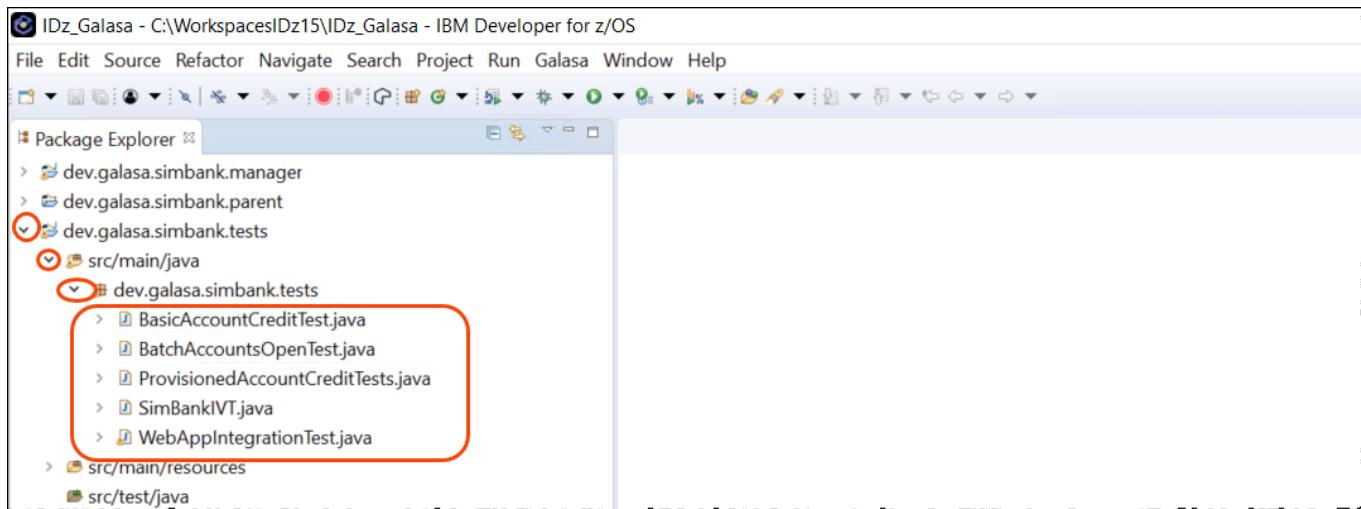
The screenshot shows the Eclipse IDE interface. The title bar says 'IDz_Galasa - C:\Workspaces\IDz15\IDz_Galasa - IBM Developer for z/OS'. The menu bar includes File, Edit, Navigate, Search, Project, Run, Galasa, Window, Help. The toolbar has various icons for file operations. The 'Package Explorer' view is open on the left, showing three projects:

- > dev.galasa.simbank.manager
- > dev.galasa.simbank.parent
- > dev.galasa.simbank.tests

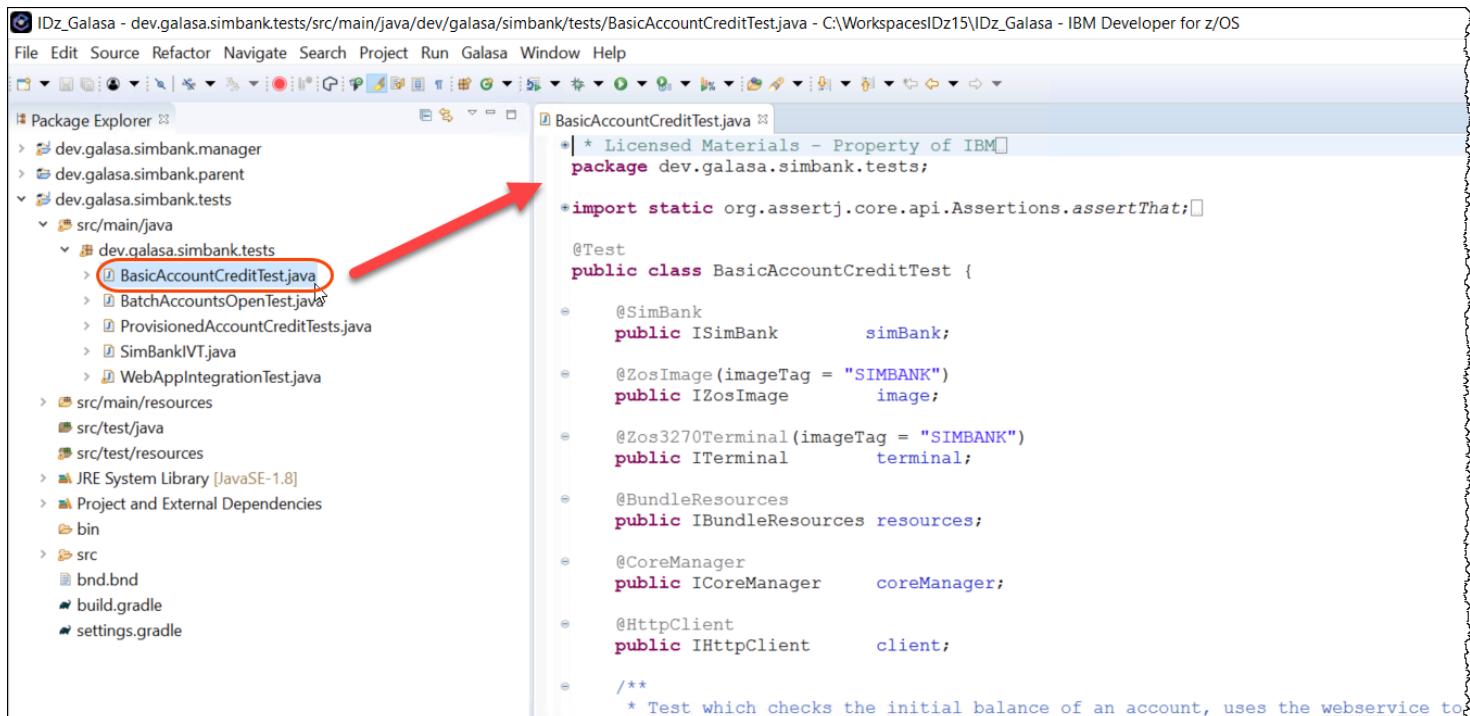
5.2 These three projects are the standard sample tests that are included as examples within the eclipse Galasa plugin.

▶▶ Expand the project **dev.galasa.simbank.tests**, **src/main/java** and **dev.galasa.simbank.tests**. The project should contain the five tests.

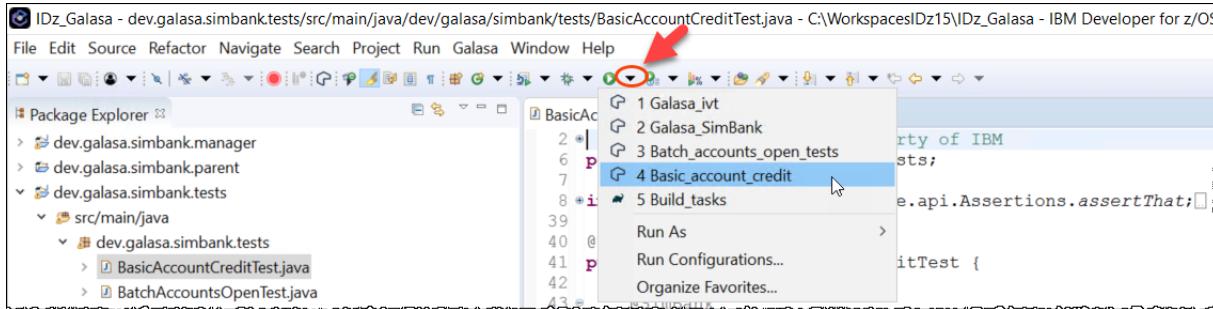
The **SimBankIVT test** was the test that was executed in *Part #1*. Please feel free to examine the source code of the test.



5.3 ▶▶ Double click on **BasicAccountCreditTest.java** to open the test

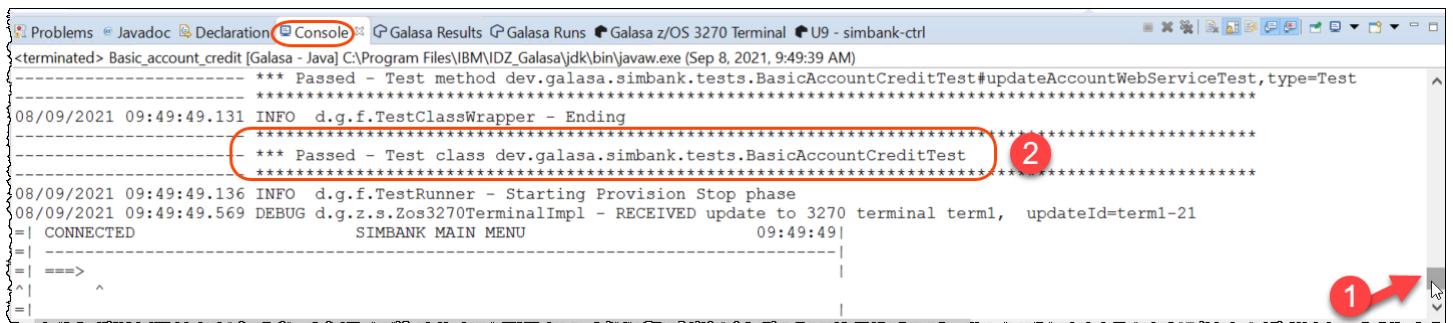


5.4 ► To execute this test, similarly as before, click the drop down next to the green play icon and select 'Basic_account_credit' – note that the numbering might be different to that in the screenshot



5.5 The test will execute.

► Using the *Console* view, scroll up a bit and you will see the message as below:



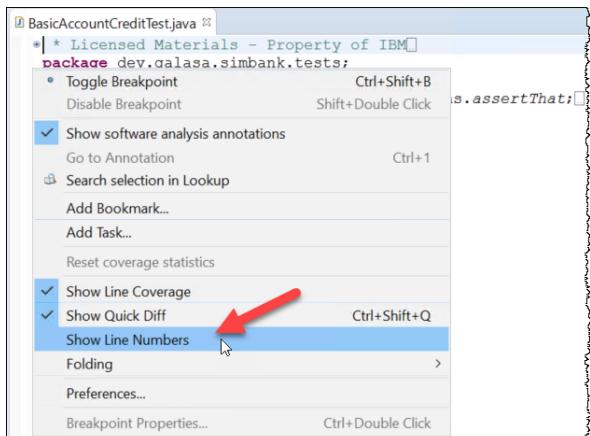
This message ensure that the test has correctly finished. If this message does not appear or the test appears to have failed. Ensure that the SimBank application is running by following the steps in Part #1. Remember if IDZ is closed then the SimBank application will exit and will need to be manually restarted.

Section 6. Examine the source code to see how the Galasa test works

Now that we have executed the test we will examine the source code. We will show how the test connected to a z/OS image, a 3270 terminal and executed a web service request

6.1 Ensure that the editor containing the source for test **BasicAccountCreditTest.java** is opened
To easily direct you to specific areas of the code, turn on line numbers by

► Right click in the **margin on the left** hand side of the editor and select **show line numbers**



Notice that this could be done by selecting ‘window’ -> ‘Preferences’ from the menu bar and navigating to ‘General’ -> ‘Editors’ -> ‘Text Editors’ and selecting ‘Show line numbers’.

6.2 ►► Scroll to line 40, the public class is annotated with **@Test** to denote this Java class as a Galasa test that should be executed within the Galasa Framework

```
BasicAccountCreditTest.java
2 * * Licensed Materials - Property of IBM
6 package dev.galasa.simbank.tests;
7
8 *import static org.assertj.core.api.Assertions.assertThat;
39
40 @Test
41 public class BasicAccountCreditTest {
42
43     @SimBank
```

6.3 ►► Scroll to line 46-47.

Here we instruct the Galasa framework that we need to bind to a z/OS image by annotating a public **IzosImage** with **@ZosImage(imageTag = "SIMBANK")**. Galasa will at runtime use it configuration stores to connect to a particular z/OS LPAR where the SimBank application has been deployed. Galasa will populate the variable *image* with an object that represents the z/OS image

```
46 @ZosImage(imageTag = "SIMBANK")
47 public IZosImage image;
```

6.4 ►► Scroll to line 49-50.

Here we use a similar technique to allow Galasa to generate and bind the test to a virtual 3270 terminal connected to the same z/OS image as the example above. The test can then use this object to interact with the 3270 data stream.

```
48
49 @Zos3270Terminal(imageTag = "SIMBANK")
50 public ITerminal terminal;
```

6.5 ►► Scroll to line 58-59.

Here Galasa is being asked to provision a HTTP client that the test can use to invoke the webservice that it requires.

```
57
58 @HttpClient
59 public IHHttpClient client;
```

Now that we have seen some of the ways a Galasa test access mainframe resources, we will now go on to see how these object can be used to interact with z/OS Resources.

6.6 ►► Scroll to line 87-91.

Here the code uses the terminal object to enter the **userid** and **password** into the initial 3270 screen, **clear** the screen and enter the **bank** transaction

```
86     // Initial actions to get into banking application
87     terminal.waitForKeyboard().positionCursorToFieldContaining("Userid").tab().type("IBMUSER")
88         .positionCursorToFieldContaining("Password").tab().type("SYS1").enter().waitForKeyboard()
89
90         // Open banking application
91         .pf1().waitForKeyboard().clear().waitForKeyboard().type("bank").enter().waitForKeyboard();
```

Although Galasa tests are coded, rather than typed it uses a fluent style of programming that makes this type of interaction, simple to read and understand. Note that although we are entering a userid and password into these fields, these can be held within our credential store and not hard coded into the test.

6.7 ►| Scroll to line 96-107.

Those lines contain all the code to execute the web service.

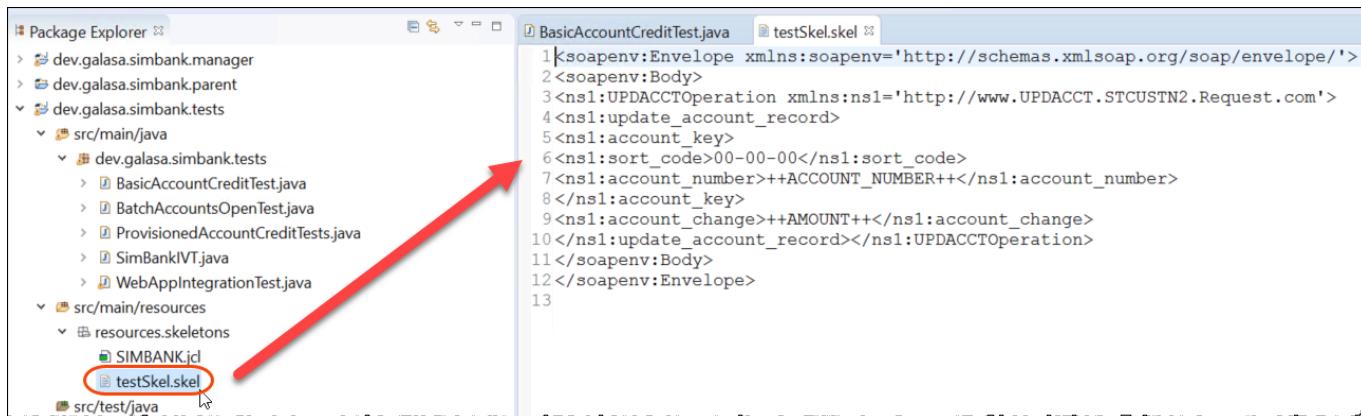
First a *HashMap* is created and populated with an *account number* and an *amount*.

This is then used to fill out a skeleton file found at

Src/main/resources -> resources.skeletons -> testSkel.skel

```
96     // Set the amount be credited and call web service
97     BigDecimal amount = BigDecimal.valueOf(500.50);
98     HashMap<String, Object> parameters = new HashMap<String, Object>();
99     parameters.put("ACCOUNT_NUMBER", "123456789");
100    parameters.put("AMOUNT", amount.toString());
101
102    // Load sample request with the given parameters
103    String textContent = resources.retrieveSkeletonFileAsString("/resources/skeletons/testSkel.skel", parameters);
104
105    // Invoke the web request
106    client.setURI(new URI("http://" + this.simBank.getHost() + ":" + this.simBank.getWebnetPort()));
107    client.postText("updateAccount", textContent);
```

6.8 ►| Expand **src/main/resources, resources.skeletons** and double click **testSkel.skel** to open it

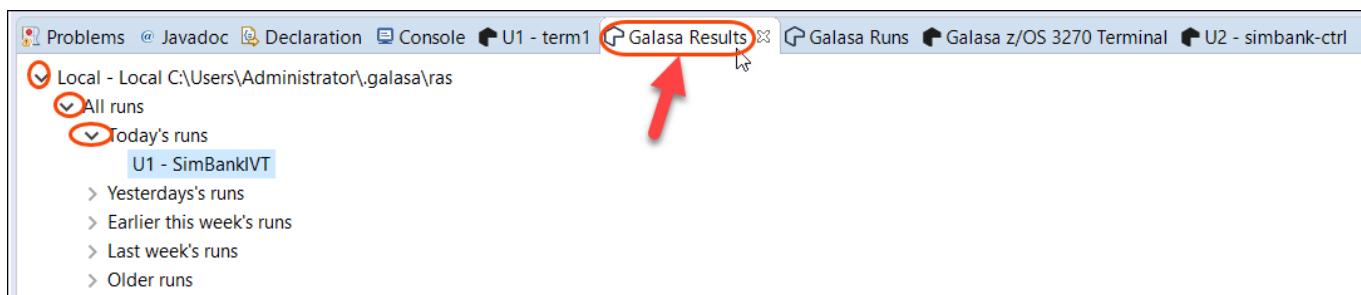


Section 7. Examine the Run Editor to see the stored artifacts

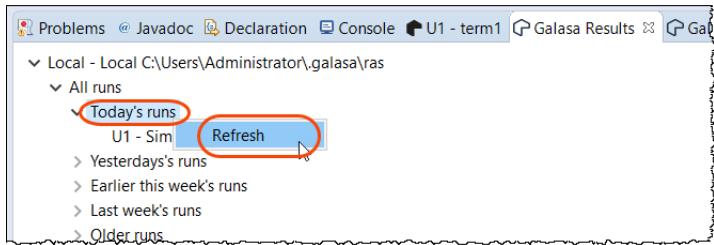
Now you have seen the source code that makes up the test and run it within the Galasa framework we will now examine the Galasa run editor.

7.1 Examine the Run Editor to see the stored artifacts

►| Using the '**Galasa Results**' View, expand Local, All runs and Today's runs

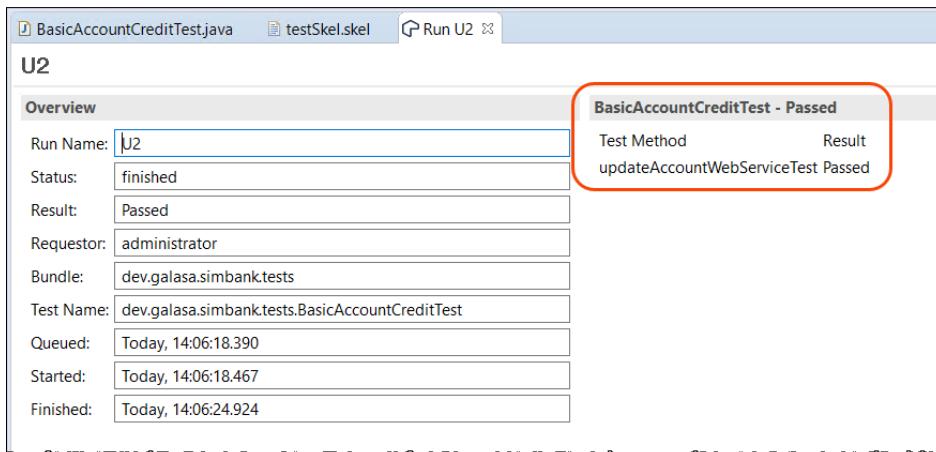


7.2 ➡ Right click on Todays runs and select Refresh,



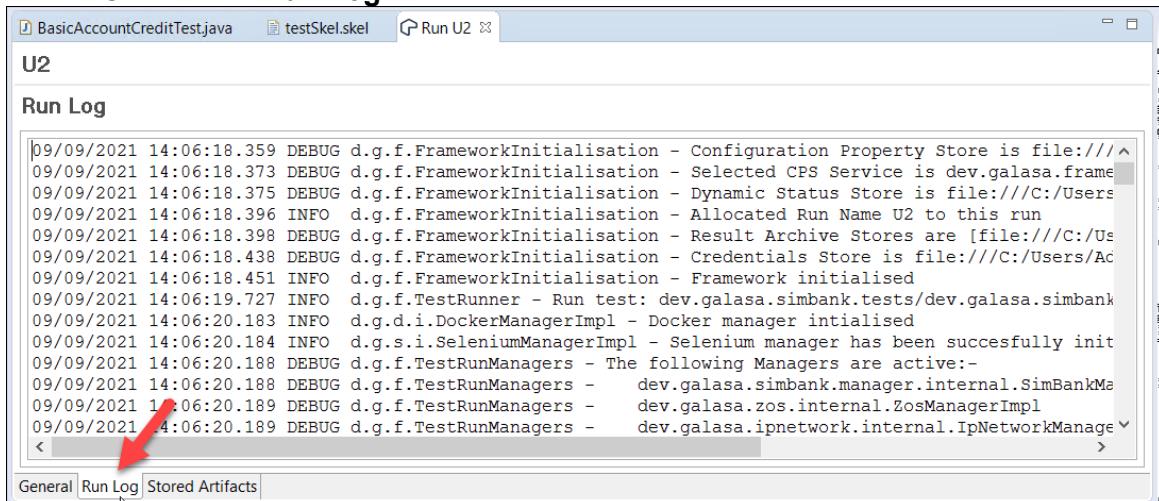
This should bring up (if not already available) **U2 – BasicAccountCreditTest**
(Notice that this number can vary. Usually is **U2**, but it may be different if you run multiple times)

7.3 ➡ Double click on **U2 – BasicAccountCreditTest** to bring up the *Galasa Run Editor* for that test



From this screen we can see the that the test has completed, that the overall status of the test is passed, each of the test methods have passed and the timestamps of when the test passed.
For example this test only took 7 seconds to complete.

7.4 ➡ Click on the **Run Log** tab at the bottom of the run editor



7.5 This view shows the entire log of the test as it ran.

► Scroll through the Run log and you will see that the test captured and added to the run log each 3270 screen that was seen as the test executed. Also all the passwords that were entered into a screen has been masked.

```
=|          ***\\ / \\ **|  \ *\\ / \\  **\\ **\\ / \\ **\\
=|          **\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          *****\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          *****\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          \\\\\\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          **\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          *****\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          *****\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|          \\\\\\\ / \\ **|  **\\ / \\ **\\ / \\ **\\ / \\
=|
=|          P L A T F O R M
=|
=| -----
=| Userid ===> IBMUSER Password ===> *1**
^|
=|
=| !! Connected-Plain Size=24x80 Cursor=1720,21x40 Keyboard Locked
<
```

7.6 ► Click on the **Stored Artifacts** view of the Run editor Any resources that are captured during the test will be stored in the stored artifacts view.

► Expand **zos3270 -> terminals** and double click on **simbank-ctrl** to bring up the terminal view

```
General Run Log Stored Artifacts
```

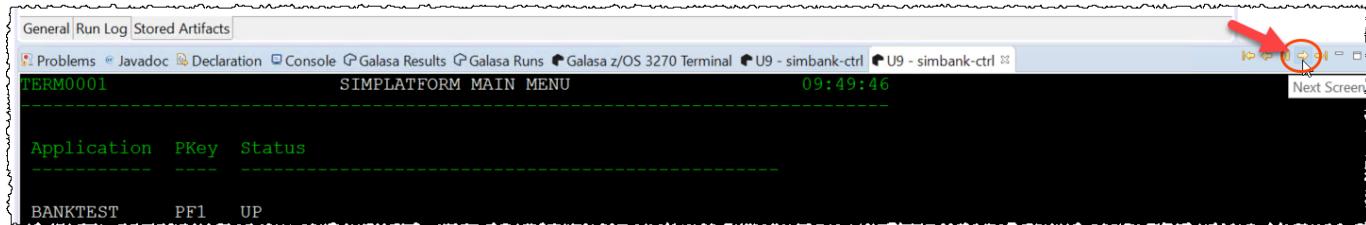
TERM0001 SIMPLATFORM LOGON SCREEN 09:49:45

```
*****\\ / \\ **|  *****\\ / \\ **|  *****\\ / \\ **|  ****\\ / \\
*****\\ / \\ **|  *****\\ / \\ **|  *****\\ / \\ **|  ****\\ / \\
**\\ / \\ **|  \\\\\\\ / \\ **|  **\\ / \\ **|  **\\ / \\
*****\\ / \\ **|  **\\ / \\ **|  **\\ / \\ **|  **\\ / \\
*****\\ / \\ **|  **\\ / \\ **|  **\\ / \\ **|  **\\ / \\
\\\\\\\ / \\ **|  **\\ / \\ **|  **\\ / \\ **|  **\\ / \\
**\\ / \\ **|  **\\ / \\ **|  **\\ / \\ **|  **\\ / \\
*****\\ / \\ **|  *****\\ / \\ **|  **\\ / \\ **|  **\\ / \\
*****\\ / \\ **|  *****\\ / \\ **|  **\\ / \\ **|  **\\ / \\
\\\\\\\ / \\ **|  \\\\\\\ / \\ **|  **\\ / \\ **|  **\\ / \\
          P L A T F O R M
-----
```

Userid ===> Password ===>

Screen 1/9 - simbank-ctrl-1 - 80x24 - Inbound

7.7 Use the navigation controls at the top right corner of the view to move through the sequence of screens



Section 8 Change the test to log the request and response for the web service

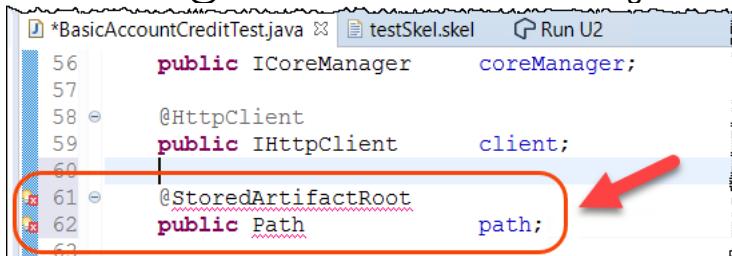
Unlike with 3270 screens, the HTTPManager cannot store every request and response that it processes as this might contain a lot of data which we would not want to always log, so Galasa provides a way to easily log this content. In this case we will change this test to log the request and response messages from the web service call, we will then rebuild the test, execute it and check that the new data is stored in the results archive.

8.1 Ensure that the editor for **BasicAccountCreditTest.java** is open

After the line **60** insert the 2 lines below: (copy/paste from here)

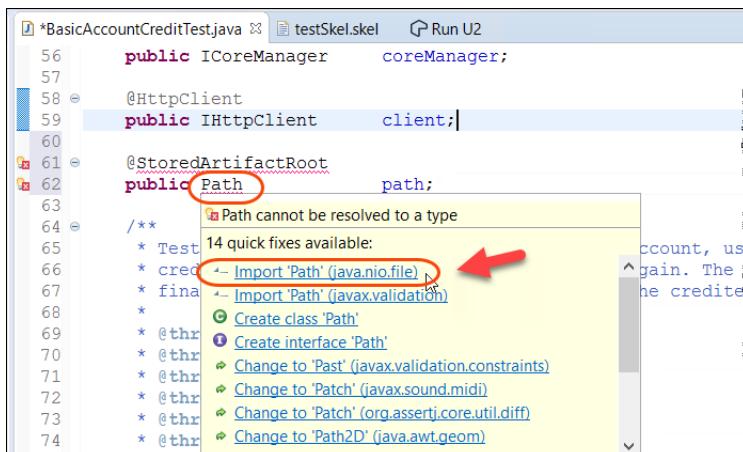
```
@StoredArtifactRoot  
public Path path;
```

You used the `@StoredArtifactRoot` annotation against a declaration of a `public Path` called `path`

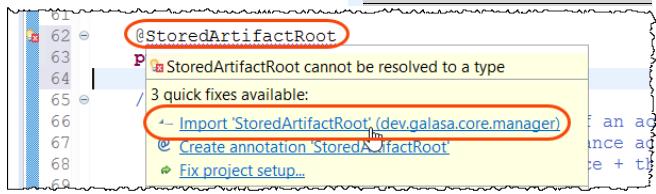


8.2 Note that errors are flagged.

Move the mouse to "Path" and select `import 'Path' (java.nio.file)` as below



8.3 ➡ Move the mouse to “`@StoredArtifactRoot`” and select `importStoredArtifactRoot` as below



8.4 At runtime Galasa will set the value of path to the location of the results archive for the currently running test.

➡ Add the five lines below after line 118 to log the request and response of the webservice call (use copy/paste from the PDF).

```
// Invoke the web request

client.setURI(new URI("http://" + this.simBank.getHost() + ":" +
this.simBank.getWebnetPort()));

Files.write(path.resolve("webService").resolve("request"), textContent.getBytes(),
StandardOpenOption.CREATE);

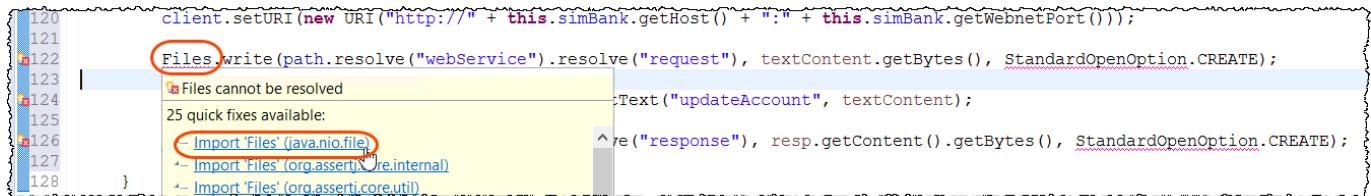
HttpClientResponse<String> resp = client.postText("updateAccount", textContent);

Files.write(path.resolve("webService").resolve("response"), resp.getContent().getBytes(),
StandardOpenOption.CREATE);
```

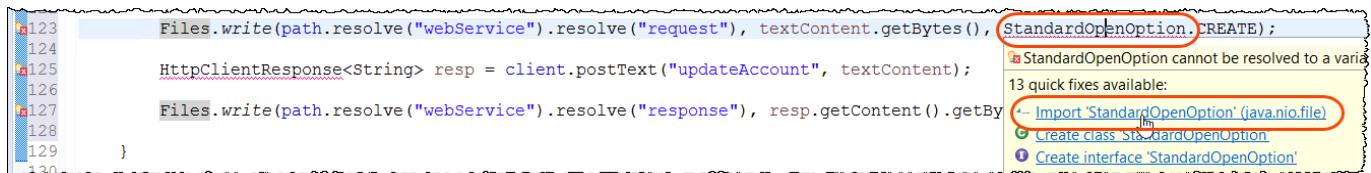
Notice that more errors are flagged. We must correct like we did a while ago.



8.5 ➡ Move the mouse to “`Files.write`” and select `import 'Files' (java.nio.file)` as below



8.6 ➡ Move the mouse to “`StandardOpenOption`” and select `import 'StandardOpenOption' (java.nio.file)` as below



8.7 ► Move the mouse to “HttpClientResponse” and select **import ‘HttpClientResponse’ (dev.galasa.http)** as below

The screenshot shows a Java code editor with line numbers 126 to 131. At line 126, there is a red circle around the text "HttpClientResponse<String> resp = client.postText("updateAccount", textContent);". A tooltip appears with the message "HttpClientResponse cannot be resolved to a type" and "12 quick fixes available". One of the quick fixes is highlighted with a red circle and labeled "Import 'HttpClientResponse' (dev.galasa.http)".

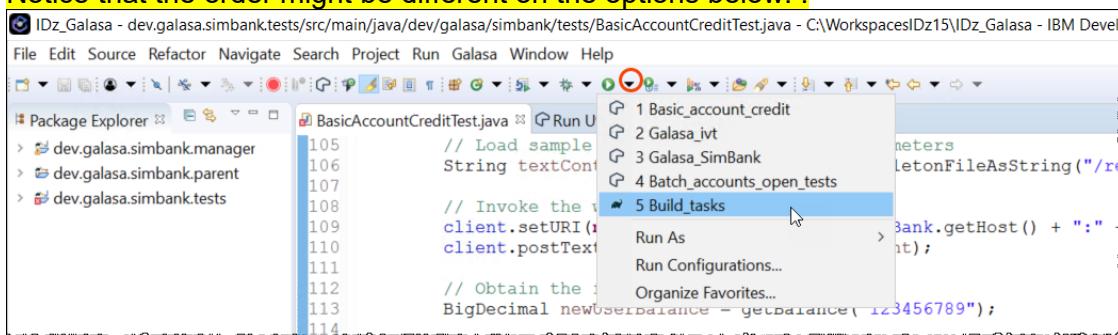
The Java errors should be gone

8.8 ► Use **Ctrl + S** to save the change.

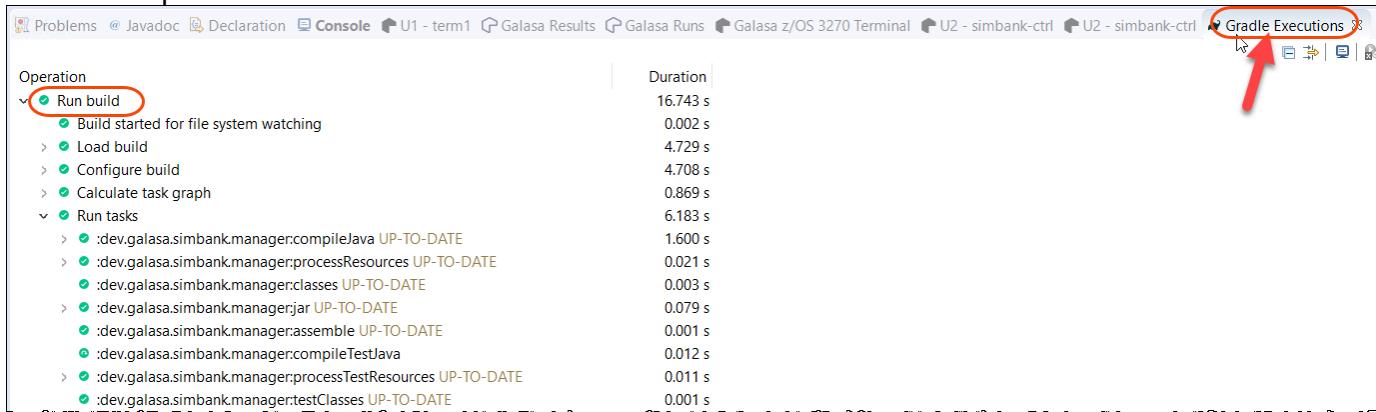
8.9 You can now build the tests by

► Clicking on the small drop down icon next to the green play button and selecting ‘Build_tasks’

Notice that the order might be different on the options below. .

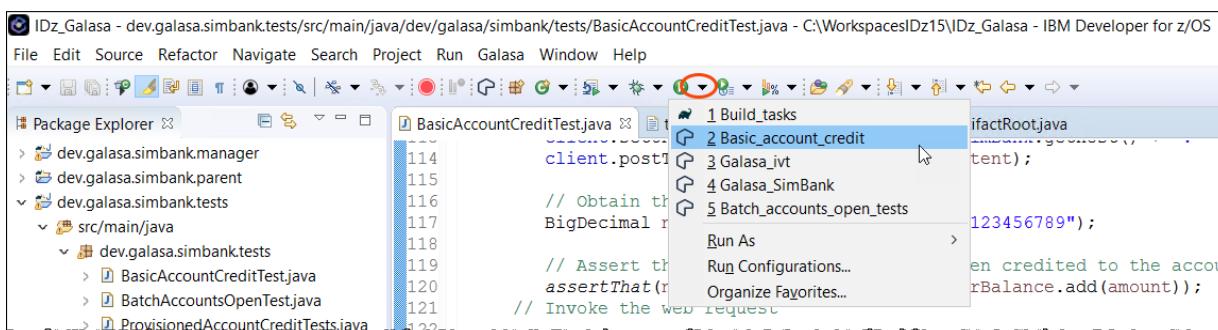


8.10 The ‘Gradle Executions’ view will open, when the **Run build** item in the view has a green tick, the build is complete.



8.11 In the same way as you did before re-execute the test again.

► As you did it before. To execute this test click the drop down next to the green play icon and select ‘**Basic_account_credit**’ – note that the numbering might be different to that in the screenshot



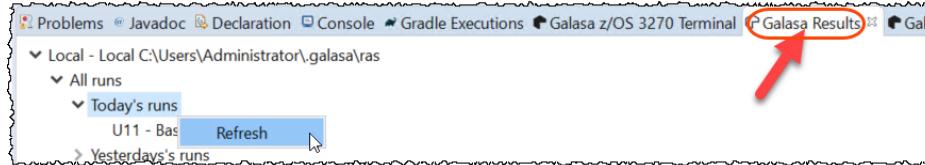
8.12 The test will execute.

► Using the **Console** view, scroll up a bit and you will see the message as below:

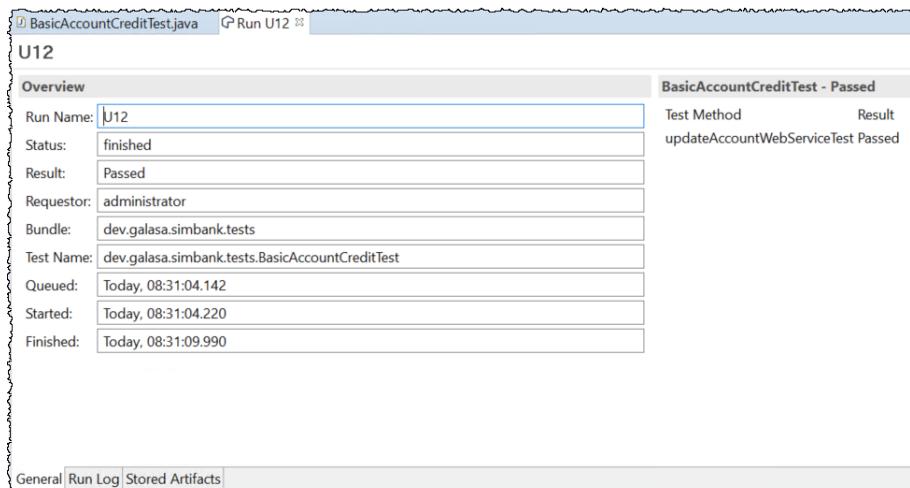
The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays log messages from a Java application running on z/OS. A red circle labeled '2' highlights a specific line of output: '*** Passed - Test class dev.galasa.simbank.tests.BasicAccountCreditTest'. A red arrow labeled '1' points to the 'Galasa Results' tab at the top of the interface.

```
<terminated> Basic_account_credit [Galasa - Java] C:\Program Files\IBM\IDZ_Galasa\jdk\bin\javaw.exe (Sep 8, 2021, 9:49:39 AM)
----- *** Passed - Test method dev.galasa.simbank.tests.BasicAccountCreditTest#updateAccountWebServiceTest, type=Test
----- **** Passed - Test class dev.galasa.simbank.tests.BasicAccountCreditTest
----- ****
08/09/2021 09:49:49.131 INFO d.g.f.TestClassWrapper - Ending
----- ****
08/09/2021 09:49:49.136 INFO d.g.f.TestRunner - Starting Provision Stop phase
08/09/2021 09:49:49.569 DEBUG d.g.z.s.Zos3270TerminalImpl - RECEIVED update to 3270 terminal term1, updateId=term1-21
=| CONNECTED SIMBANK MAIN MENU 09:49:49|
=| -----
=| ===>
^| ^
=| 
```

8.8 Within the Galasa Results View right click on Today's runs and select refresh

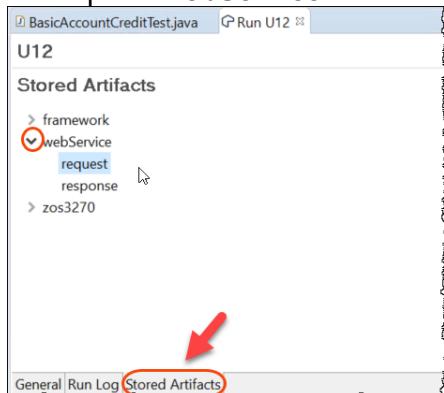


8.9 Double click the latest run of the BasicAccountCreditTest to open the Galasa run editor for your new run



8.10 Click on the **Stored Artifacts** tab within the run editor. You should see a new artifact called **webService**.

► Expand **webService** and double click **request** and **response**



8.11 ► Examine the **request** and **response** items to see the request and the response that was captured during the test run

```
<soapenv:Envelope xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'>
<soapenv:Body>
<ns1:UPDACCOTOperation xmlns:ns1='http://www.UPDACCSTCUSTN2.Request.com'>
<ns1:update_account_record>
<ns1:account_key>
<ns1:sort_code>00-00-00</ns1:sort_code>
<ns1:account_number>123456789</ns1:account_number>
</ns1:account_key>
<ns1:account_change>500.5</ns1:account_change>
</ns1:update_account_record></ns1:UPDACCOTOperation>
</soapenv:Body>
</soapenv:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<UPDACCOTOperationResponse xmlns="http://www.UPDACCSTCUSTN2.Response.com">
<update_account_record_response>
<account_data>
<account_available_balance>2559.22</account_available_balance>
<account_actual_balance>2559.22</account_actual_balance>
</account_data>
</update_account_record_response>
</UPDACCOTOperationResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Well done. You have now successfully changed, rebuilt and executed a new Galasa test that interacts with both a web service and a set of 3270 data to both exercise and validate the correct operation of the application.

Note that in the case where this test is running within the Galasa eco-system the **StoredArtifactRoot** that we used in this example will point to the results archive in the ecosystem without you needing to update the source code.

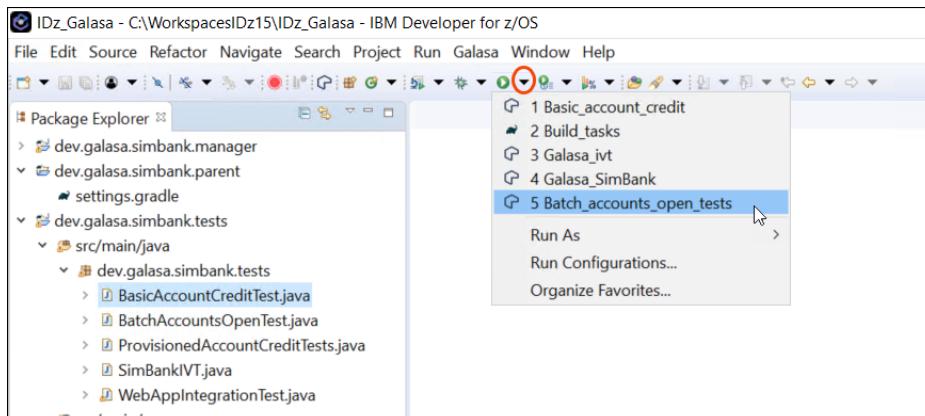
8.12 ► Use **Ctrl + Shift + F4** to close all opened editors.

PART #3 – Execute a Batch test

Section 9 Execute a batch test within the Galasa framework

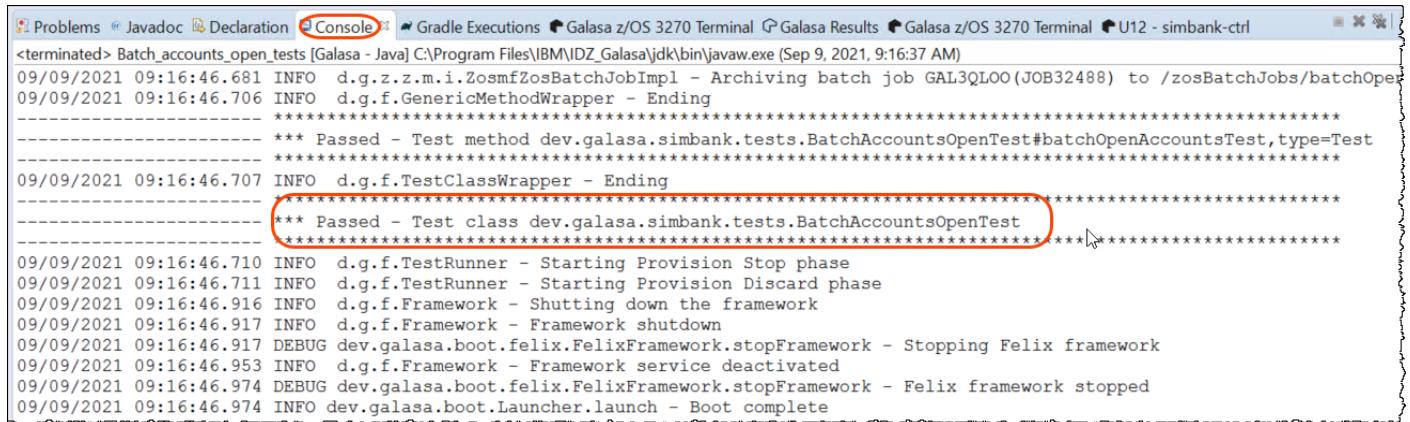
This test executes a sample JCL Batch job against the z/OS Application to load a set of accounts into the application.

9.1 ► Click on the drop down to the right of the green play icon and select **Batch_accounts_open_tests** to execute the batch and 3270 automated test



9.2 The test will execute.

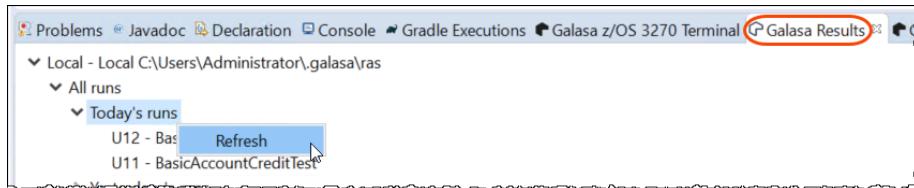
► Using the **Console** view you will see the message as below:



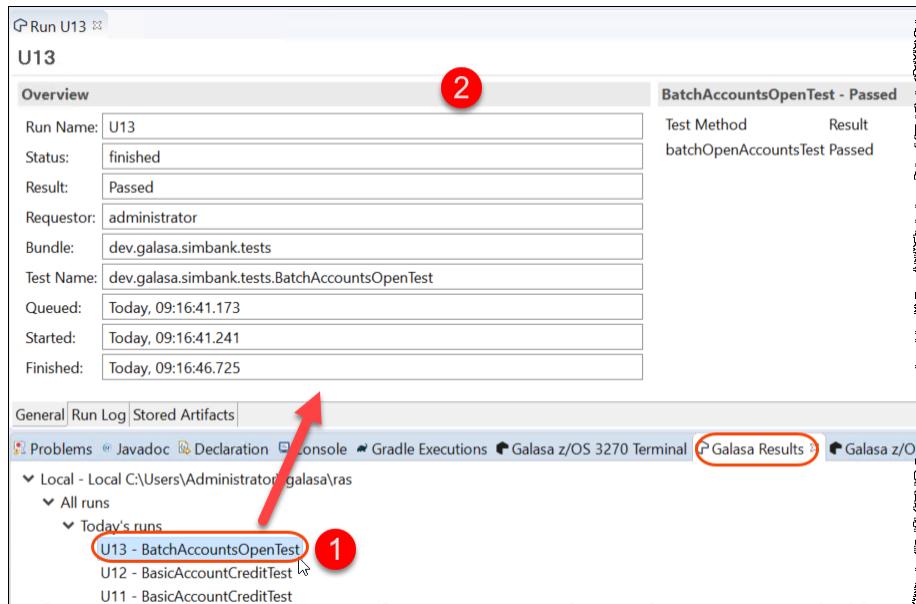
The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The log output is as follows:

```
<terminated> Batch_accounts_open_tests [Galasa - Java] C:\Program Files\IBM\IDZ_Galasa\jdk\bin\javaw.exe (Sep 9, 2021, 9:16:37 AM)
09/09/2021 09:16:46.681 INFO d.g.z.z.m.i.ZosmfZosBatchJobImpl - Archiving batch job GAL3QLOO(JOB32488) to /zosBatchJobs/batchOpen
09/09/2021 09:16:46.706 INFO d.g.f.GenericMethodWrapper - Ending
*****
*** Passed - Test method dev.galasa.simbank.tests.BatchAccountsOpenTest#batchOpenAccountsTest,type=Test
*****
09/09/2021 09:16:46.707 INFO d.g.f.TestClassWrapper - Ending
*****
*** Passed - Test class dev.galasa.simbank.tests.BatchAccountsOpenTest
*****
09/09/2021 09:16:46.710 INFO d.g.f.TestRunner - Starting Provision Stop phase
09/09/2021 09:16:46.711 INFO d.g.f.TestRunner - Starting Provision Discard phase
09/09/2021 09:16:46.916 INFO d.g.f.Framework - Shutting down the framework
09/09/2021 09:16:46.917 INFO d.g.f.Framework - Framework shutdown
09/09/2021 09:16:46.917 DEBUG dev.galasa.boot.felix.FelixFramework.stopFramework - Stopping Felix framework
09/09/2021 09:16:46.953 INFO d.g.f.Framework - Framework service deactivated
09/09/2021 09:16:46.974 DEBUG dev.galasa.boot.felix.FelixFramework.stopFramework - Felix framework stopped
09/09/2021 09:16:46.974 INFO dev.galasa.boot.Launcher.launch - Boot complete
```

9.3 ► As before in the **Galasa Results** view, right click on **Today's runs** and select **refresh** to load the latest test



9.4 ► Double click on the last run for **BatchAccountsOpenTest** and examine the results



The screenshot shows the Eclipse IDE interface with the 'Run U13' view open. The 'U13' run is selected. The 'Overview' tab is active, showing the following details:

Run Name:	U13
Status:	finished
Result:	Passed
Requestor:	administrator
Bundle:	dev.galasa.simbank.tests
Test Name:	dev.galasa.simbank.tests.BatchAccountsOpenTest
Queued:	Today, 09:16:41.173
Started:	Today, 09:16:41.241
Finished:	Today, 09:16:46.725

The 'BatchAccountsOpenTest - Passed' section shows the test method and result:

Test Method	Result
batchOpenAccountsTest	Passed

A red arrow points from the 'Galasa Results' tab in the top bar to the 'U13 - BatchAccountsOpenTest' entry in the 'Today's runs' list. A red circle with the number '1' is on the 'Today's runs' list, and another red circle with the number '2' is on the 'BatchAccountsOpenTest - Passed' section.

Section 10 Examine the source code to see how the Galasa test works

Let's see the source code created for this test.

- 10.1 ► Within the *Package Explorer* view double click on the **BatchAccountsOpenTest** to open the source code for this test

The screenshot shows the Eclipse IDE interface. The title bar says "IDz_Galasa - dev.galasa.simbank.tests/src/main/java/dev/galasa/simbank/tests/BatchAccountsOpenTest.java - C:\Workspaces\IDz15\IDz_Galasa - IBM". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Galasa, Window, Help. The toolbar has various icons for file operations. The left side shows the "Package Explorer" view with a tree structure of Java packages and files. A red arrow points to the "BatchAccountsOpenTest.java" file under "src/main/java/dev.galasa.simbank.tests". The right side shows the "Run U13" editor with the source code for "BatchAccountsOpenTest.java". The code includes annotations like @Test, @ZosImage, and @ZosBatch, and defines class variables for IZosImage and IZosBatch.

```
2 * Licensed Materials - Property of IBM
3 package dev.galasa.simbank.tests;
4
5 import java.io.IOException;
6
7
8 @Test
9 public class BatchAccountsOpenTest {
10     @ZosImage(imageTag = "SIMBANK")
11     public IZosImage image;
12
13     @ZosBatch(imageTag="SIMBANK")
14     public IZosBatch zosBatch;
15
16     @ZosBatchJobname(imageTag="SIMBANK")
17     public IZosBatchJobname zosBatchJobname;
18 }
```

This test is very similar in style to the previous test, however as it needs to interact with a z/OS batch job it needs to declare two new annotations to instruct the Galasa framework that we need to use the z/OS Batch Manager. These can be seen on lines **36-37** where we declare a **zosBatch** and on 39-40 where we ask Galasa to define a unique name that we want to use in our batch job name

- 10.2 ► Scroll down a bit and verify:

- On Lines **58-72** we create some input data for the batch job. In this case it is just a list of account numbers. This input is used to populate a *HashMap* in the same way we did for the webservice call.

The screenshot shows the "Run U13" editor with the source code for "BatchAccountsOpenTest.java". A red box highlights the code from line 58 to 72, which creates a list of account numbers. A red arrow points to the bottom right corner of the code area.

```
55     @Test
56     public void batchOpenAccountsTest() throws TestBundleResourceException, IOException, ZosBatchException {
57         // Create a list of accounts to create
58         List<String> accountList = new LinkedList<>();
59         accountList.add("90100001,20-40-60,1000");
60         accountList.add("90100002,20-40-60,1000");
61         accountList.add("90100003,20-40-60,1000");
62         accountList.add("90100004,20-40-60,1000");
63         accountList.add("90100005,20-40-60,1000");
64         accountList.add("90100006,20-40-60,1000");
65         accountList.add("90100007,20-40-60,1000");
66         accountList.add("90100008,20-40-60,1000");
67         accountList.add("90100009,20-40-60,1000");
68
69         // Create the substitution parameters for the JCL
70         HashMap<String, Object> parameters = new HashMap<>();
71         parameters.put("CONTROL", "ACCOUNT_OPEN");
72         parameters.put("DATAIN", String.join("\n", accountList));
73     }
74 }
```

- On lines 78-82 we use the Galasa API to submit the batch job and to wait for the response. Again notice that this is very simple code and the Galasa framework is doing all the hard work of interacting with z/OS.

```

77 // Submit the JCL
78 IZosBatchJob batchJob = zosBatch.submitJob(jcl, zosBatchJobname);
79
80 // Wait for the batch job to complete
81 logger.info("batchJob.toString() = " + batchJob.toString());
82 int rc = batchJob.waitForJob();
83
84 // If highest CC was not 0, fail the test
85 if (rc != 0) {
86     // Print the job output to the run log
87     batchJob.retrieveOutput().forEach(jobOutput ->
88         logger.info("batchJob.retrieveOutput(): " + jobOutput.getDdname() + "\n" + jobOutput.getRecords());
89     );
90     Fail.fail("Batch job failed RETCODE=" + batchJob.getRetcode() + " Check batch job output");
91 }
92
93 logger.info("Batch job complete RETCODE=" + batchJob.getRetcode());

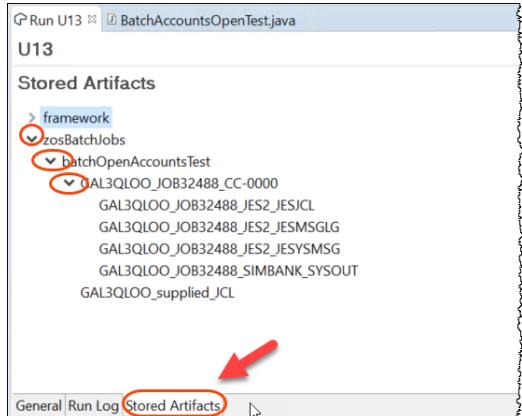
```

Section 11 Examine the Run Editor to see the stored artifacts

Let's see the artifacts that were stored .

11.1 ► The Run Editor for the *BatchAccountsOpenTest* should already be open, if not refresh the Galasa Results view and open the item for the test from that view

► Open the **Stored Artifacts** tab of the Run Editor and expand the **zosBatchJobs** element within the view



Note that all of the spool files from the batch job were pulled back from z/OS including the *JESJCL*, *JESMSLG*, *JESSYSMSG* as well as any unique spool files for this job.

Note that as we generate jobnames the names that appear in the screen shots might differ from your example. Galasa also logs the JCL that was supplied to the manager.

11.2 ► Double click on the **SIMBANK_SYSOUT** spool file

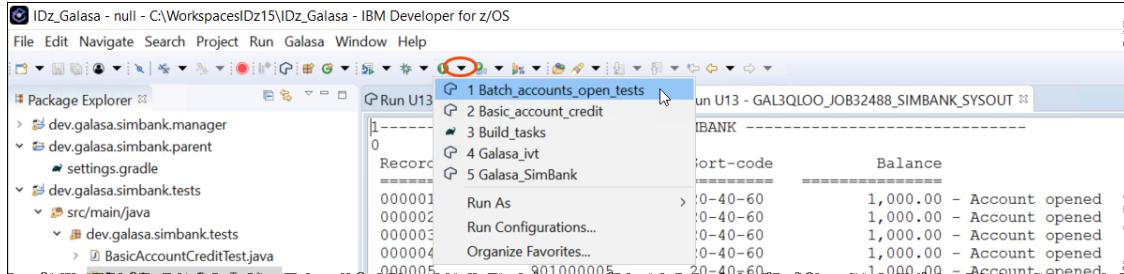
Record Number	Account Number	Sort-code	Balance
000001	901000001	20-40-60	1,000.00 - Account opened
000002	901000002	20-40-60	1,000.00 - Account opened
000003	901000003	20-40-60	1,000.00 - Account opened
000004	901000004	20-40-60	1,000.00 - Account opened
000005	901000005	20-40-60	1,000.00 - Account opened
000006	901000006	20-40-60	1,000.00 - Account opened
000007	901000007	20-40-60	1,000.00 - Account opened
000008	901000008	20-40-60	1,000.00 - Account opened
000009	901000009	20-40-60	1,000.00 - Account opened

0 Records read 9
Records rejected 0
Records processed 9

Congratulations the test ran and successfully the batch job

11.3. Note that the test passes or fails on the success of the batch job by looking at the return code. Repeat the execution of the test again and check the result.

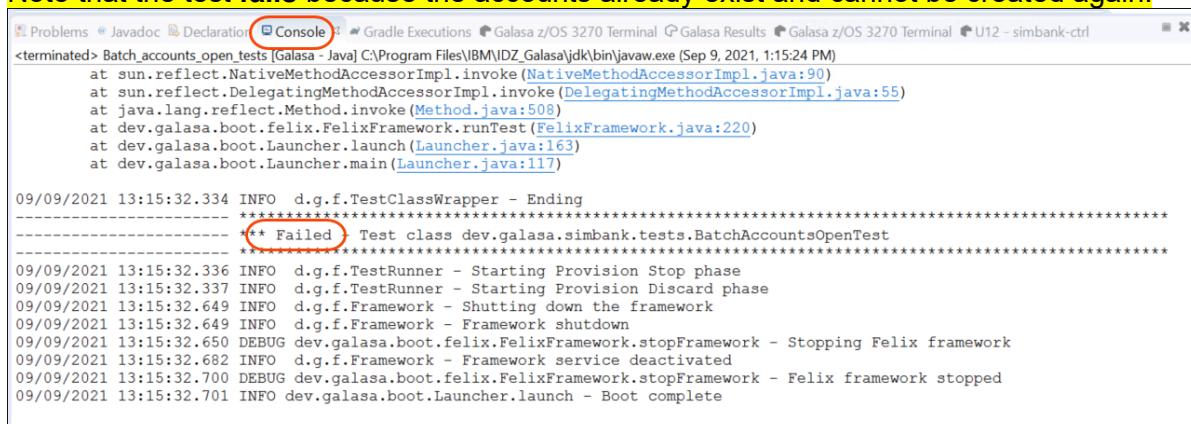
► Click on the drop down to the right of the green play icon and select **Batch_accounts_open_tests** to execute the batch and 3270 automated test



11.4 The test will execute.

► Using the **Console** view you will see the message as below:

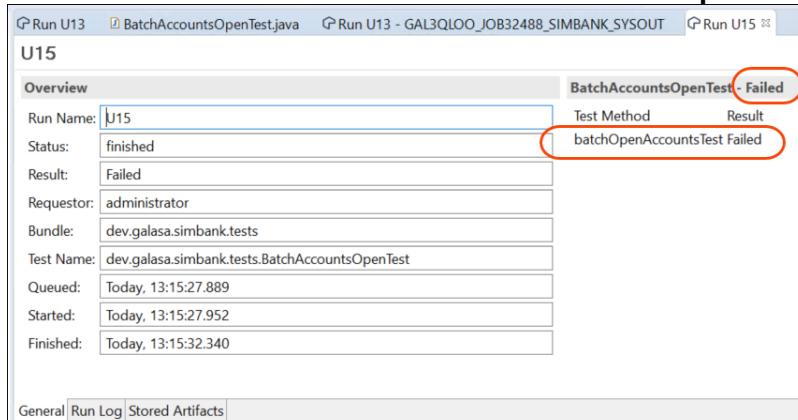
Note that the test fails because the accounts already exist and cannot be created again.



11.5 ► As before in the **Galasa Results** view, right click on **Today's runs** and select **refresh** to load the latest test

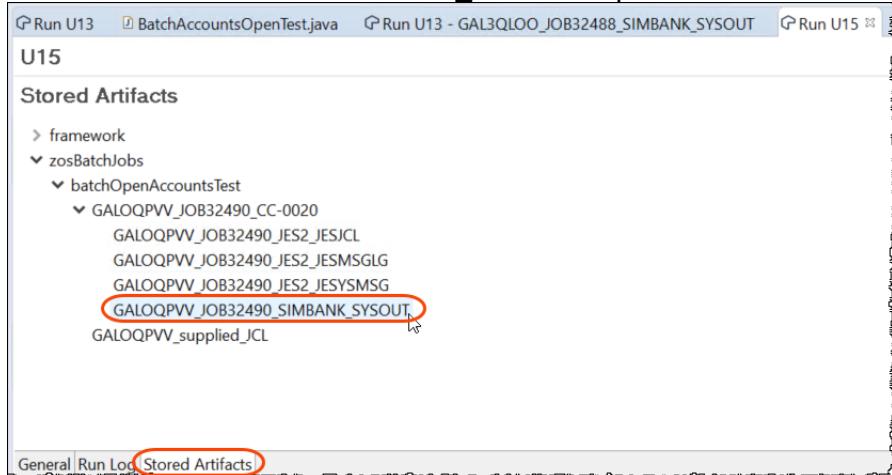


11.6 ► Double click on the run for **BatchAccountsOpenTest** and examine the results



11.7 ► Open the **Stored Artifacts** tab of the Run Editor and expand the **zosBatchJobs** element within the view

► Double click on the **SIMBANK_SYSOUT** spool file



11.8 See the results below.

The screenshot shows the 'Run Editor' interface with the 'Run Log' tab selected. The output window displays the results of the 'GALOQPVV_JOB32490_SIMBANK_SYSOUT' run. The log shows the following:

```
1----- SIMBANK -----
0
Record Number Account Number Sort-code Balance
=====
0 ERROR: No valid input records supplied
0 Records read 9
Records rejected 9
Records processed 0
1----- SIMBANK -----
0
0 Error Report
Record Number Record Message
=====
000001 901000001,20-40-60,1000 INVALID INPUT - Account exists
000002 901000002,20-40-60,1000 INVALID INPUT - Account exists
000003 901000003,20-40-60,1000 INVALID INPUT - Account exists
000004 901000004,20-40-60,1000 INVALID INPUT - Account exists
000005 901000005,20-40-60,1000 INVALID INPUT - Account exists
000006 901000006,20-40-60,1000 INVALID INPUT - Account exists
000007 901000007,20-40-60,1000 INVALID INPUT - Account exists
000008 901000008,20-40-60,1000 INVALID INPUT - Account exists
000009 901000009,20-40-60,1000 INVALID INPUT - Account exists
```

CONGRATULATIONS – you have completed this lab. You have run 3 Galasa tests from a local IDE workstation which connected to a mainframe using combinations of 3270, web service and batch to exercise and validate the date within a z/OS application.

Don't forget to find out more at <https://galasa.dev>



© Copyright IBM Corporation 2019.

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.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

Other company, product and service names may be trademarks or service marks of others.

