

DevOps on Z Proof of Technology (PoT)

Exercises Using zDT on Cloud

Updated May 11, 2023

OVERVIEW.....	13
LAB 1 - WORKING WITH MAINFRAME USING COBOL AND DB2 (90 - 120 MINUTES)	15
SECTION 1 – CONNECT TO A Z/OS SYSTEM	16
____ 1.1 WORKING WITH THE IDz WORKSPACE	16
____ 1.2 CONNECTING TO THE Z/OS REMOTE SYSTEM	18
SECTION 2 EXECUTE THE DB2/COBOL BATCH PROGRAM AND VERIFY THE ABEND.....	19
____ 2.1 SUBMIT A COBOL/DB2 BATCH TO EXECUTE	20
____ 2.2 ADD Z/OS RESOURCES TO THE MVS SUBPROJECT	22
____ 2.3 SUBMIT A JCL TO EXECUTE THE COBOL PROGRAM	23
SECTION 3. USE FAULT ANALYZER TO IDENTIFY THE CAUSE OF THE ABEND	25
____ 3.1 USING FAULT ANALYZER PERSPECTIVE	26
SECTION 4. USING THE IBM Z/OS DEBUGGER FOR A TEMPORARY FIX	32
____ 4.0 BE SURE THAT IDz CLIENT IS LISTENING ON PORT 8001.....	32
____ 4.1 SUBMIT THE JCL TO INVOKE THE DEBUG.....	32
____ 4.2 (OPTIONAL) USING THE VISUAL DEBUGGER FOR STACK PATTERN BREAKPOINTS.....	41
____ 4.3 ACCESSING THE Z/OS JES	44
SECTION 5. MODIFY THE COBOL CODE TO FIX THE BUG	46
____ 5.1 USING THE EDITOR WITH Z/OS COBOL PROGRAMS	46
____ 5.2 FIXING THE PROGRAM REGI0B	52
____ 5.3 COMPILE AND LINK REGI0B.....	56
____ 5.4 EXECUTE THE COBOL/DB2 PROGRAM.....	61
SECTION 6. USING THE CODE COVERAGE	64
____ 6.1 CREATING A JCL TO RUN THE CODE COVERAGE	64
____ 6.2 SUBMIT THE JCL FOR Z/OS EXECUTION	66
SECTION 7. (OPTIONAL) EXECUTING SQL STATEMENTS WHEN EDITING THE PROGRAM.....	70
____ 7.0 STARTING IDz VERSION 14 AND CONNECT TO Z/OS	70
____ 7.1 CREATING A CONNECTION TO THE DB2 ON Z/OS	72
____ 7.2 RUNNING A SQL QUERY FROM COBOL PROGRAM	73
____ 7.3 USING THE SQL OUTLINE VIEW	77
____ 7.4 USING SQL VISUAL EXPLAIN	77
____ 7.5 REVERSE ENGINEER THE QUERY	78
____ 7.6 DISPLAYING DB2 TABLE CONTENT	80
SECTION 8. (OPTIONAL) USING FILE MANAGER.....	83
____ 8.1 VERIFY THAT YOU ARE CONNECTED TO THE PDTTOOLS COMMON COMPONENTS	83
____ 8.2 USING VIEW LOAD MODULE UTILITY	83
____ 8.3 WORKING WITH Z/OS DATA SETS	87
____ 8.4 WORKING WITH TEMPLATES.....	97
LAB 2 (OPTIONAL) CREATE URBancode DEPLOY INFRASTRUCTURE AND DEPLOY TO Z/OS.....	103
LAB ARCHITECTURE AND PROPOSED SCENARIO	103
PART 1 – CREATE THE URBancode APPLICATION INFRASTRUCTURE.....	104
TASK 2 – CREATE A UCD COMPONENT	107
TASK 3 – CREATE A SHIP LIST FILE AND Z/OS COMPONENT VERSION	109
TASK 4 - CREATE THE UCD COMPONENT VERSION FROM JCL	114
TASK 5 - CREATE UCD RESOURCES	120
TASK 6 - CREATE AN URBancode APPLICATION.....	124
TASK 7 - CREATE ENVIRONMENT	125
PART 2 - CREATE THE URBancode DEPLOYMENT PROCESSES	128
TASK 8 - CREATE A COMPONENTS PROCESS	128
TASK 9 - CREATE AN APPLICATION PROCESS.....	144
TASK 10 – ENVIRONMENT PROPERTIES CONFIGURATION	147
PART 3 – DEPLOY THE APPLICATION TO Z/OS CICS	152
TASK 11 – RUN AN APPLICATION PROCESS	152
TASK 12 – VERIFYING THE DEPLOY RESULTS AT Z/OS CICS	156

LAB 3A – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING REMOTE ASSETS (60 MINUTES)	158
OVERVIEW OF DEVELOPMENT TASKS	158
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL.....	159
____ 1.1 CONNECT TO z/OS AND EMULATE A CICS 3270 TERMINAL.....	159
____ 1.2 RUN CICS TRANSACTION HCAZ.....	161
SECTION 2 – IMPORT A Z/OS PROJECT	162
____ 2.1 IMPORTING THE LAB6B Z/OS PROJECT	163
SECTION 3 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	165
____ 3.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE	165
____ 3.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	166
SECTION 4. GENERATE, BUILD AND RUN THE UNIT TEST.	176
____ 4.1 GENERATING, BUILDING AND RUNNING THE TEST CASE PROGRAM.	176
____ 4.2 RUNNING zUNIT TEST CASE WITH CODE COVERAGE	179
SECTION 5. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.....	182
____ 5.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	182
____ 5.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS	183
____ 5.3 RERUNNING THE TEST CASE	184
____ 5.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	186
SECTION 6. RUN THE UNIT TEST FROM A BATCH JCL.	191
____ 6.1 RUNNING THE UNIT TEST FROM A BATCH JCL.....	191
LAB 3B – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING LOCAL ASSETS (60 MINUTES)	195
OVERVIEW OF DEVELOPMENT TASKS.....	195
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL.....	196
____ 1.1 CONNECT TO z/OS AND EMULATE A CICS 3270 TERMINAL.....	196
____ 1.2 RUN CICS TRANSACTION HCAZ.....	198
SECTION 2 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	199
____ 2.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE	199
____ 2.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	200
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST.	209
____ 3.1 GENERATING THE TEST CASE PROGRAM.	209
____ 3.2 BUILDING THE GENERATED TEST CASE PROGRAMS USING IBM DBB	210
____ 3.3 RUNNING THE TEST CASE,	216
____ 3.4 VERIFY THE JCL SUBMITTED	219
____ 3.5 RUNNING zUNIT TEST CASE CODE COVERAGE	219
SECTION 4. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.	223
____ 4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	223
____ 4.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS USING DBB.....	224
____ 4.3 RUNNING THE TEST CASE AGAIN	226
____ 4.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	229
SECTION 5. (OPTIONAL)RUN THE UNIT TEST FROM A BATCH JCL.....	233
____ 5.1 RUNNING THE UNIT TEST FROM A BATCH JCL.....	233
LAB 3C – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL/DB2 BATCH PROGRAM (60 MINUTES).....	236
OVERVIEW OF DEVELOPMENT TASKS.....	236
PART #1 – UNIT TEST ON PROGRAM DB2BATCH AND INTRODUCE A BUG.....	237
SECTION 1. RUN THE COBOL/DB2 BATCH PROGRAM USING JCL.....	237
____ 1.0 CONNECT TO z/OS USING IDz	237
____ 1.1 SUBMIT A PROVIDED JCL FOR EXECUTION	238
SECTION 2 – USE zUNIT TO RECORD THE BATCH EXECUTION.....	241
____ 2.1 UNDERSTANDING THE MAIN COBOL PROGRAM THAT READS FROM DB2 TABLE	241
____ 2.2 RECORDING THE BATCH JCL EXECUTION.....	243
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.	249

3.1 GENERATING THE COBOL TEST CASE PROGRAMS.....	249
3.2 GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.....	250
3.3 RUNNING THE TEST CASE,.....	253
3.4 VERIFY THE JCL SUBMITTED THAT RUNS THE TEST CASE	255
SECTION 4. MODIFY THE COBOL/DB2 PROGRAM (INTRODUCE A BUG) AND RERUN THE UNIT TEST	256
4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	256
4.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	258
4.3 RUNNING THE TEST CASE AGAIN	260
SECTION 5. RUN THE BATCH PROGRAM AND VERIFY THE BUG	262
5.1 VERIFY THE BUG ON THE PRINTED REPORT	262
SECTION 6. USE IDz TO FIX THE BUG, RECOMPILE THE COBOL/DB2 PROGRAM.	264
6.1 MODIFYING THE PROGRAM TO ELIMINATE THE BUG	264
6.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	265
SECTION 7. RERUN THE zUNIT AND VERIFY THAT THE BUG IS ELIMINATED.	267
7.1 RUNNING THE TEST CASE AGAIN	267
LAB 4 – USING IBM DEPENDENCY BASED BUILD WITH GIT, JENKINS AND UCD ON Z/OS (60 MINUTES).....	269
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	271
1.2 RUN CICS TRANSACTION J05P	273
SECTION 2 – LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE	275
2.1 CLONING THE GIT REPOSITORY.....	275
2.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	280
SECTION 3 –MODIFY THE COBOL CODE USING IDz	281
3.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE.....	281
SECTION 4. USE IDz DBB USER BUILD TO COMPILE/BIND AND PERFORM PERSONAL TESTS.	283
4.1 USING DEPENDENCY BASED BUILDING OPTION	283
4.2 VERIFY THE DBB USER BUILD RESULTS	287
4.3 ISSUING A CICS NEW COPY	288
4.4 RUNNING J05P CICS TRANSACTION	291
SECTION 5. PUSH AND COMMIT THE CHANGED CODE TO GIT.....	292
5.1 USING IDz WITH THE GIT PLUGIN TO COMPARE THE CHANGE VERSUS ORIGINAL	292
5.2 PUSH AND COMMIT THE CHANGES TO GIT	293
SECTION 6. USE JENKINS WITH GIT PLUGIN TO BUILD ALL THE MODIFIED CODE COMMITTED TO GIT.....	295
6.1 LOGON TO JENKINS USING A WEB BROWSER AND START THE Z/OS AGENT.....	295
6.2 STARTING THE JENKINS PIPELINE.....	296
6.3 CHECKING THE RESULTS.....	298
6.4 UNDERSTAND THE RESULTS	300
SECTION 7. USE JENKINS AND UCD PLUGIN TO DEPLOY RESULTS AND TEST THE CICS TRANSACTION AGAIN	303
7.1 CHECKING THE DEPLOY RESULTS (SECOND STAGE)	303
7.2 CHECKING URBANCODE DEPLOY LOGS	305
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 5 – (OPTIONAL) USING IBM Z VTP TO TEST A COBOL /CICS / DB2 TRANSACTION (60 MINUTES).....	317
OVERVIEW OF DEVELOPMENT TASKS.....	317
SECTION 1. USE VTP TO RECORD THE CICS/DB2 APPLICATION IN ACTION.....	318
1.1 EMULATE A 3270 TERMINAL AND CONNECT TO CICS VERSION 5.4	318
1.2 USING VTP TO RECORD THE HEALTH CARE APPLICATION TRANSACTION SEQUENCE.....	319
SECTION 2 – RUN A JCL TO EXECUTE THE RECORDED REPLAY SEQUENCE	324

____ 2.1 CONNECT TO Z/OS USING IDz.....	324
____ 2.2 SUBMIT A PROVIDED VTP JCL FOR EXECUTION	325
SECTION 3 – MODIFY ONE PROGRAM (INTRODUCE A BUG) AND RERUN THE VTP JCL	328
____ 3.1 MODIFYING ONE COBOL PROGRAM INTRODUCING A BUG.....	328
____ 3.2 BUILDING THE MODIFIED PROGRAM USING DBB	329
____ 3.3 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM.....	331
SECTION 4. RUN THE CICS TRANSACTION AND VERIFY THE BUG	334
____ 4.1 ISSUING A CICS NEW COPY USING 3270 EMULATION TERMINAL.....	334
____ 4.2 EXECUTING HCAZ TRANSACTION	336
SECTION 5 USE IDz TO FIX THE BUG AND RECOMPILE/BIND THE PROGRAM.	338
____ 5.1 MODIFYING ONE COBOL PROGRAM AGAIN TO REMOVE THE BUG	338
____ 5.2 REBUILDING THE FIXED PROGRAM USING DBB.....	339
SECTION 6 RERUN THE VTP JCL AND VERIFY THAT THE BUG IS ELIMINATED.	340
____ 6.1 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM.....	340
LAB 6 – (OPTIONAL) DEPLOYING COBOL/CICS/DB2 APPLICATION USING A GITLAB CI PIPELINE (60 MINUTES)	344
SECTION 1. REVIEW THE GITLAB ISSUE AND VERIFY THE BUG USING THE 3270 TERMINAL.....	346
____ 1.0 ACCESS GITLAB AND VERIFY THE ISSUE.	346
____ 1.1 CONNECT TO z/OS AND EMULATE A CICS 3270 TERMINAL.....	347
____ 1.2 RUN CICS TRANSACTION SSC1	348
SECTION 2 – LOAD THE SOURCE CODE FROM GITLAB TO THE LOCAL IDz WORKSPACE.....	350
____ 2.1 CLONING THE GIT REPOSITORY.....	350
____ 2.2 VERIFY THE CODE CLONED USING z/OS PROJECTS PERSPECTIVE	355
SECTION 3 – USE IDz TO RUN THE ZUNIT TEST CASE AND VERIFY THE ERROR.....	356
____ 3.1 RUNNING THE ZUNIT IN BATCH.....	356
____ 3.2 VERIFYING THE ZUNIT RESULTS	359
SECTION 4. MODIFY THE COBOL/CICS/DB2 PROGRAM THAT HAS THE BUG USING IDz.....	360
____ 4.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE.....	360
SECTION 5. USE IDz DBB USER BUILD TO COMPILE/LINK AND RUN THE ZUNIT.....	361
____ 5.1 USING DEPENDENCY BASED BUILDING OPTION	361
____ 5.2 VERIFY THE DBB USER BUILD RESULTS	367
____ 5.3 RUNNING THE ZUNIT AGAIN	368
SECTION 6. PUSH AND COMMIT THE CHANGED CODE TO GITLAB.....	371
____ 6.1 USING IDz AND GIT PERSPECTIVE TO COMPARE THE CHANGE VERSUS ORIGINAL.....	371
____ 6.2 PUSH AND COMMIT THE CHANGES TO GITLAB	371
SECTION 7. VERIFY THE GITLAB CI BUILD EXECUTION.....	374
____ 7.1 VERIFYING THE BUILD	374
SECTION 8. VERIFY THE GITLAB CI PACKAGE AND DEPLOY TO UCD AND TEST THE CICS TRANSACTION AGAIN USING 3270	376
____ 8.1 CHECKING THE PACKAGING RESULTS (SECOND STAGE).....	376
____ 8.2 CHECKING URBancode CREATED VERSION	377
____ 8.3 CHECKING THE DEPLOYMENT RESULTS (THIRD STAGE)	379
____ 8.4 CHECKING URBancode DEPLOY RESULTS	380
____ 8.5 TESTING THE CICS CODE DEPLOYED TO DEV ENVIRONMENT	382
LAB 7 – (OPTIONAL) MIGRATING PARTITIONED DATA SETS (PDS) TO A DISTRIBUTED GIT REPOSITORY. (50 MINUTES)	385
SECTION 1. LOGON TO IDz AND PREPARE THE NECESSARY DIRECTORIES AND FILES ON ZFS	387
____ 1.1 CONNECT TO z/OS USING IDz	387
____ 1.2 VERIFY THE Z/OS DATASETS TO BE MIGRATED	388
____ 1.3 CREATE DIRECTORIES AND SCRIPT MIGRATION FILE AT ZFS	390
SECTION 2. MIGRATING USING DBB MIGRATION TOOL TO A LOCAL ROCKET GIT REPOSITORY ON ZFS	393
____ 2.1 SETTING UP A LOCAL ROCKET GIT REPOSITORY ON ZFS	394
____ 2.2 RUN MIGRATION USING A MAPPING FILE	395
____ 2.3 SETTING THE FILE TAG AFTER THE MIGRATION	397

— 2.4 FINISHING MIGRATION PROCESS	399
SECTION 3. PUSH THE MIGRATED FILES TO GIT SERVER (GITLAB)	400
— 3.1 CREATING A NEW GITLAB REPOSITORY	400
— 3.2 CONNECT THE LOCAL ROCKET GIT REPOSITORY WITH THE GITLAB REPOSITORY	403
— 3.3 PUSH THE EXISTING ROCKET GIT CLIENT REPOSITORY TO THE GIT SERVER REPOSITORY	403
— 3.4 ACCESS GITLAB REPOSITORY TO VERIFY THE CODE MIGRATED	403
SECTION 4. LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE AND START WORKING WITH THE MIGRATED ASSETS	406
— 4.1 CLONING THE GIT SERVER REPOSITORY USING IDz	406
— 4.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	412
— 4.3 COMMIT THE UPDATE TO THE GIT SERVER REPOSITORY	417
LAB 8 - (OPTIONAL) APPLICATION DISCOVERY : FIND A CANDIDATE FUNCTION IN A CICS APPLICATION IN ORDER TO CREATE AN API	420
OVERVIEW OF DEVELOPMENT TASKS	420
SECTION 1 REQUEST THE ADDI Z TRIAL LAB (ASSUMED THAT YOU HAVE COMPLETED THIS ON DAY 1)	421
SECTION 2 – LAB : DISCOVER A CANDIDATE API AND FIND ITS INTERFACE	423
— 2.1 OPEN YOUR ADDI zTRIAL	424
— 2.2 THE DISCOVER SCENARIO WILL OPEN THE WORKSHOP INSTRUCTIONS AND START IDz IN THE ADDI PERSPECTIVE	425
LAB 9 - (OPTIONAL) RUNNING INTEGRATION TESTS USING GALASA (60 MINUTES)	427
OVERVIEW OF DEVELOPMENT TASKS	428
SECTION 1. START THE SAMPLE PROVIDED SIMBANK APPLICATION	429
SECTION 2. USE IBM PERSONAL COMMUNICATIONS (PCOM) TO LOG ONTO THE SAMPLE APPLICATION	431
SECTION 3. RUN THE IVT TEST TO VALIDATE THAT GALASA CAN CONNECT TO THE SIMBANK APPLICATION	435
SECTION 4. OPEN THE TEST RESULT EDITOR	436
SECTION 5. EXECUTE A MIXED WEB SERVICE AND 3270 TEST WITHIN THE GALASA FRAMEWORK	437
SECTION 6. EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS	439
SECTION 7. EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS	441
SECTION 8 CHANGE THE TEST TO LOG THE REQUEST AND RESPONSE FOR THE WEB SERVICE	444
SECTION 9 EXECUTE A BATCH TEST WITHIN THE GALASA FRAMEWORK	449
SECTION 10 EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS	452
SECTION 11 EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS	453
LAB 10 – (OPTIONAL) USING APPLICATION PERFORMANCE ANALYZER (APA) (60 MINUTES)	456
SECTION 1. CREATE A NEW OBSERVATION REQUEST FOR A JOB THAT IS NOT RUNNING YET	458
— 1.1 CONNECT TO THE ADFz COMMON COMPONENTS	458
— 1.2 BEGIN A NEW APA OBSERVATION SESSION	460
SECTION 2 – RUN A SAMPLE BATCH JOB TO COLLECT PERFORMANCE DATA	462
— 2.1 CONNECT TO Z/OS	462
— 2.2 FIND THE JCL TO BE SUBMITTED	463
— 2.3 SUBMIT THE JCL TO EXECUTE THE COBOL/DB2 BATCH PROGRAM	464
SECTION 3 – REVIEW SOME OF THE REPORTS CREATED	465
— 3.1 DOWNLOAD THE MOST RECENT APA OBSERVATION REPORTS	466
— 3.2 ANALYZING THE APA MEASUREMENT PROFILE REPORT (S01)	468
— 3.3 ANALYZING THE APA LOAD MODULE SUMMARY REPORT (S03)	471
— 3.4 MEASUREMENT ANALYSIS REPORT (S09)	472
— 3.5 ANALYZING THE APA CPU USAGE BY CATEGORY REPORT (C01)	473
— 3.6 ANALYZING THE APA DB2 MEASUREMENT PROFILE REPORT (F01)	475
— 3.7 ANALYZING THE APA DB2 ACTIVITY BY STATEMENT REPORT (F04)	476

Contents

OVERVIEW	13
LAB 1 - WORKING WITH MAINFRAME USING COBOL AND DB2 (90 - 120 MINUTES)	15

SECTION 1 – CONNECT TO A z/OS SYSTEM	16
1.1 WORKING WITH THE IDZ WORKSPACE	16
1.2 CONNECTING TO THE z/OS REMOTE SYSTEM	18
SECTION 2 EXECUTE THE DB2/COBOL BATCH PROGRAM AND VERIFY THE ABEND.....	19
2.1 SUBMIT A COBOL/DB2 BATCH TO EXECUTE	20
2.2 ADD z/OS RESOURCES TO THE MVS SUBPROJECT	22
2.3 SUBMIT A JCL TO EXECUTE THE COBOL PROGRAM	23
SECTION 3. USE FAULT ANALYZER TO IDENTIFY THE CAUSE OF THE ABEND	25
3.1 USING FAULT ANALYZER PERSPECTIVE	26
SECTION 4. USING THE IBM z/OS DEBUGGER FOR A TEMPORARY FIX	32
4.0 BE SURE THAT IDZ CLIENT IS LISTENING ON PORT 8001.....	32
4.1 SUBMIT THE JCL TO INVOKE THE DEBUG.....	32
4.2 (OPTIONAL) USING THE VISUAL DEBUGGER FOR STACK PATTERN BREAKPOINTS.....	41
4.3 ACCESSING THE z/OS JES	44
SECTION 5. MODIFY THE COBOL CODE TO FIX THE BUG	46
5.1 USING THE EDITOR WITH z/OS COBOL PROGRAMS	46
5.2 FIXING THE PROGRAM REGI0B	52
5.3 COMPILE AND LINK REGI0B.....	56
5.4 EXECUTE THE COBOL/DB2 PROGRAM.....	61
SECTION 6. USING THE CODE COVERAGE	64
6.1 CREATING A JCL TO RUN THE CODE COVERAGE	64
6.2 SUBMIT THE JCL FOR z/OS EXECUTION	66
SECTION 7. (OPTIONAL) EXECUTING SQL STATEMENTS WHEN EDITING THE PROGRAM.....	70
7.0 STARTING IDZ VERSION 14 AND CONNECT TO z/OS	70
7.1 CREATING A CONNECTION TO THE DB2 ON z/OS	72
7.2 RUNNING A SQL QUERY FROM COBOL PROGRAM	73
7.3 USING THE SQL OUTLINE VIEW	77
7.4 USING SQL VISUAL EXPLAIN	77
7.5 REVERSE ENGINEER THE QUERY	78
7.6 DISPLAYING DB2 TABLE CONTENT	80
SECTION 8. (OPTIONAL) USING FILE MANAGER.....	83
8.1 VERIFY THAT YOU ARE CONNECTED TO THE PDTTOOLS COMMON COMPONENTS.....	83
8.2 USING VIEW LOAD MODULE UTILITY	83
8.3 WORKING WITH z/OS DATA SETS	87
8.4 WORKING WITH TEMPLATES.....	97
LAB 2 (OPTIONAL) CREATE URBancode DEPLOY INFRASTRUCTURE AND DEPLOY TO z/OS.....	103
LAB ARCHITECTURE AND PROPOSED SCENARIO	103
PART 1 – CREATE THE URBancode APPLICATION INFRASTRUCTURE.....	104
TASK 2 – CREATE A UCD COMPONENT	107
TASK 3 – CREATE A SHIP LIST FILE AND z/OS COMPONENT VERSION	109
TASK 4 - CREATE THE UCD COMPONENT VERSION FROM JCL	114
TASK 5 - CREATE UCD RESOURCES	120
TASK 6 - CREATE AN URBancode APPLICATION.....	124
TASK 7 - CREATE ENVIRONMENT	125
PART 2 - CREATE THE URBancode DEPLOYMENT PROCESSES	128
TASK 8 - CREATE A COMPONENTS PROCESS	128
TASK 9 - CREATE AN APPLICATION PROCESS.....	144
TASK 10 – ENVIRONMENT PROPERTIES CONFIGURATION	147
PART 3 – DEPLOY THE APPLICATION TO z/OS CICS	152
TASK 11 – RUN AN APPLICATION PROCESS	152
TASK 12 – VERIFYING THE DEPLOY RESULTS AT z/OS CICS	156
LAB 3A – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING REMOTE ASSETS (60 MINUTES)	158

OVERVIEW OF DEVELOPMENT TASKS	158
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL.....	159
— 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL.....	159
— 1.2 RUN CICS TRANSACTION HCAZ.....	161
SECTION 2 – IMPORT A Z/OS PROJECT	162
— 2.1 IMPORTING THE LAB6B Z/OS PROJECT	163
SECTION 3 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	165
— 3.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE	165
— 3.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	166
SECTION 4. GENERATE, BUILD AND RUN THE UNIT TEST.	176
— 4.1 GENERATING, BUILDING AND RUNNING THE TEST CASE PROGRAM.....	176
— 4.2 RUNNING zUNIT TEST CASE WITH CODE COVERAGE	179
SECTION 5. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.....	182
— 5.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	182
— 5.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS	183
— 5.3 RERUNNING THE TEST CASE	184
— 5.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	186
SECTION 6. RUN THE UNIT TEST FROM A BATCH JCL.	191
— 6.1 RUNNING THE UNIT TEST FROM A BATCH JCL	191
LAB 3B – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING LOCAL ASSETS (60 MINUTES)	195
OVERVIEW OF DEVELOPMENT TASKS.....	195
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL.....	196
— 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL.....	196
— 1.2 RUN CICS TRANSACTION HCAZ.....	198
SECTION 2 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	199
— 2.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE	199
— 2.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	200
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST.	209
— 3.1 GENERATING THE TEST CASE PROGRAM.....	209
— 3.2 BUILDING THE GENERATED TEST CASE PROGRAMS USING IBM DBB	210
— 3.3 RUNNING THE TEST CASE,	216
— 3.4 VERIFY THE JCL SUBMITTED	219
— 3.5 RUNNING zUNIT TEST CASE CODE COVERAGE	219
SECTION 4. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.	223
— 4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	223
— 4.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS USING DBB.....	224
— 4.3 RUNNING THE TEST CASE AGAIN	226
— 4.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	229
SECTION 5. (OPTIONAL)RUN THE UNIT TEST FROM A BATCH JCL.....	233
— 5.1 RUNNING THE UNIT TEST FROM A BATCH JCL	233
LAB 3C – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL/DB2 BATCH PROGRAM (60 MINUTES).....	236
OVERVIEW OF DEVELOPMENT TASKS.....	236
PART #1 – UNIT TEST ON PROGRAM DB2BATCH AND INTRODUCE A BUG.....	237
SECTION 1. RUN THE COBOL/DB2 BATCH PROGRAM USING JCL.....	237
— 1.0 CONNECT TO Z/OS USING IDz	237
— 1.1 SUBMIT A PROVIDED JCL FOR EXECUTION	238
SECTION 2 – USE zUNIT TO RECORD THE BATCH EXECUTION.....	241
— 2.1 UNDERSTANDING THE MAIN COBOL PROGRAM THAT READS FROM DB2 TABLE	241
— 2.2 RECORDING THE BATCH JCL EXECUTION.....	243
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.....	249
— 3.1 GENERATING THE COBOL TEST CASE PROGRAMS.....	249
— 3.2 GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM	250

— 3.3 RUNNING THE TEST CASE	253
— 3.4 VERIFY THE JCL SUBMITTED THAT RUNS THE TEST CASE	255
SECTION 4. MODIFY THE COBOL/DB2 PROGRAM (INTRODUCE A BUG) AND RERUN THE UNIT TEST	256
— 4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	256
— 4.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	258
— 4.3 RUNNING THE TEST CASE AGAIN	260
SECTION 5. RUN THE BATCH PROGRAM AND VERIFY THE BUG	262
— 5.1 VERIFY THE BUG ON THE PRINTED REPORT	262
SECTION 6. USE IDz TO FIX THE BUG, RECOMPILE THE COBOL/DB2 PROGRAM	264
— 6.1 MODIFYING THE PROGRAM TO ELIMINATE THE BUG	264
— 6.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	265
SECTION 7. RERUN THE zUNIT AND VERIFY THAT THE BUG IS ELIMINATED	267
— 7.1 RUNNING THE TEST CASE AGAIN	267
LAB 4 – USING IBM DEPENDENCY BASED BUILD WITH GIT, JENKINS AND UCD ON Z/OS (60 MINUTES)	269
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	271
— 1.2 RUN CICS TRANSACTION J05P	273
SECTION 2 – LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE	275
— 2.1 CLONING THE GIT REPOSITORY	275
— 2.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	280
SECTION 3 –MODIFY THE COBOL CODE USING IDz	281
— 3.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE	281
SECTION 4. USE IDz DBB USER Build TO COMPILE/BIND AND PERFORM PERSONAL TESTS	283
— 4.1 USING DEPENDENCY BASED BUILDING OPTION	283
— 4.2 VERIFY THE DBB USER BUILD RESULTS	287
— 4.3 ISSUING A CICS NEW COPY	288
— 4.4 RUNNING J05P CICS TRANSACTION	291
SECTION 5. PUSH AND COMMIT THE CHANGED CODE TO GIT	292
— 5.1 USING IDz WITH THE GIT PLUGIN TO COMPARE THE CHANGE VERSUS ORIGINAL	292
— 5.2 PUSH AND COMMIT THE CHANGES TO GIT	293
SECTION 6. USE JENKINS WITH GIT PLUGIN TO BUILD ALL THE MODIFIED CODE COMMITTED TO GIT	295
— 6.1 LOGON TO JENKINS USING A WEB BROWSER AND START THE Z/OS AGENT	295
— 6.2 STARTING THE JENKINS PIPELINE	296
— 6.3 CHECKING THE RESULTS	298
— 6.4 UNDERSTAND THE RESULTS	300
SECTION 7. USE JENKINS AND UCD PLUGIN TO DEPLOY RESULTS AND TEST THE CICS TRANSACTION AGAIN	303
— 7.1 CHECKING THE DEPLOY RESULTS (SECOND STAGE)	303
— 7.2 CHECKING URBANCODE DEPLOY LOGS	305
— 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 5 – (OPTIONAL) USING IBM Z VTP TO TEST A COBOL /CICS / DB2 TRANSACTION (60 MINUTES)	317
OVERVIEW OF DEVELOPMENT TASKS	317
SECTION 1. USE VTP TO RECORD THE CICS/DB2 APPLICATION IN ACTION	318
— 1.1 EMULATE A 3270 TERMINAL AND CONNECT TO CICS VERSION 5.4	318
— 1.2 USING VTP TO RECORD THE HEALTH CARE APPLICATION TRANSACTION SEQUENCE	319
SECTION 2 – RUN A JCL TO EXECUTE THE RECORDED REPLAY SEQUENCE	324
— 2.1 CONNECT TO Z/OS USING IDz	324
— 2.2 SUBMIT A PROVIDED VTP JCL FOR EXECUTION	325

SECTION 3 – MODIFY ONE PROGRAM (INTRODUCE A BUG) AND RERUN THE VTP JCL	328
— 3.1 MODIFYING ONE COBOL PROGRAM INTRODUCING A BUG.....	328
— 3.2 BUILDING THE MODIFIED PROGRAM USING DBB	329
— 3.3 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM.....	331
SECTION 4. RUN THE CICS TRANSACTION AND VERIFY THE BUG	334
— 4.1 ISSUING A CICS NEW COPY USING 3270 EMULATION TERMINAL.....	334
— 4.2 EXECUTING HCAZ TRANSACTION	336
SECTION 5 Use IDz TO FIX THE BUG AND RECOMPILE/BIND THE PROGRAM.	338
— 5.1 MODIFYING ONE COBOL PROGRAM AGAIN TO REMOVE THE BUG	338
— 5.2 REBUILDING THE FIXED PROGRAM USING DBB.....	339
SECTION 6 RERUN THE VTP JCL AND VERIFY THAT THE BUG IS ELIMINATED.	340
— 6.1 RUNNING THE VTP JCL AGAIN AGAINST THE MODIFIED PROGRAM.....	340
LAB 6 – (OPTIONAL) DEPLOYING COBOL/CICS/DB2 APPLICATION USING A GITLAB CI PIPELINE (60 MINUTES)	344
SECTION 1. REVIEW THE GITLAB ISSUE AND VERIFY THE BUG USING THE 3270 TERMINAL.....	346
— 1.0 ACCESS GITLAB AND VERIFY THE ISSUE.	346
— 1.1 CONNECT TO z/OS AND EMULATE A CICS 3270 TERMINAL.....	347
— 1.2 RUN CICS TRANSACTION SSC1	348
SECTION 2 – LOAD THE SOURCE CODE FROM GITLAB TO THE LOCAL IDz WORKSPACE.....	350
— 2.1 CLONING THE GIT REPOSITORY.....	350
— 2.2 VERIFY THE CODE CLONED USING z/OS PROJECTS PERSPECTIVE	355
SECTION 3 – USE IDz TO RUN THE ZUNIT TEST CASE AND VERIFY THE ERROR.....	356
— 3.1 RUNNING THE ZUNIT IN BATCH.....	356
— 3.2 VERIFYING THE ZUNIT RESULTS	359
SECTION 4. MODIFY THE COBOL/CICS/DB2 PROGRAM THAT HAS THE BUG USING IDz.....	360
— 4.1 EDIT AND MODIFY THE CODE THAT SEND THE MESSAGE.....	360
SECTION 5. USE IDz DBB USER BUILD TO COMPILE/LINK AND RUN THE ZUNIT.....	361
— 5.1 USING DEPENDENCY BASED BUILDING OPTION	361
— 5.2 VERIFY THE DBB USER BUILD RESULTS	367
— 5.3 RUNNING THE ZUNIT AGAIN	368
SECTION 6. PUSH AND COMMIT THE CHANGED CODE TO GITLAB.....	371
— 6.1 USING IDz AND GIT PERSPECTIVE TO COMPARE THE CHANGE VERSUS ORIGINAL	371
— 6.2 PUSH AND COMMIT THE CHANGES TO GITLAB	371
SECTION 7. VERIFY THE GITLAB CI BUILD EXECUTION.....	374
— 7.1 VERIFYING THE BUILD	374
SECTION 8. VERIFY THE GITLAB CI PACKAGE AND DEPLOY TO UCD AND TEST THE CICS TRANSACTION AGAIN USING 3270	376
— 8.1 CHECKING THE PACKAGING RESULTS (SECOND STAGE).....	376
— 8.2 CHECKING URBancode CREATED VERSION	377
— 8.3 CHECKING THE DEPLOYMENT RESULTS (THIRD STAGE)	379
— 8.4 CHECKING URBancode DEPLOY RESULTS	380
— 8.5 TESTING THE CICS CODE DEPLOYED TO DEV ENVIRONMENT	382
LAB 7 – (OPTIONAL) MIGRATING PARTITIONED DATA SETS (PDS) TO A DISTRIBUTED GIT REPOSITORY. (50 MINUTES)	385
SECTION 1. LOGON TO IDz AND PREPARE THE NECESSARY DIRECTORIES AND FILES ON ZFS	387
— 1.1 CONNECT TO z/OS USING IDz	387
— 1.2 VERIFY THE Z/OS DATASETS TO BE MIGRATED	388
— 1.3 CREATE DIRECTORIES AND SCRIPT MIGRATION FILE AT ZFS	390
SECTION 2 . MIGRATING USING DBB MIGRATION TOOL TO A LOCAL ROCKET GIT REPOSITORY ON ZFS	393
— 2.1 SETTING UP A LOCAL ROCKET GIT REPOSITORY ON ZFS	394
— 2.2 RUN MIGRATION USING A MAPPING FILE	395
— 2.3 SETTING THE FILE TAG AFTER THE MIGRATION	397
— 2.4 FINISHING MIGRATION PROCESS	399
SECTION 3. PUSH THE MIGRATED FILES TO GIT SERVER (GITLAB)	400

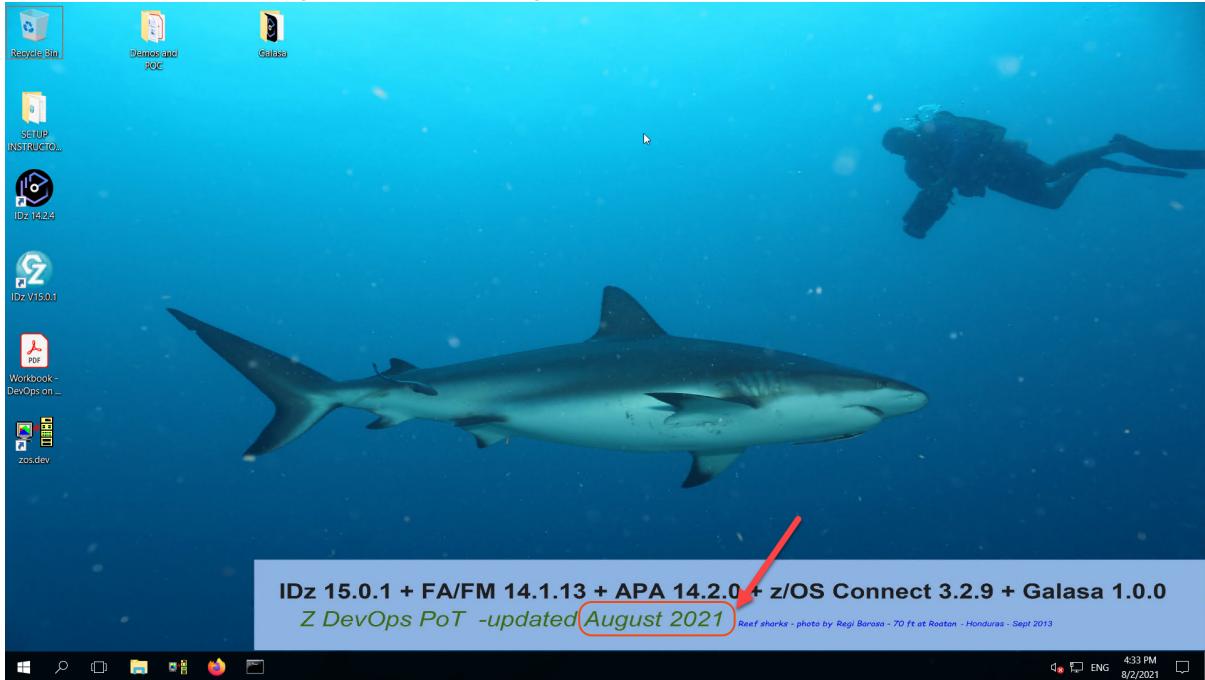
— 3.1 CREATING A NEW GITLAB REPOSITORY	400
— 3.2 CONNECT THE LOCAL ROCKET GIT REPOSITORY WITH THE GITLAB REPOSITORY	403
— 3.3 PUSH THE EXISTING ROCKET GIT CLIENT REPOSITORY TO THE GIT SERVER REPOSITORY	403
— 3.4 ACCESS GITLAB REPOSITORY TO VERIFY THE CODE MIGRATED	403
SECTION 4. LOAD THE SOURCE CODE FROM GIT TO THE LOCAL IDz WORKSPACE AND START WORKING WITH THE MIGRATED ASSETS	406
— 4.1 CLONING THE GIT SERVER REPOSITORY USING IDz	406
— 4.2 VERIFY THE CODE CLONED USING Z/OS PROJECTS PERSPECTIVE	412
— 4.3 COMMIT THE UPDATE TO THE GIT SERVER REPOSITORY	417
LAB 8 -(OPTIONAL) APPLICATION DISCOVERY : FIND A CANDIDATE FUNCTION IN A CICS APPLICATION IN ORDER TO CREATE AN API	420
OVERVIEW OF DEVELOPMENT TASKS	420
SECTION 1 REQUEST THE ADDI Z TRIAL LAB (ASSUMED THAT YOU HAVE COMPLETED THIS ON DAY 1)	421
SECTION 2 – LAB : DISCOVER A CANDIDATE API AND FIND ITS INTERFACE.....	423
— 2.1 OPEN YOUR ADDI zTRIAL	424
— 2.2 THE DISCOVER SCENARIO WILL OPEN THE WORKSHOP INSTRUCTIONS AND START IDz IN THE ADDI PERSPECTIVE	425
LAB 9 – (OPTIONAL) RUNNING INTEGRATION TESTS USING GALASA (60 MINUTES)	427
OVERVIEW OF DEVELOPMENT TASKS	428
SECTION 1. START THE SAMPLE PROVIDED <i>SIMBANK</i> APPLICATION	429
SECTION 2. USE IBM PERSONAL COMMUNICATIONS (PCOM) TO LOG ONTO THE SAMPLE APPLICATION.....	431
SECTION 3. RUN THE IVT TEST TO VALIDATE THAT GALASA CAN CONNECT TO THE <i>SIMBANK</i> APPLICATION	435
SECTION 4. OPEN THE TEST RESULT EDITOR	436
SECTION 5. EXECUTE A MIXED WEB SERVICE AND 3270 TEST WITHIN THE GALASA FRAMEWORK	437
SECTION 6. EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS	439
SECTION 7. EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS.....	441
SECTION 8. CHANGE THE TEST TO LOG THE REQUEST AND RESPONSE FOR THE WEB SERVICE	444
SECTION 9. EXECUTE A BATCH TEST WITHIN THE GALASA FRAMEWORK	449
SECTION 10 EXAMINE THE SOURCE CODE TO SEE HOW THE GALASA TEST WORKS	452
SECTION 11 EXAMINE THE RUN EDITOR TO SEE THE STORED ARTIFACTS	453

Overview

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

This material was built using a Windows on cloud that was updated on **August 2021**

Here the desktop background of this image:



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

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

LAB 1 - Working with mainframe using COBOL and DB2

(90 - 120 minutes)

Updated September 24, 2021 (created by [Regi](#), Reviewed by [Wilbert](#))

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

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

Overview of development tasks

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

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

Section 1 – Connect to a z/OS System

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

1.1 Working with the IDz workspace

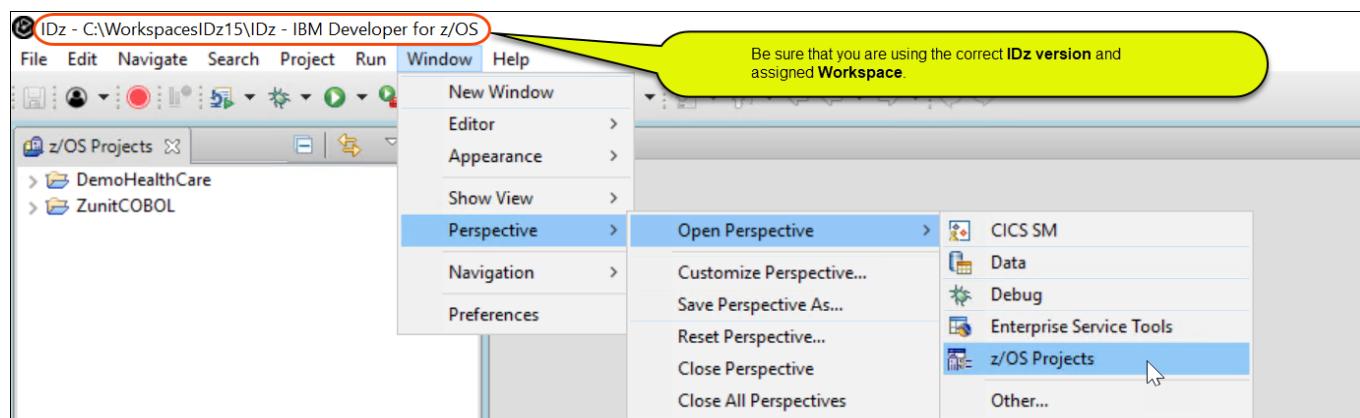
1.1.1 Start IBM Developer for z Systems version 15

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

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



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



z/OS Projects perspective

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

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



You can close and open views to customize the perspective.

To open one of the views that were closed:

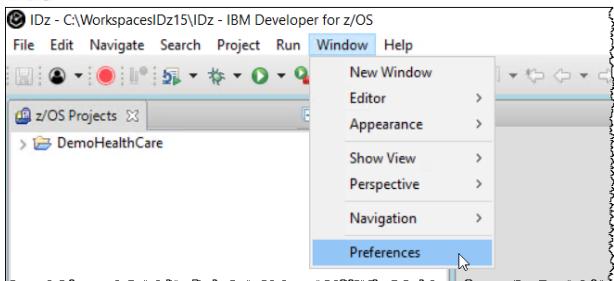
1. In the workbench, select **Window > Show View**.

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

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

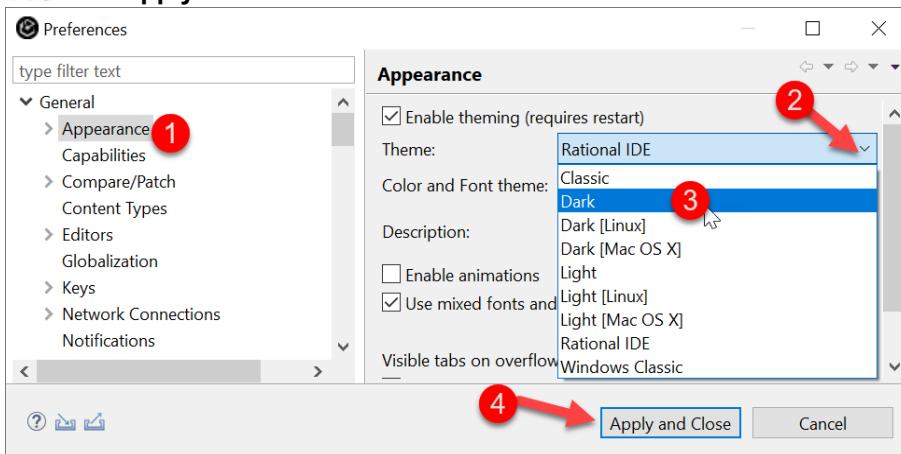
1.1.3 Starting on version 15 the developer can change the workspace appearance. If you want to try that

▶ Click **Windows > Preferences**



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

▶ Click **Apply and Close**

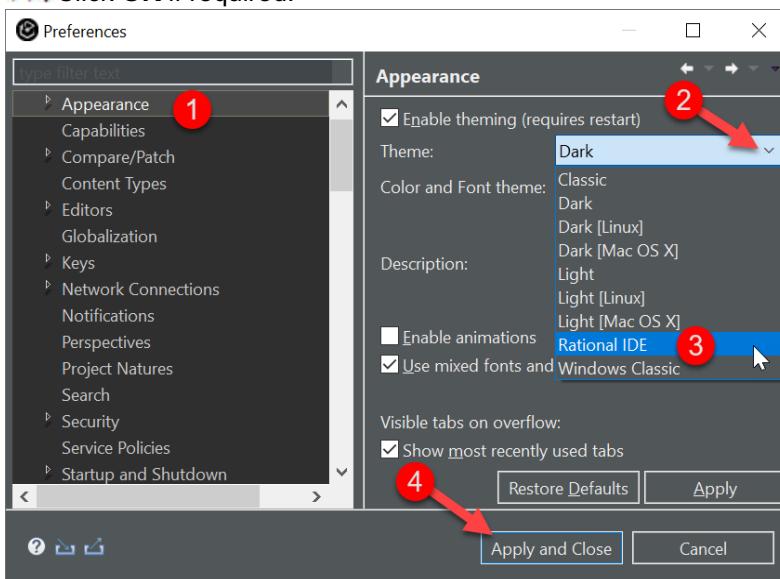


1.1.5 Since all picture here were done using “Rational IDE” theme we recommend to return to the default.

▶ Again, click **Windows > Preferences**

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

▶ Click **OK** if required.



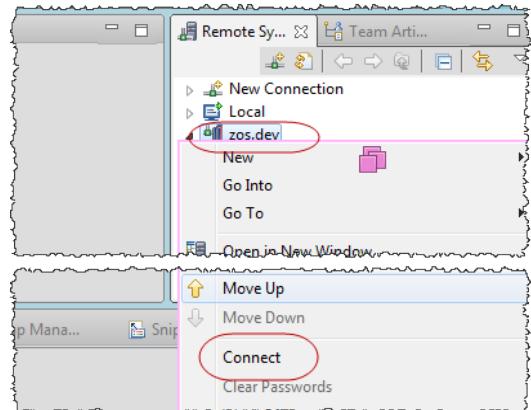
1.2 Connecting to the z/OS Remote system

1.2.1 To Connect to the z/OS system:

► Using the *Remote Systems* view on the top and right side of the screen:

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

Notice: Since you are using a cloud instance the response to the right click may be delayed first time.

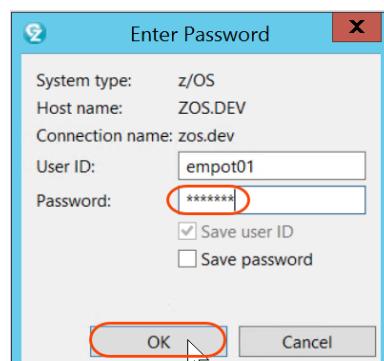


1.2.2 You will be prompted for your z/OS userid and password.

► 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.

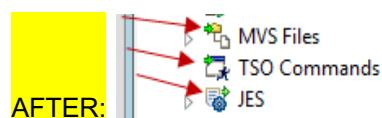
1.2.3 ► Click **OK** to connect to z/OS.



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.



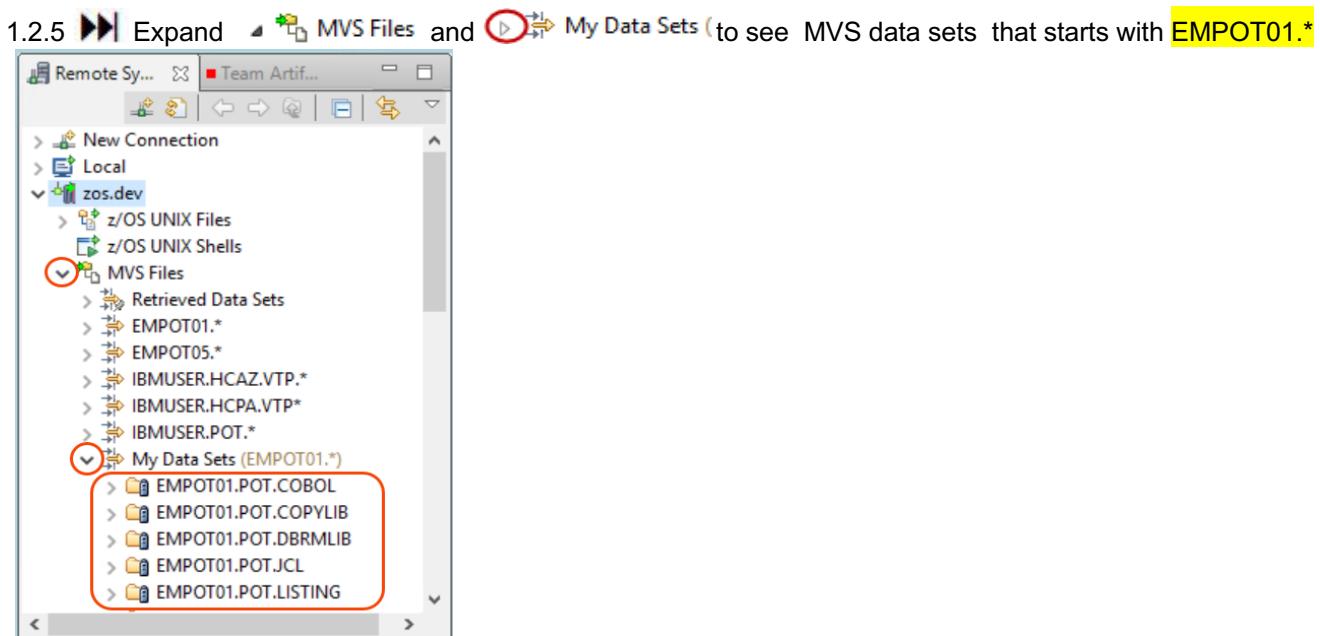
In case of connection errors...

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

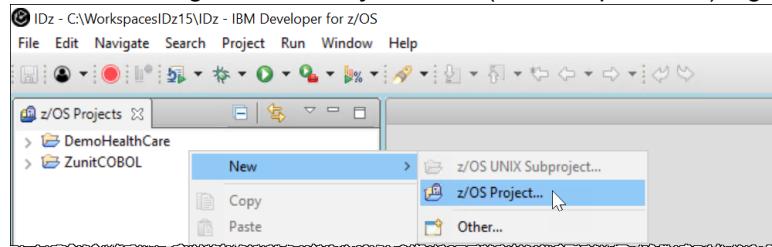


→ Creating a z/OS project

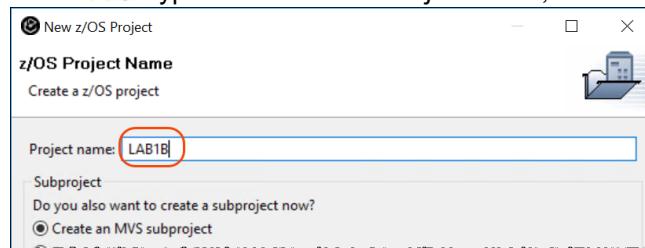
2.1 Submit a COBOL/DB2 batch to execute

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...**

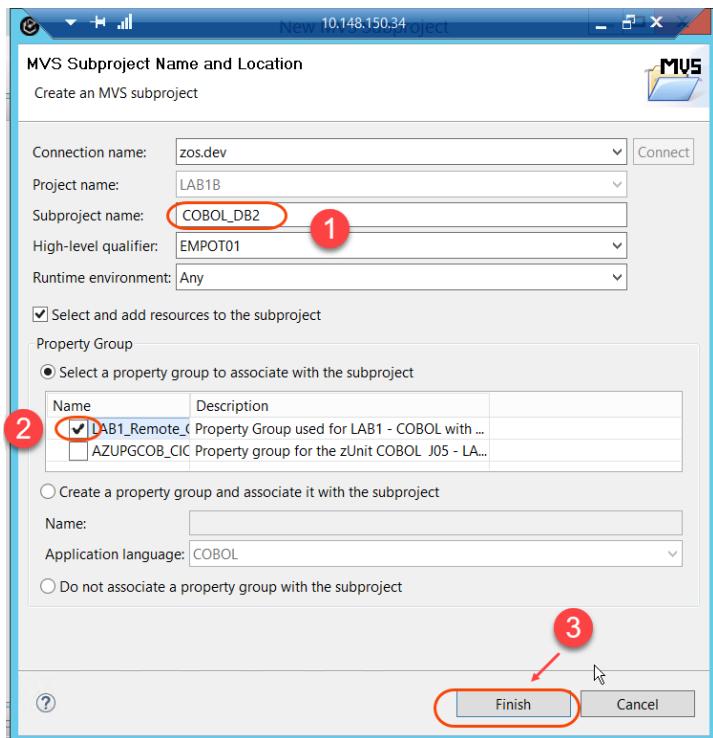


2.1.2 ► Type **LAB1B** as the *Project name*, select **Create an MVS subproject** and click **Finish**.

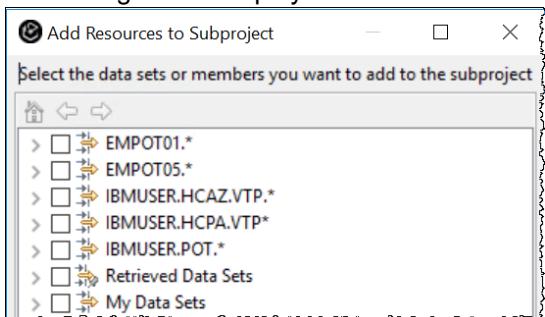


2.1.3 ► Type **COBOL_DB2** as *Subproject Name*, select **LAB1_Remote_Cobol..** as the property group and click **Finish**.

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



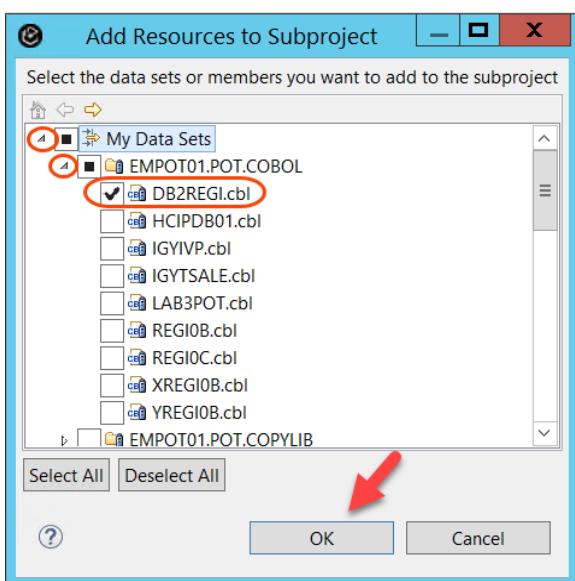
Property groups provide a way to manage resource properties, share them easily across systems, projects, resources, and users, and maintain consistency in your development and build environment.

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**

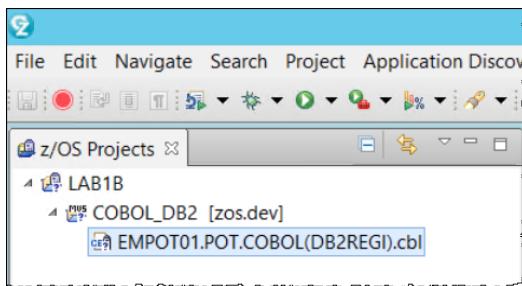


2.2.2 You should see a z/OS Project named **LAB1B** in your *z/OS Projects view*.

► 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.



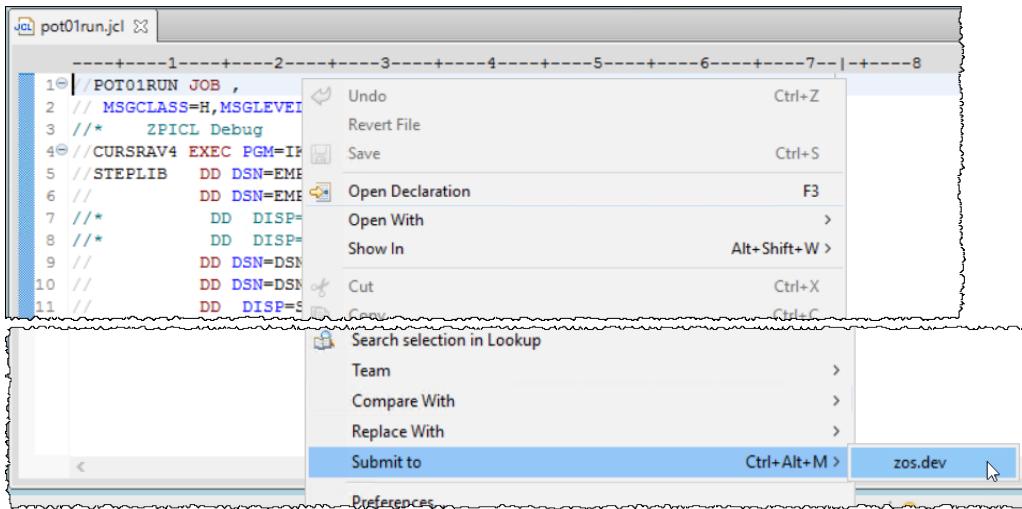
2.3 Submit a JCL to execute the COBOL program

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.

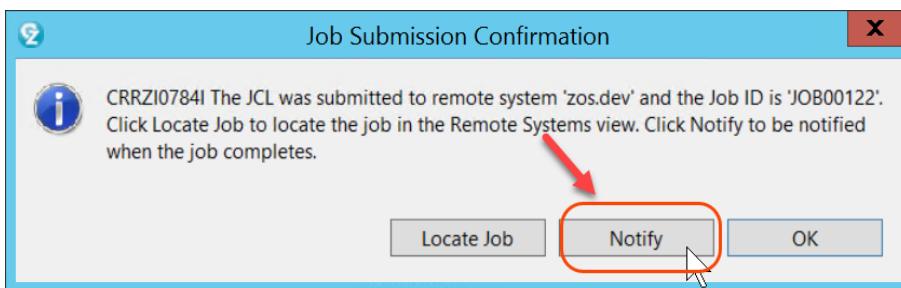
2.3.1 ► Using *Remote System View* (on top and right), scroll up to locate the file **pot01run.jcl** under **Local/Local Files/ADF_POT/empot01** and double click to edit.



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

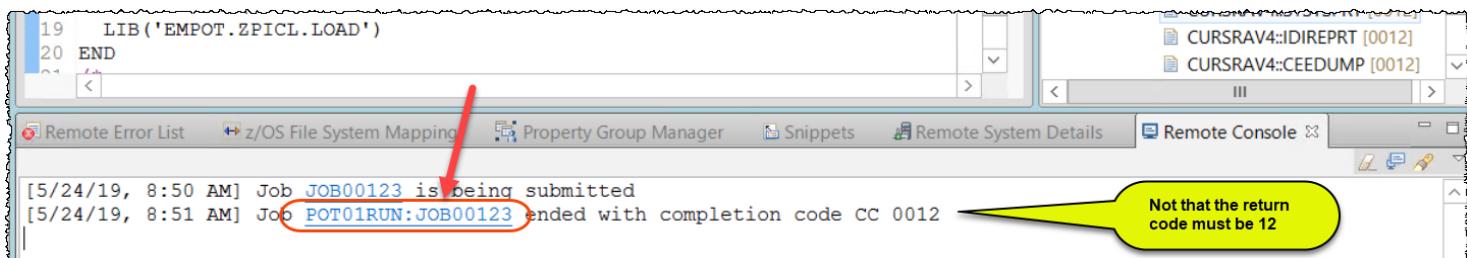


2.3.3 ►| When the dialog pop up appears, click **Notify**. This allows you to be notified when the execution is ended.



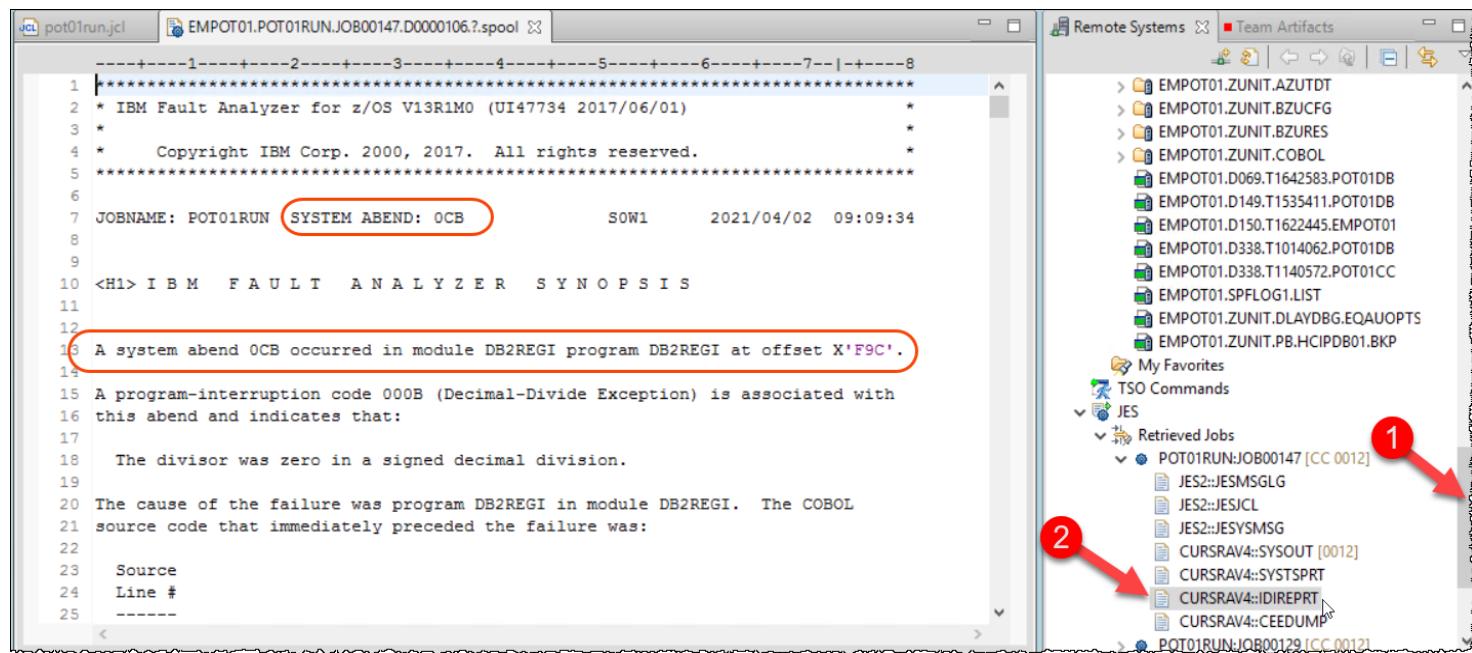
2.3.4 Under *Remote Console*, you will be notified when execution is completed.

►| 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.

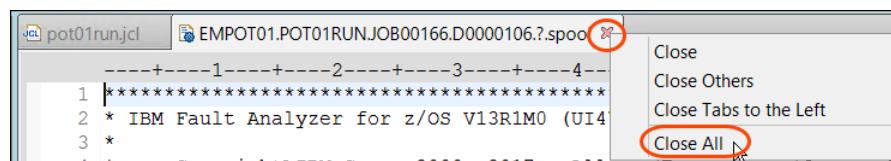


2.3.5 ► Under **Remote Systems** view scroll down, expand **JES > Retrieved Jobs > POT01RUN:JOB00xxx** and **double click CURSRAV4::IDIREPRT** step and you will see the *Fault Analyzer* report showing an **OCB ABEND**. Your mission is to fix that bug.

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**.



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?

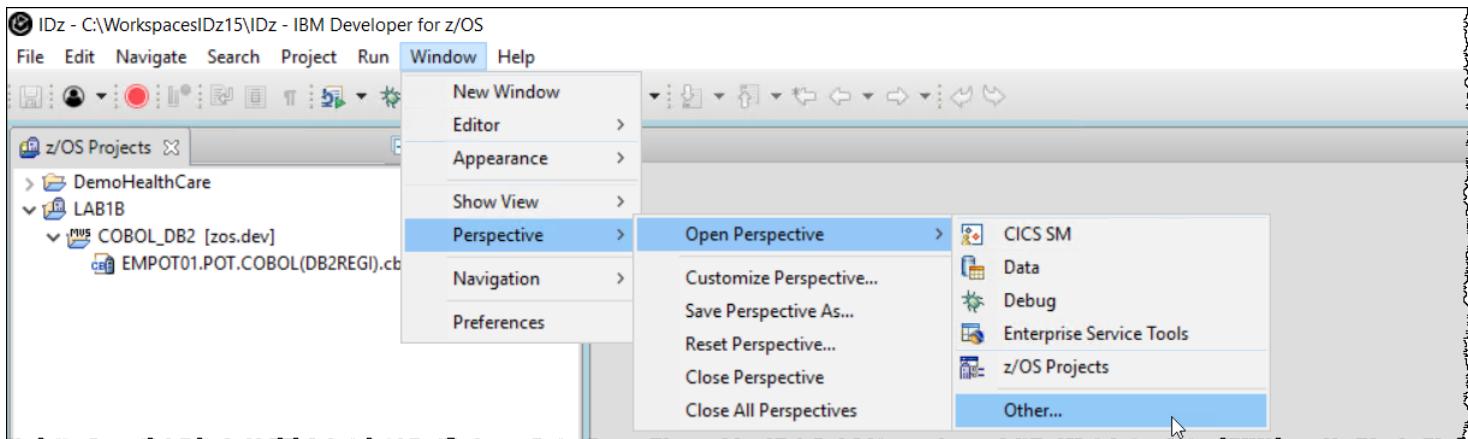
IBM Fault Analyzer for z/OS helps developers analyze and fix system and application failures for CICS, WebSphere MQ, IMS and DB2 environments. When an application ends abnormally, Fault Analyzer is automatically engaged, gathering real-time information about the event and its environment at the time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

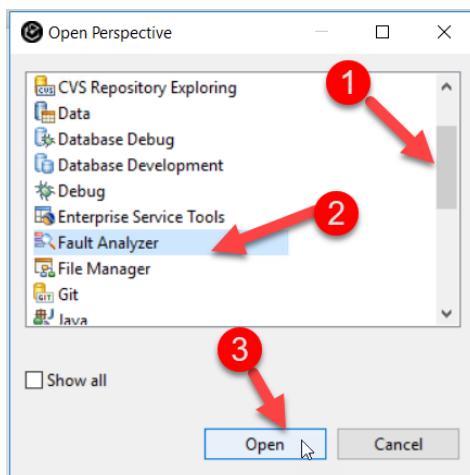


3.1 Using Fault Analyzer perspective

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

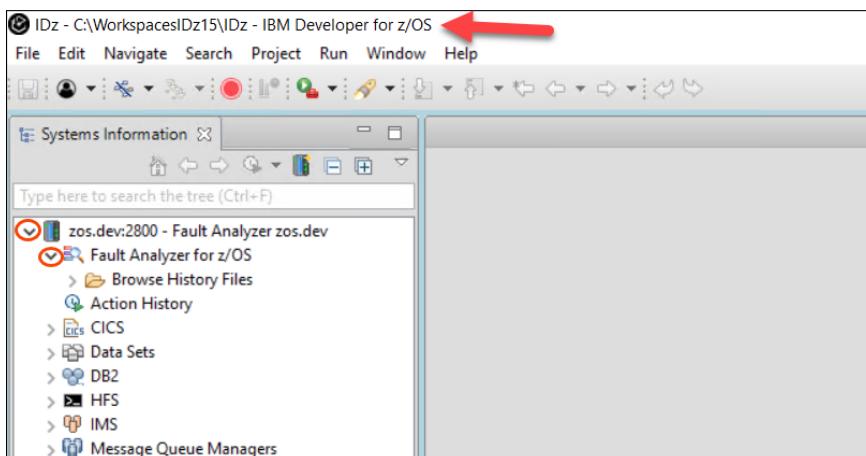


3.1.2 ► Scroll down, select **Fault Analyzer** and click **Open**

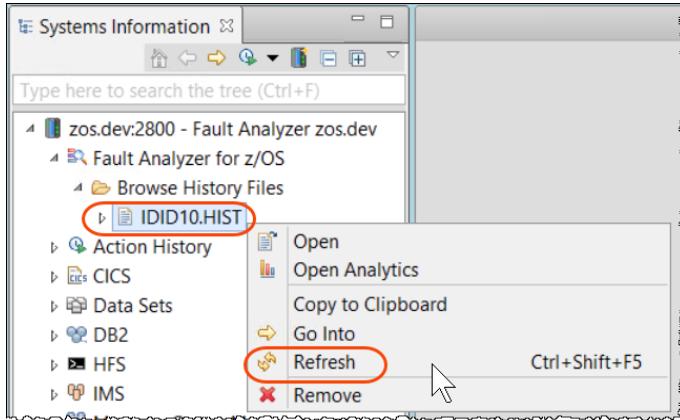


3.1.3 ► Under **Systems Information** view (on left) be sure that **Fault Analyzer for z/OS** is expanded

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...

A screenshot of the Systems Information interface. The left pane shows the same tree structure as before. In the right pane, under 'ZOS.DEV : 2800 : IDID10.HIST', there is a table 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 Fault Analyzer 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', and 'IDID10.HIST'. Below that is an 'Outline' panel with 'Prolog', 'Summary', 'Synopsis', 'Event summary', and 'Event details'. The main window displays a text-based report for fault F00353. The report header includes copyright information for IBM Fault Analyzer for z/OS V13R1M0 (UI47734 2017/06/01). It details a system abend (OCB) on job POT01RUN at SOW1 on 2019/12/18 08:45:42. A note states the report was saved after reanalysis. The report then focuses on module DB2REGI, program DB2REGI, source line 365, which had an abend (S0CB) due to a Decimal-Divide Exception. It also notes a system abend (OCB) occurred in module DB2REGI at offset X'F9C'. The report concludes with a message about a program-interruption code 000B (Decimal-Divide Exception) associated with the abend. At the bottom of the report window, there are tabs: Main Report, Event Details, Abend Information, System-Wide Information, and Miscellaneous. An arrow points from the 'Event details' link in the outline to the 'Event Details' tab. The 'Event Details' tab is currently selected. Below the report, a table lists events for fault F00353. The table has columns: FAULT_ID, JOB/TRAN, USER_ID, SYS/JOB, ABEND, L_ABEND, JOB_ID, JOBNAME, and USERNAME. The data for F00353 is: F00353, POT01RUN, EMPOT01, SOW1, S0CB, S0CB, JOB00212, POT01RUN, and empty for USERNAME. The data for F00352 is: F00352, HCMA, IBMUSER, CICSTS53, 4038, 4038, STC00044, CICSTS53, and empty for USERNAME.

FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB	ABEND	L_ABEND	JOB_ID	JOBNAME	USERNAME
> F00353	POT01RUN	EMPOT01	SOW1	S0CB	S0CB	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.

```

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

```

Main Report Event Details Abend Information System-Wide Information Miscellaneous

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
35 The COBOL source code for data fields 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.

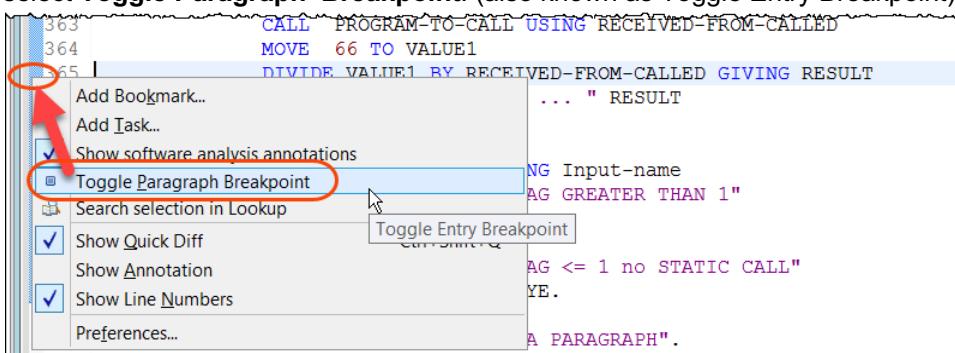
```

ZOS.DEV:2800/IDID10.HIST(F00327)-Report CBL DB2REGI.COB
-----+---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 I
            DISPLAY "The result is ... " RESULT
366    END-IF
367    IF BRANCHFLAG > 1
368      CALL 'REGIOC' USING Input-name
369      DISPLAY "BRANCHFLAG GREATER THAN 1"
370      PERFORM 530-SEYYA
371    ELSE
372      DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
373      PERFORM 540-GOODBYE.
374
375  530-SEYYA.
376  DISPLAY "EXECUTED SEYYA PARAGRAPH".

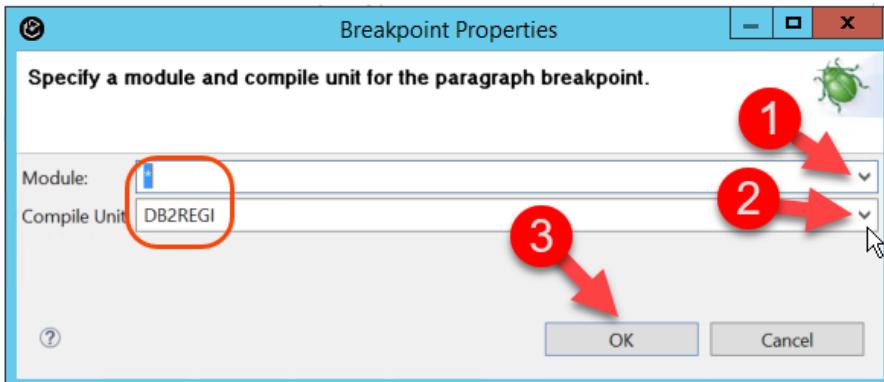
```

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)



- 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
-----+-----1-----2-----3-----4-----5-----6-----7-----8
      MOVE "LAB2" to WHICH-LAB.
  520-LOGIC.
    IF WHICH-LAB = 'LAB2'
      * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
      MOVE "REGI0B" TO PROGRAM-TO-CALL
      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
      MOVE 66 TO VALUE1
      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
      DISPLAY "The result is ... " RESULT
    END-IF
    IF BRANCHFLAG > 1
      CALL 'REGI0C' USING Input-name
      DISPLAY "BRANCHFLAG GREATER THAN 1"
      PERFORM 530-SEYYA
    ELSE
      DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
      PERFORM 540-GOODBYE.
  530-SEYYA.

```

- 3.1.13 ► Close all opened editors using **CTRL+ Shift + F4**.

Or just click on the  of each opened editor.

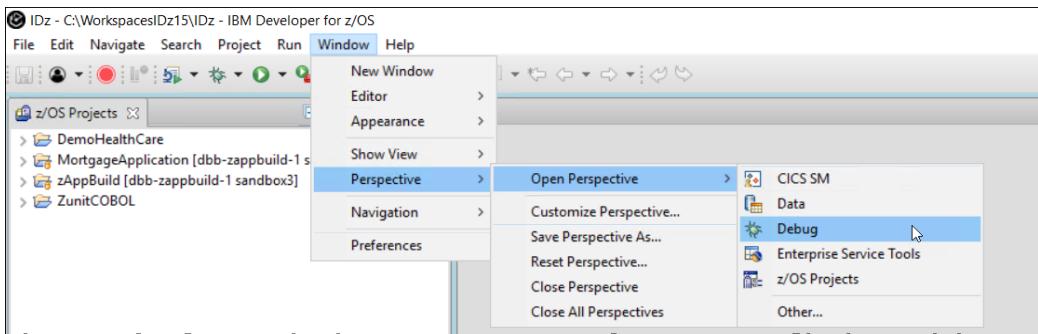
Section 4. Using the IBM z/OS Debugger for a temporary fix

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

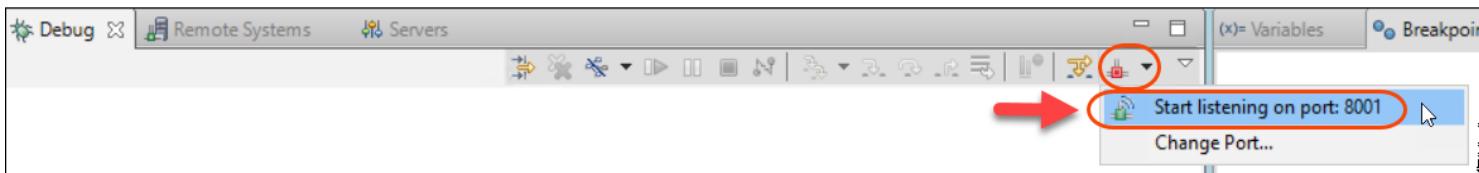
4.0 Be sure that IDz client is listening on port 8001

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**



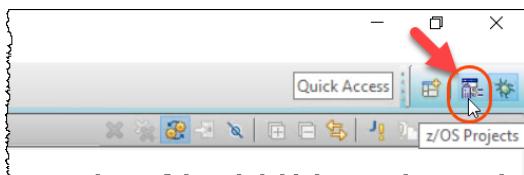
4.0.3 The listening icon will turn green and the IDz client is listening on port 8001.



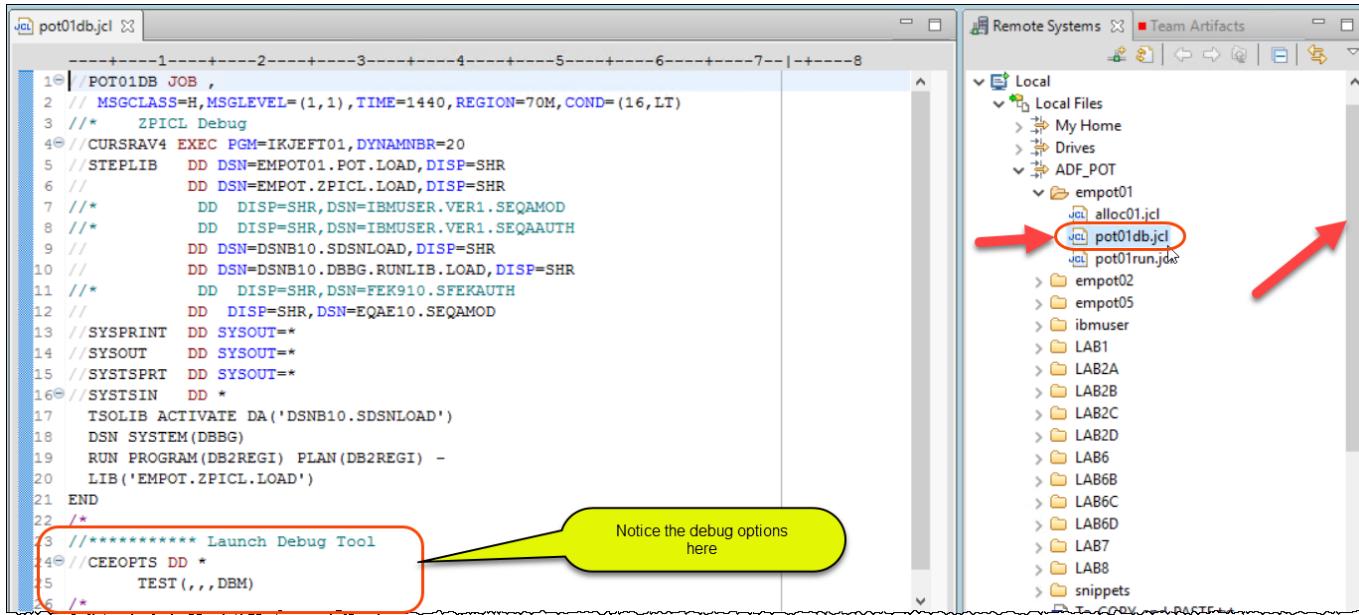
4.1 Submit the JCL to invoke the Debug

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

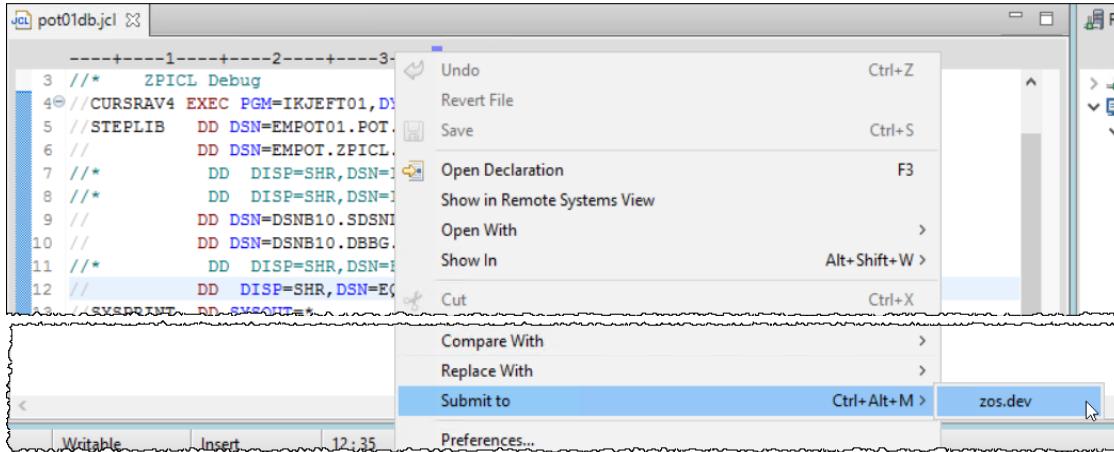
4.1.1 ► Using the top right corner Go to the **z/OS Projects** Perspective by clicking on the icon below.



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..



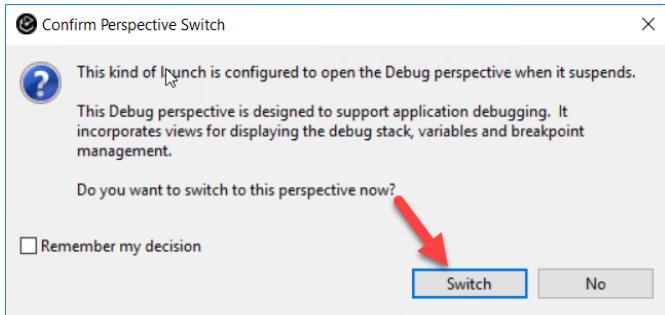
4.1.3 ► Right click on the JCL being edited and select **Submit to > zos.dev**



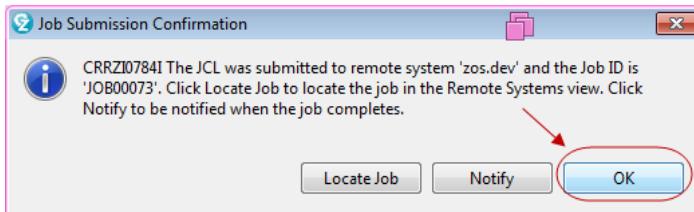
4.1.4 Once the job starts the z/OS debug will “talk” with you. Notice that the communication will be via the IDz connection (RSE), you don't need to specify IP addresses. This may work even when firewalls are in place.

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

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

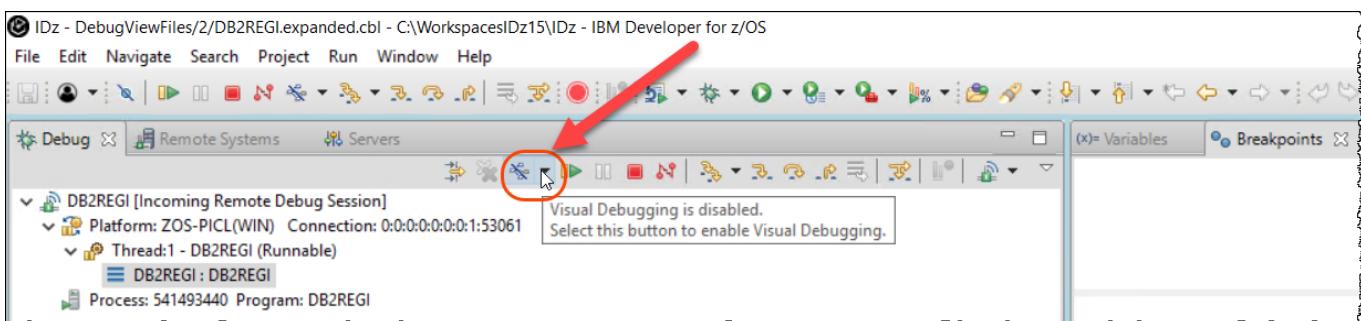


4.1.5 ➡ Also click **OK** for this dialog below



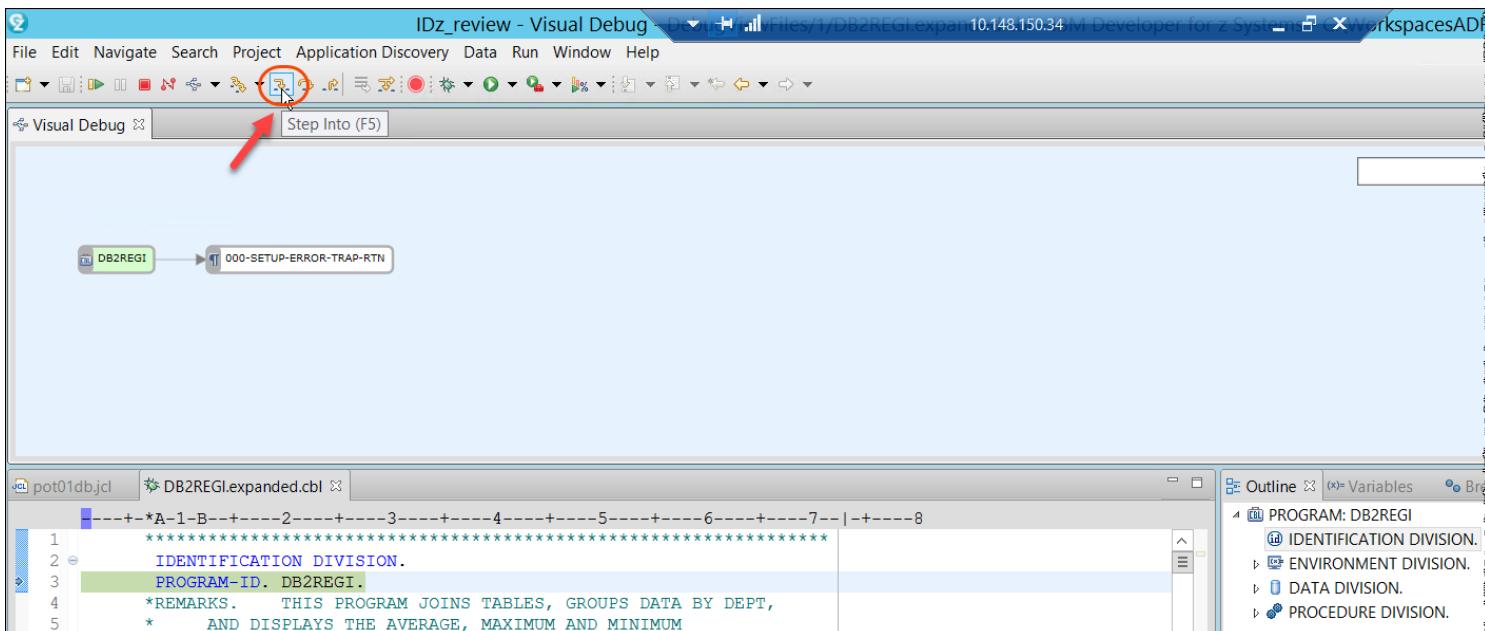
4.1.6 Using the *Debug* perspective:

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



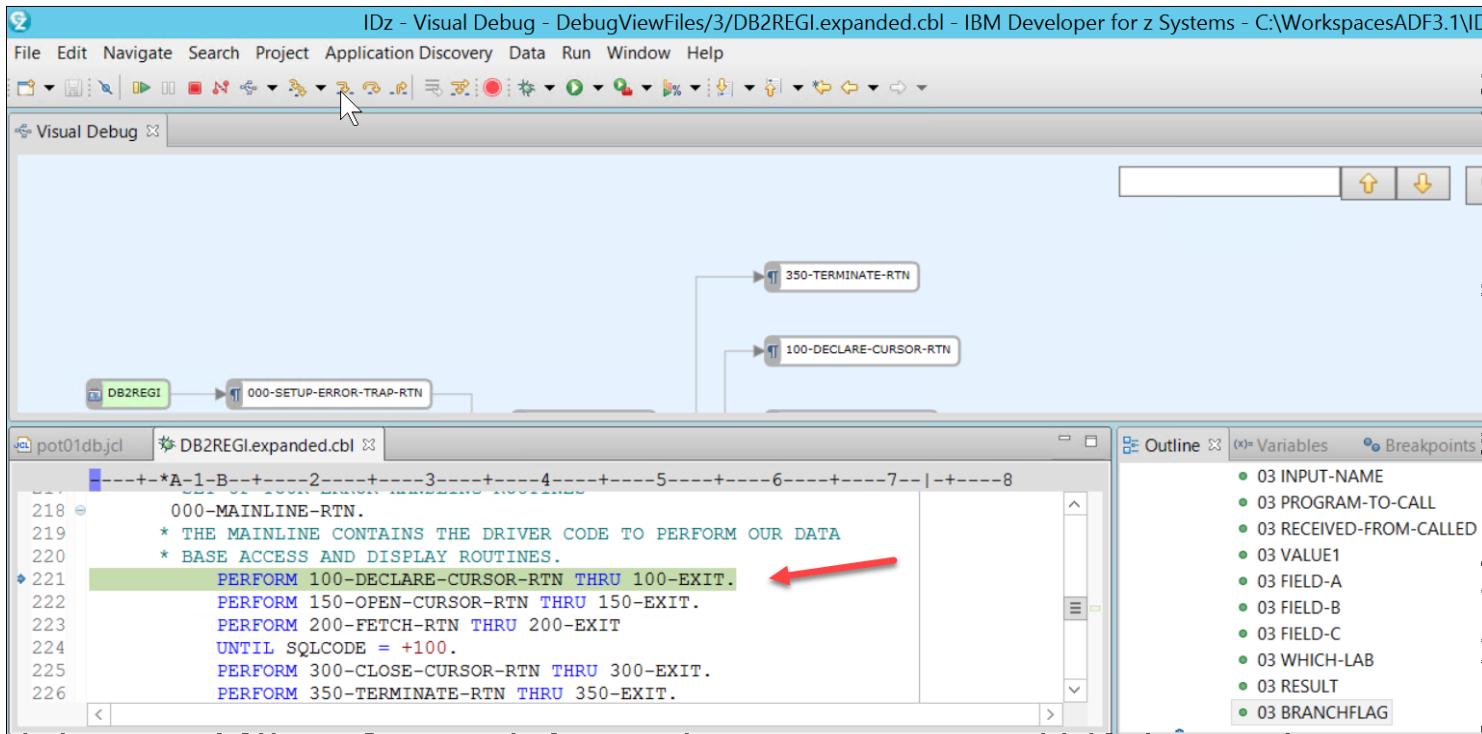
4.1.7 The *Visual Debug* view show the paragraphs being executed (in the top)

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



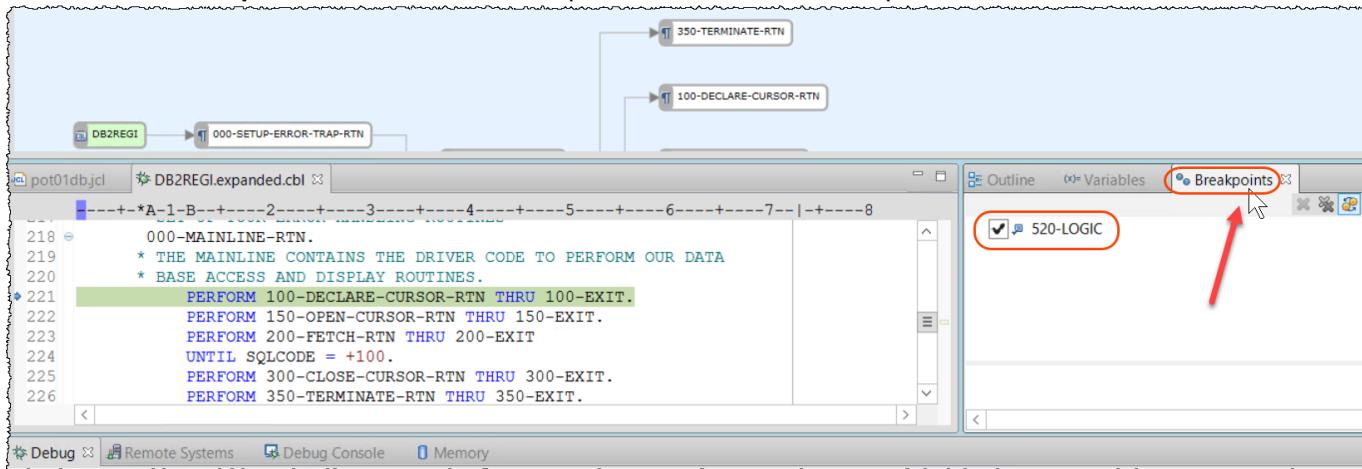
4.1.8 The execution will stop at the statement that will execute

```
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?

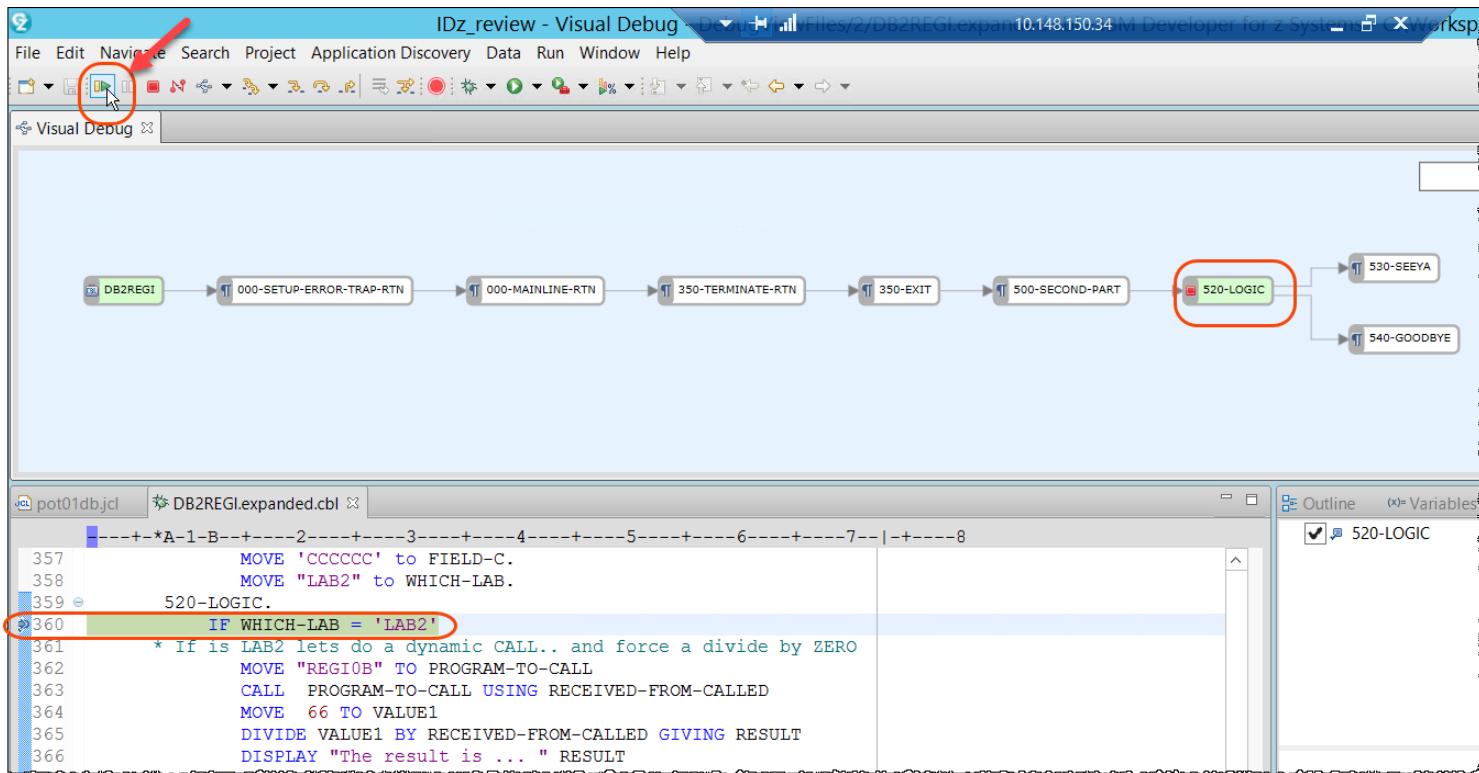


Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.

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.

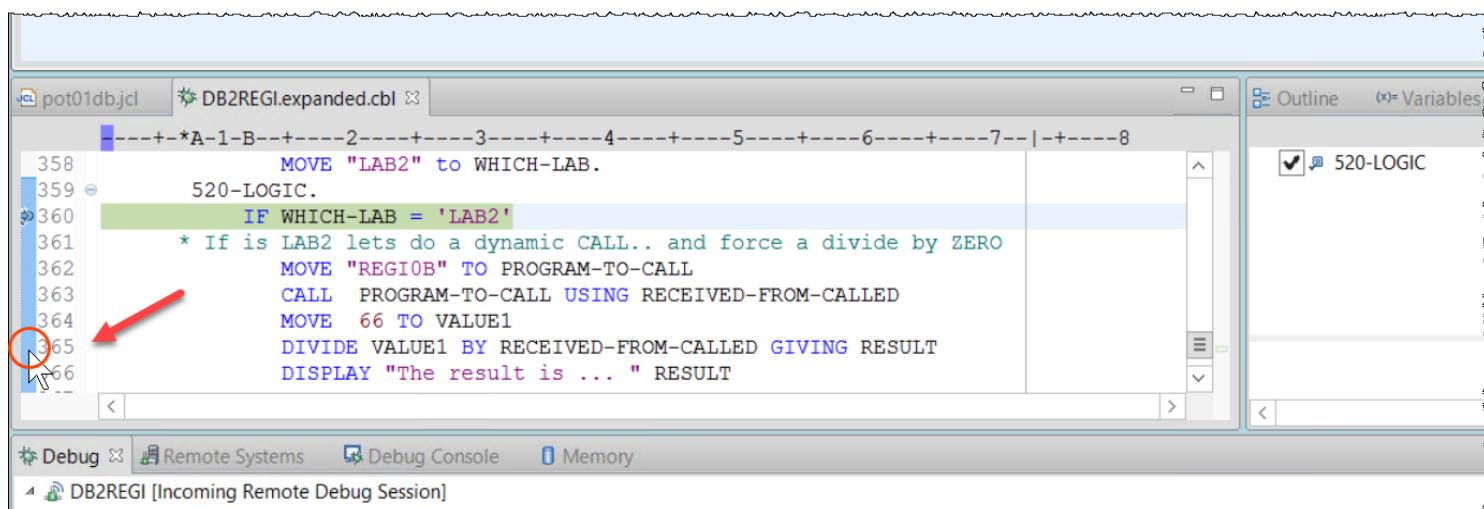
In PL/I, the stack trace represents the procedure call chain.

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.

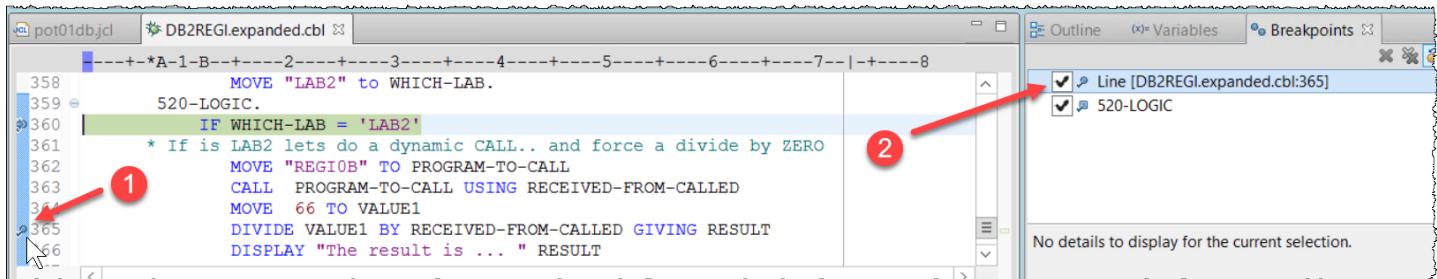


4.1.11. As you verified using Fault Analyzer, the abend occurred on line 365 where you had a divide by zero (see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.

► 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.

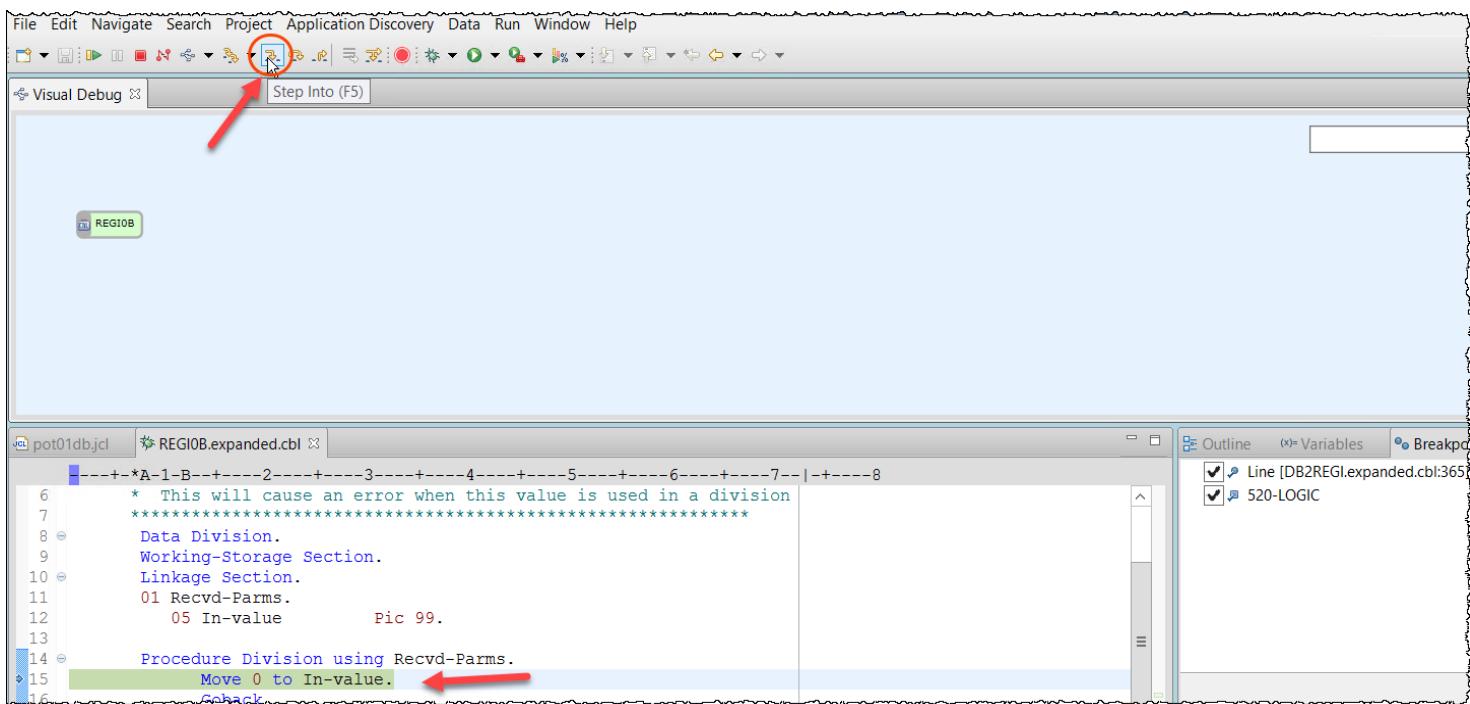


4.1.12 Notice that a small circle  is shown on the left of line 365 and also a breakpoint is displayed on the Breakpoint view



The screenshot shows the IBM Rational Application Developer interface. In the top right corner, there's a 'Breakpoints' tab. Below it, the 'Breakpoints' view displays two entries: 'Line [DB2REGI.expanded.cbl:365]' and '520-LOGIC', both with checkmarks. A red arrow points from the number '2' to the 'Breakpoints' view. In the bottom right corner of the interface, a message says 'No details to display for the current selection.' On the left side of the interface, the code editor shows a COBOL program. Line 365 is highlighted in green and contains the instruction 'DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT'. A red arrow points from the number '1' to the line 365 in the code editor.

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.



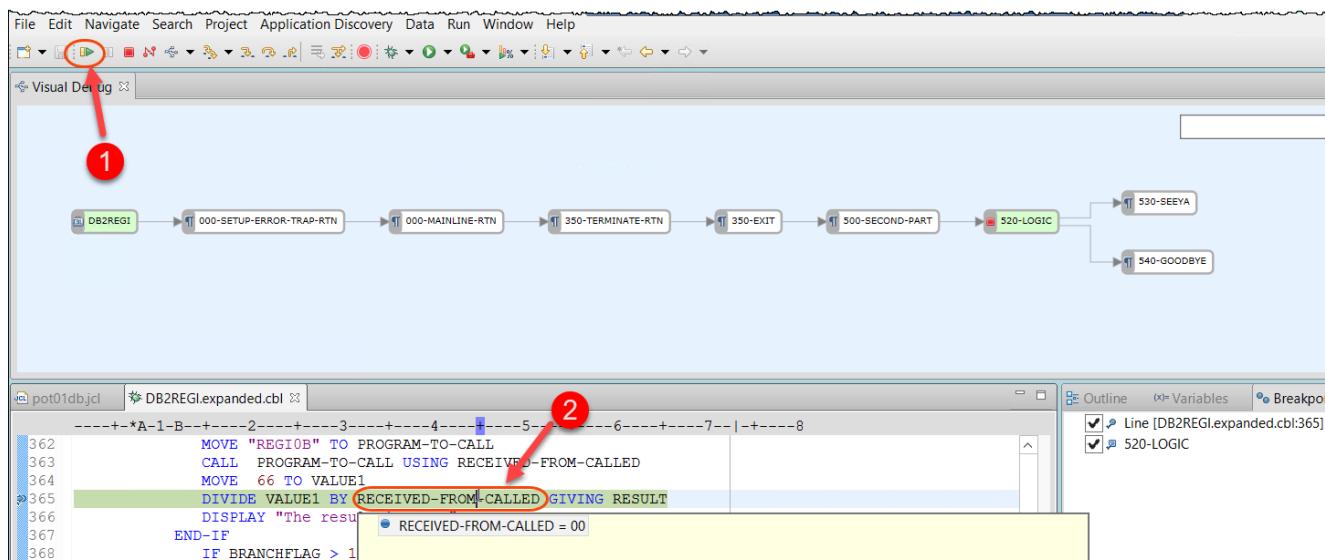
The screenshot shows the IBM Rational Application Developer interface. At the top, there's a toolbar with various icons. One icon, which looks like a magnifying glass over a line of code, is circled with a red arrow. Below the toolbar, the 'Visual Debug' tab is selected. In the center, there's a window titled 'REG10B' which is currently empty. At the bottom of the interface, the code editor shows a COBOL program. Line 15 is highlighted in green and contains the instruction 'Move 0 to In-value.'. A red arrow points from the line 15 in the code editor to the instruction 'Move 0 to In-value.'.

- 4.1.14 1 Click on or press F8.

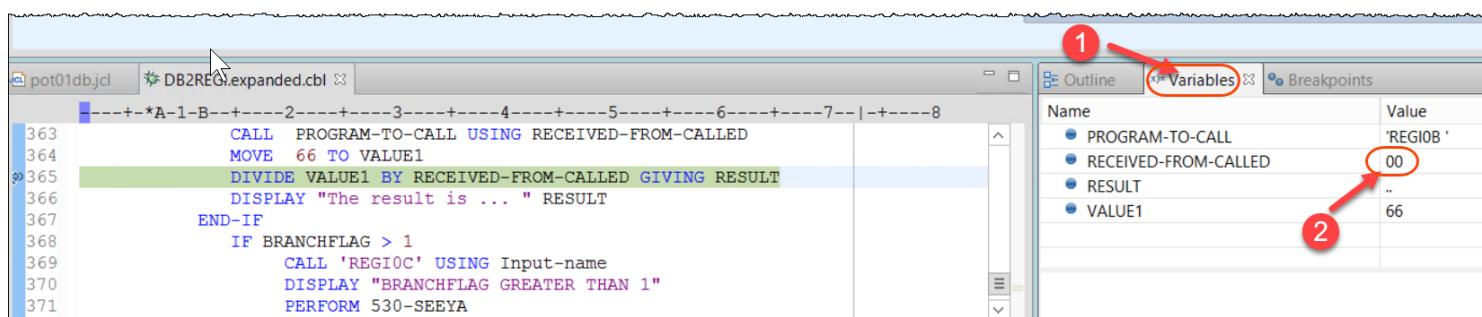
The execution will stop at the breakpoint that you created (line 365). .

- 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).

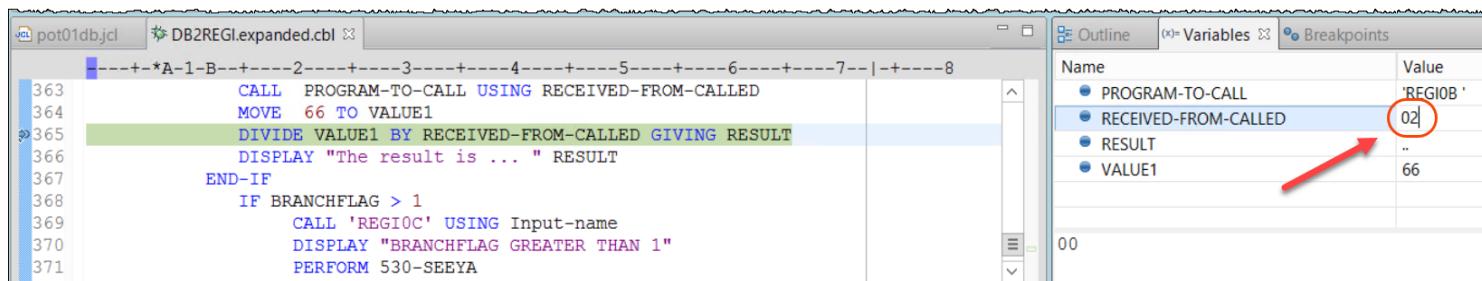


- 4.1.15 1 Click the Variables tab (right) and 2 verify that RECEIVED-FROM-CALLED is 00.

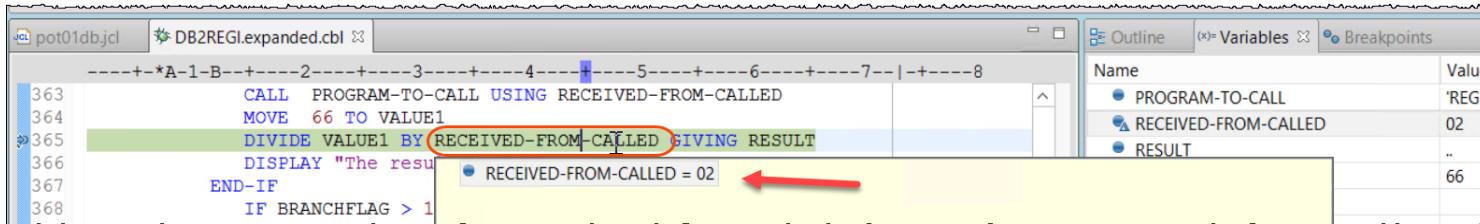


- 4.1.16 This is the abend cause. You must change the value to something different than 0.

Click on RECEIVED-FROM-CALLED value of 00 and modify to 2, just overtyping and press enter.



- 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).



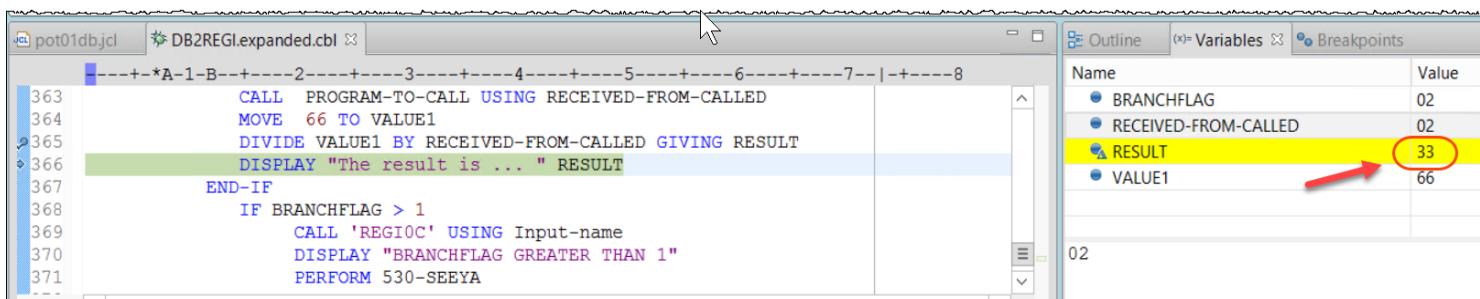
```

JCL pot01db.jcl 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

```

Name	Value
PROGRAM-TO-CALL	'REGI00B'
RECEIVED-FROM-CALLED	02
RESULT	..
BRANCHFLAG	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**.



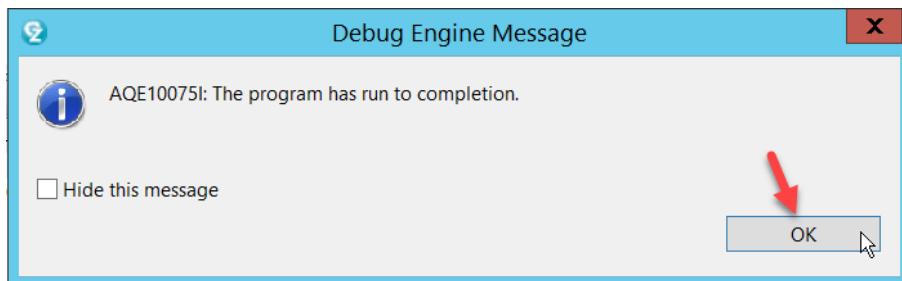
```

JCL pot01db.jcl 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-SEEEY

```

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



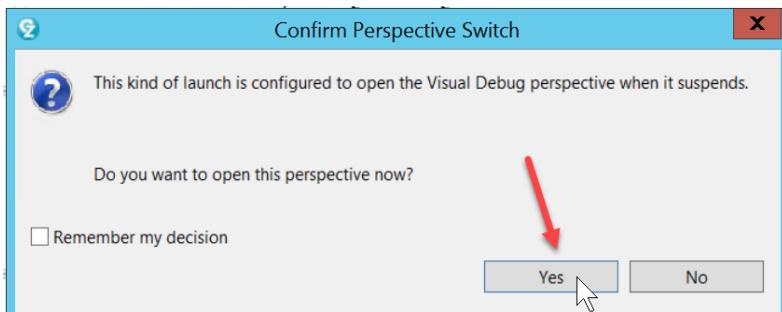
A stack pattern breakpoint is a special type of conditional breakpoint. With this feature, you can specify that you only want to stop at a location when the current stack trace contains a predefined stack pattern.

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.

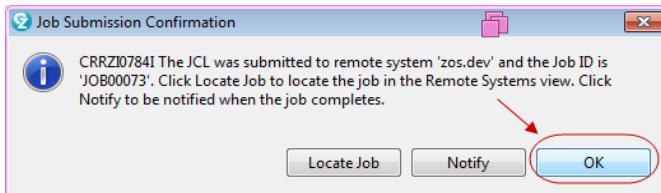
4.2.1 ► (Optional) Right click on the JCL being edited and select **Submit > zos.dev**

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).

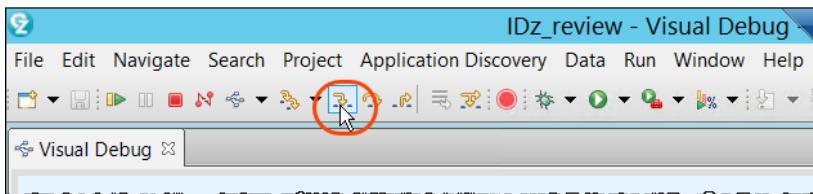


4.2.3 ► (Optional) Also click **OK** for this dialog below

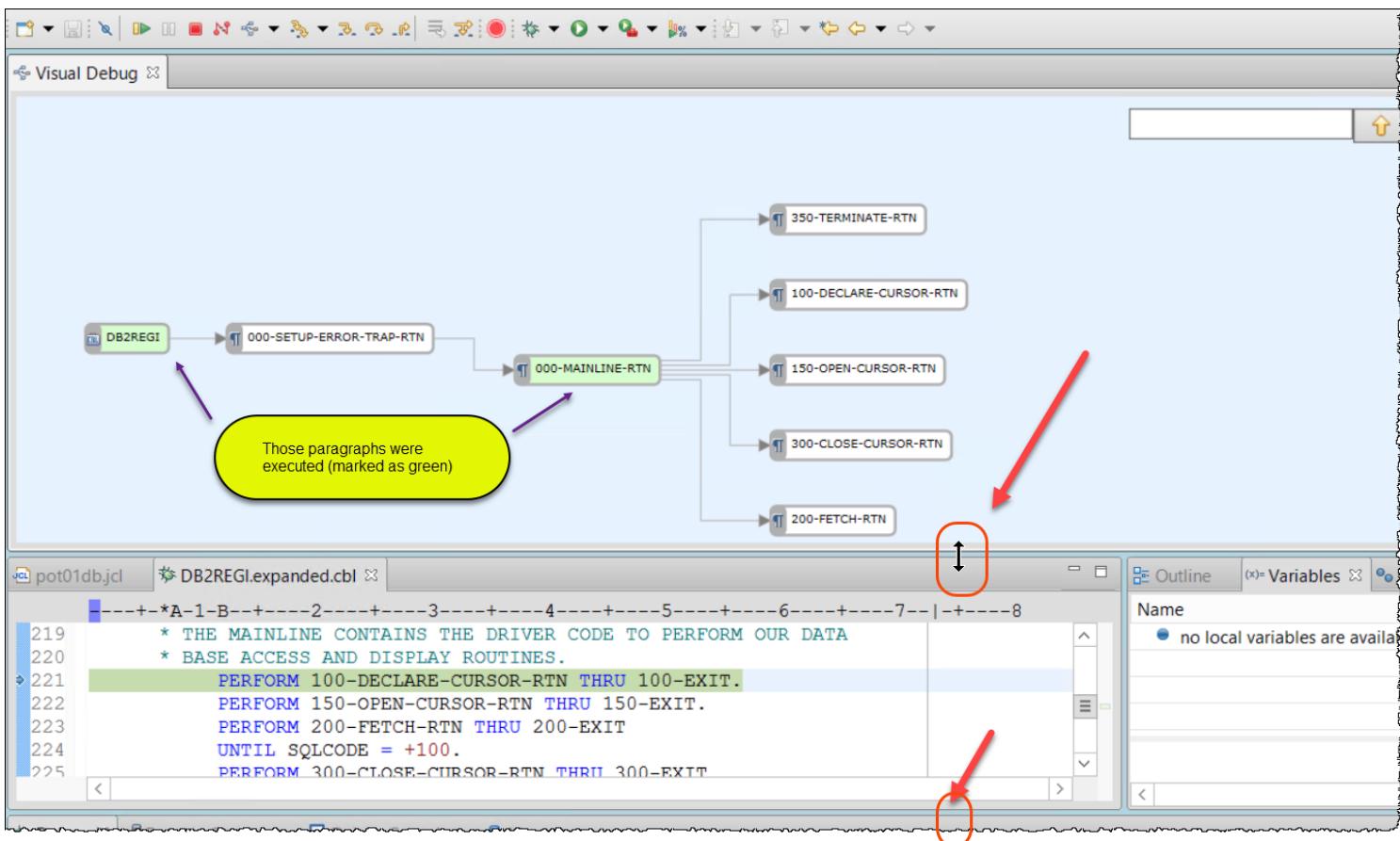


4.2.4 (Optional) Using the *Visual Debug* perspective:

► Click on icon (or Press **F5**)



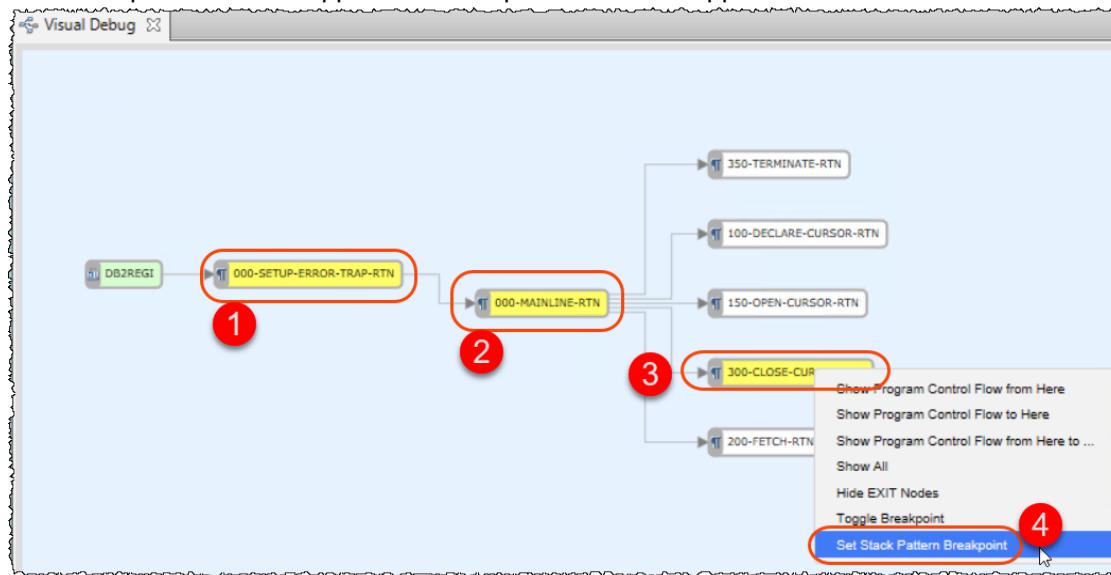
4.2.5 ► (Optional) Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green



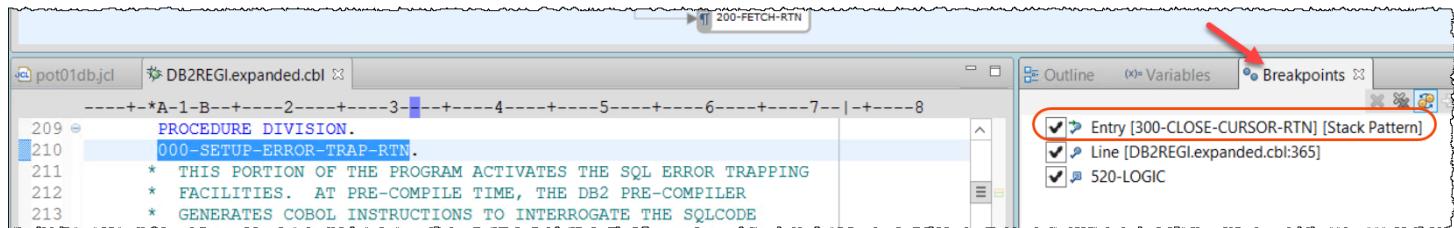
4.2.6 ► (Optional) Using the **Visual Debug** view, scroll down

► 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.

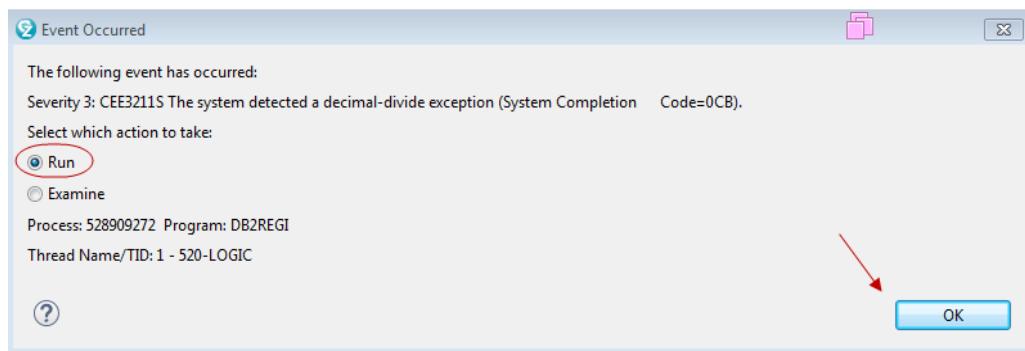


4.2.7 ► (Optional) Click on **Breakpoints** view and you will see this breakpoint created.



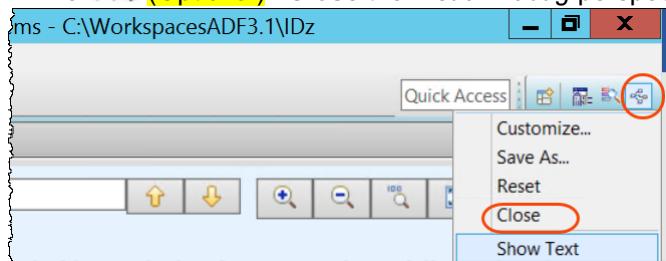
4.2.8 ► (Optional) Click on ► (or press **F8**) few times to resume the execution

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

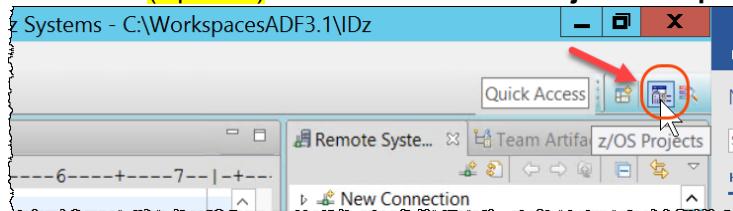


Notice – This breakpoint did not happen since the execution did not execute the 3 selected paragraphs.

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

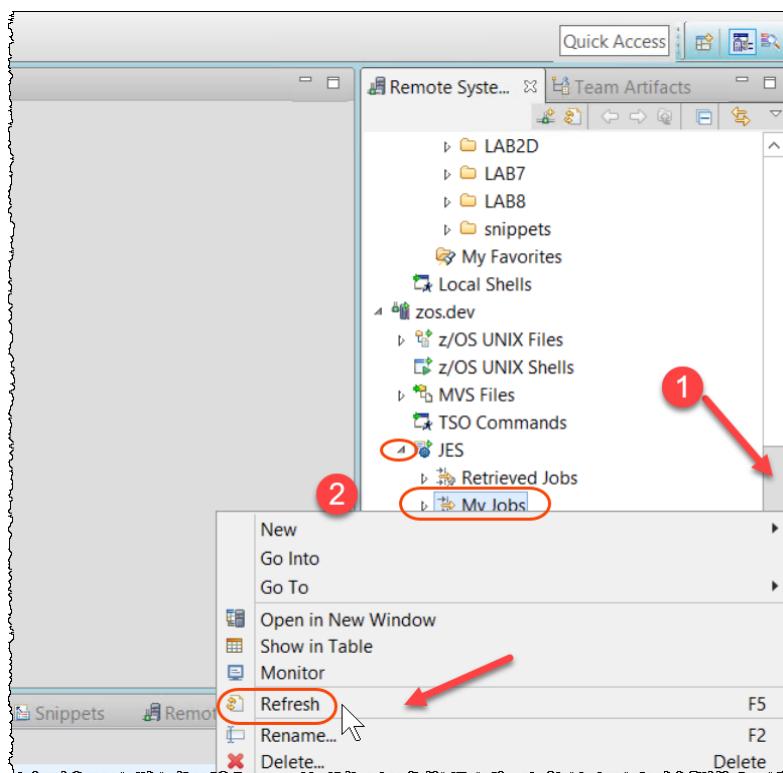
On ISPF many people use SDSF for this activity.

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

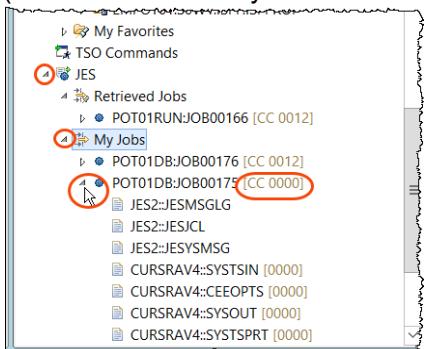
4.3.1 ► Use Ctrl + Shift + F4 to close all opened editors. Do not save any changes.

Or just click on the of each opened editor.

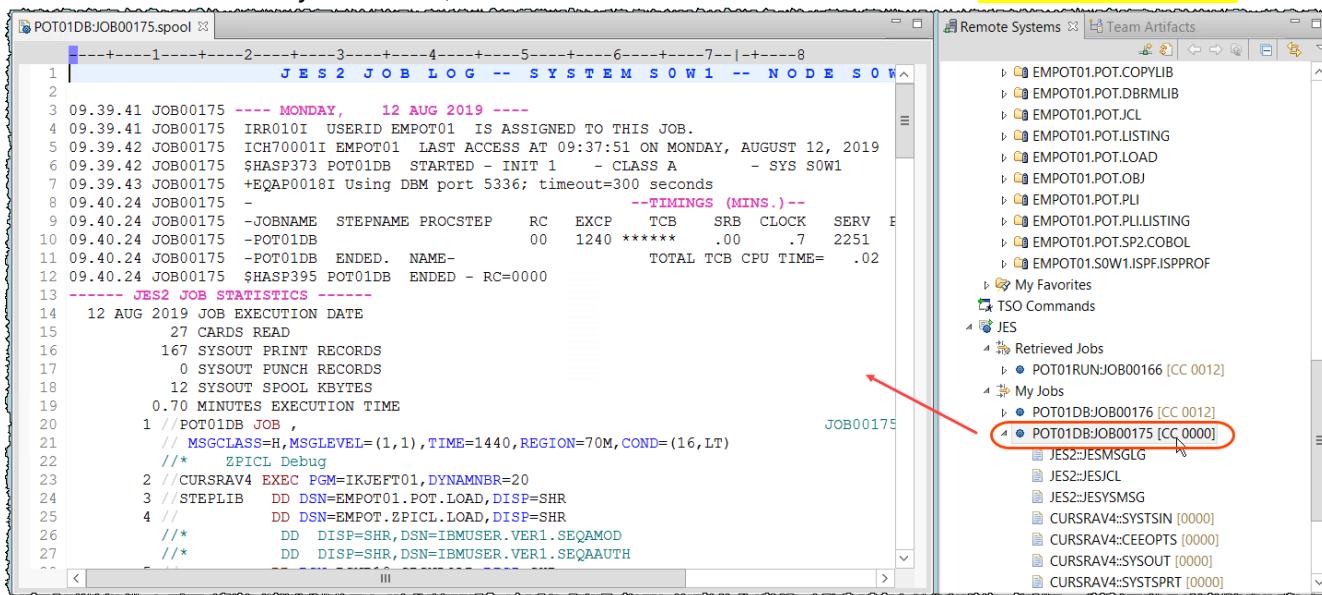
4.3.2 ► Using the Remote Systems view scroll down, expand JES right click on My Jobs and select Refresh



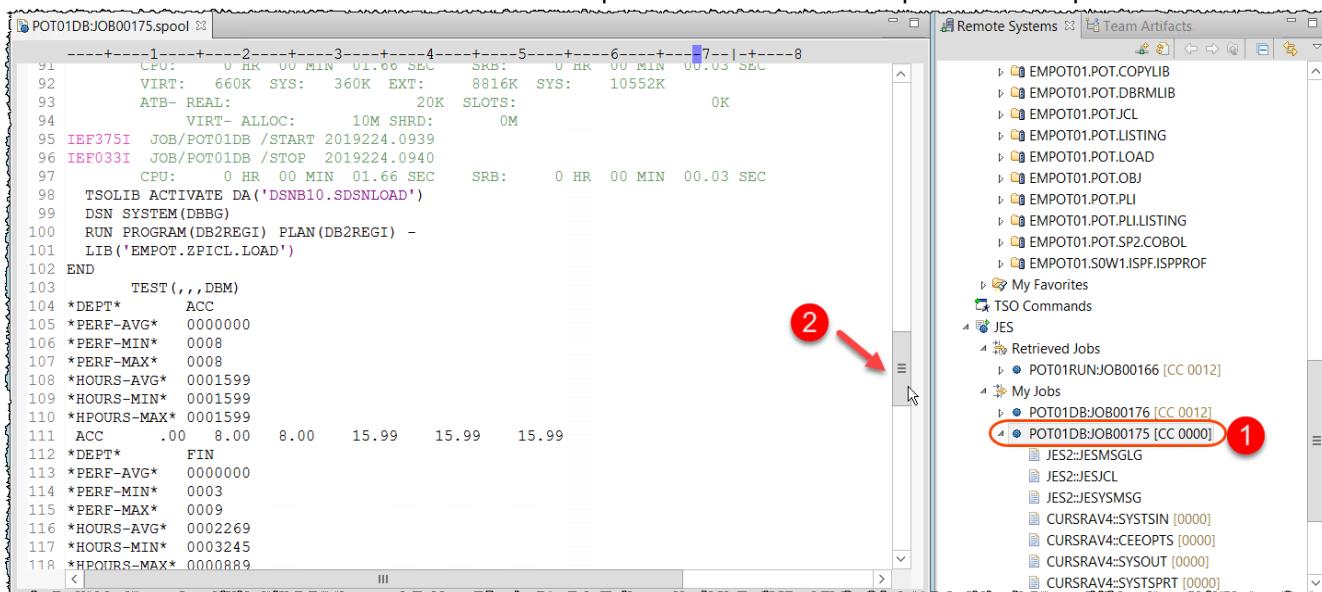
4.3.3 ► Expand My Jobs and the Job **POT01DB:JOB00xxx** that has the CC 0000 (it will be the latest if you did not make the optional step)



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



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



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

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      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

```

5.1 Using the Editor with z/OS COBOL programs

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

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

```

z/OS Projects
  LAB1B
    COBOL_DB2 [zos.dev]
      EMPOT01.POT.COBOL(DB2REGI).cbl

DB2REGI.cbl

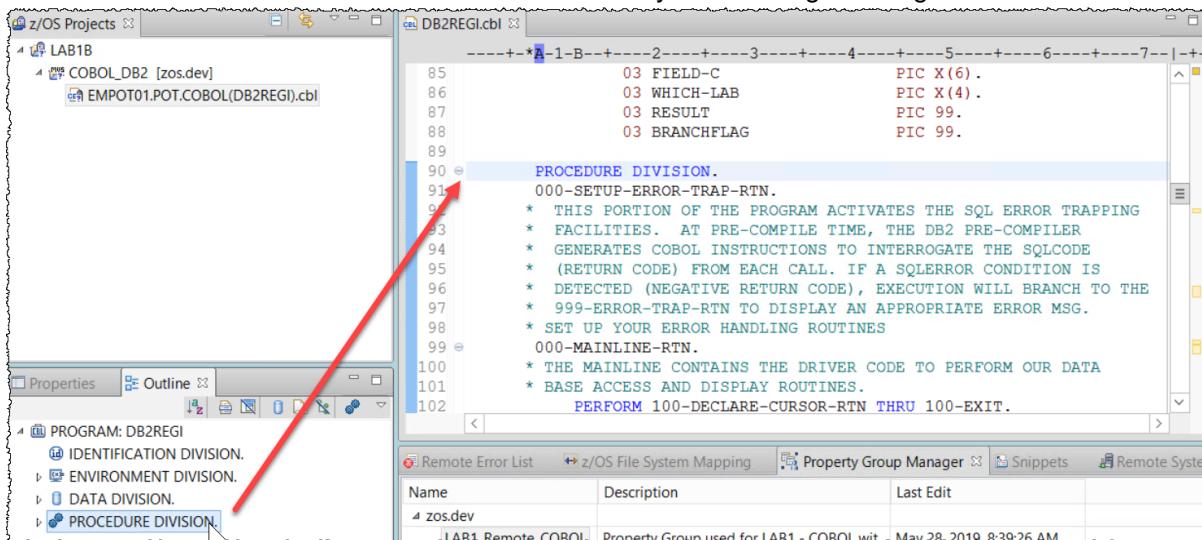
1 IDENTIFICATION DIVISION.
2   PROGRAM-ID. DB2REGI.
3   *REMARKS.  THIS PROGRAM JOINS TABLES, GROUPS DATA BY DEPT.
4   * AND DISPLAYS THE AVERAGE, MAXIMUM AND MINIMUM
5   * HOURS, AND PERFORMANCE EVALUATION BY DEPT.
6   * Modified by Regi to add DEAD CODE - Jan/2-14
7   * Modified by Regi to added test 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.

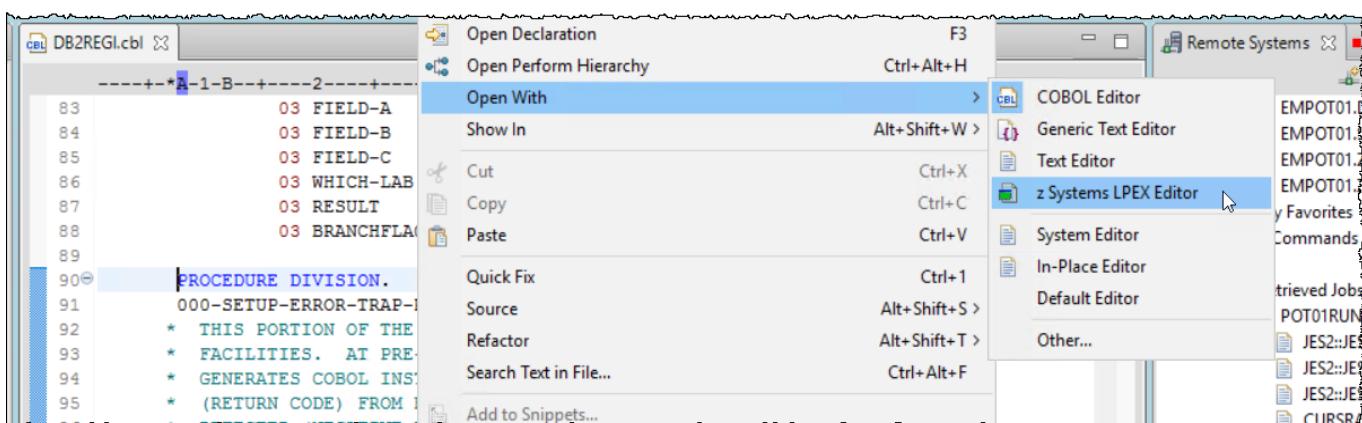
► Using the editor, browse the program and notice that the contents of the *Outline* view are synchronized with the COBOL source code and vice versa.

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



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

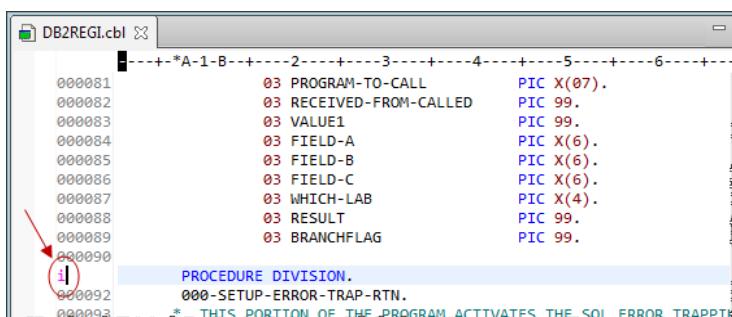
► To switch the editor, right click on the program and select **Open with > z Systems LPEX Editor**



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.**



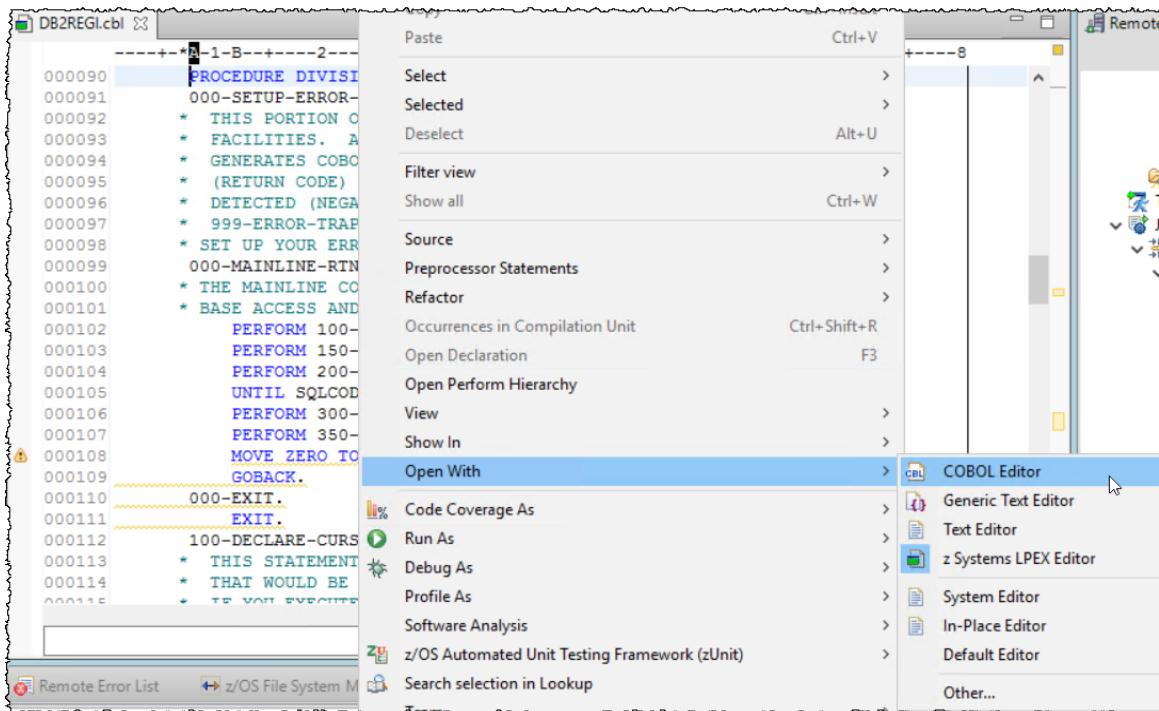
5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)

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

Click **No** for saving changes.

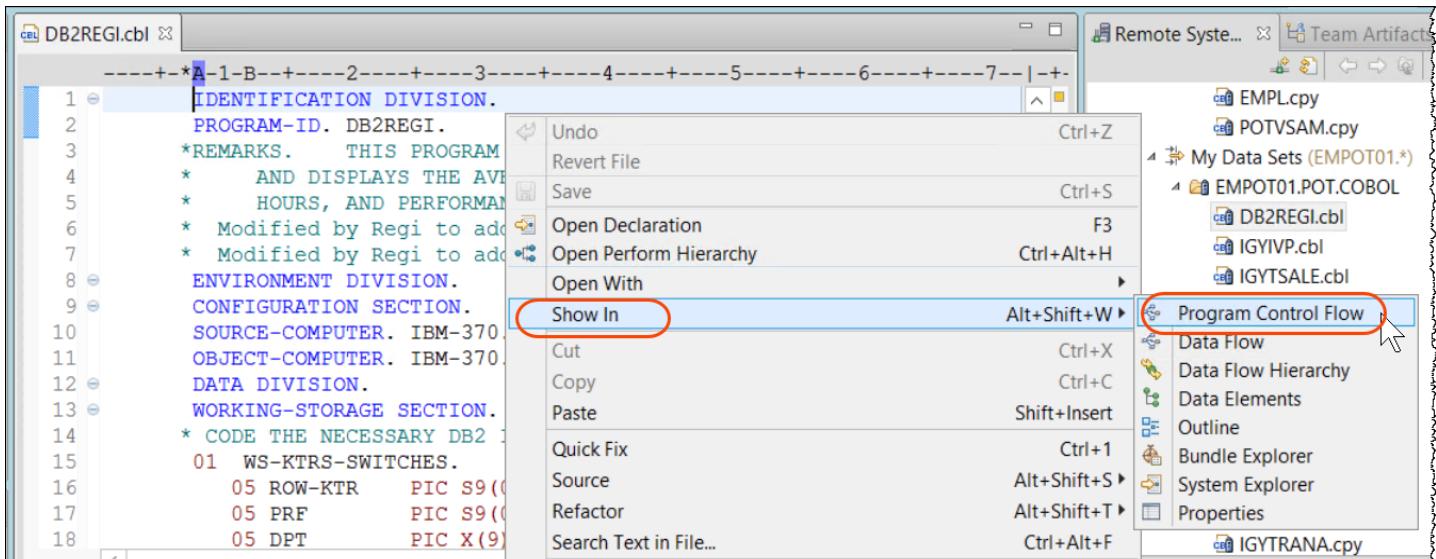
This editor is more like the Java Editor and gives you more capabilities. When you see all advantages, probably you will prefer this editor than the ISPF like editor (LPEX)..

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



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

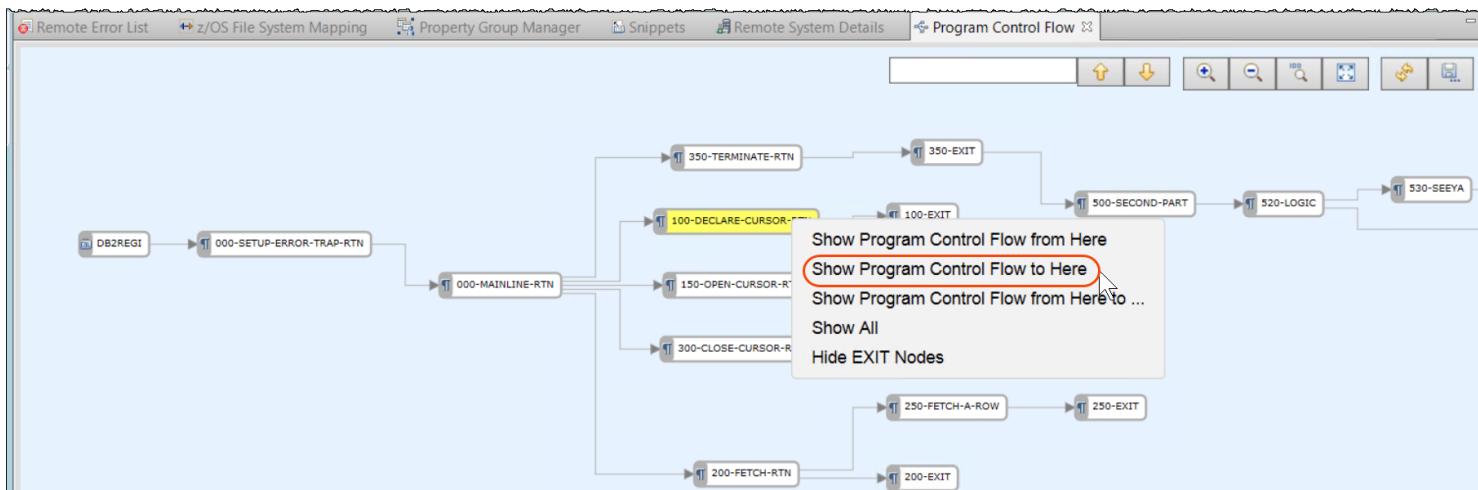
► Right click on the editor and select **Show In → Program Control Flow**



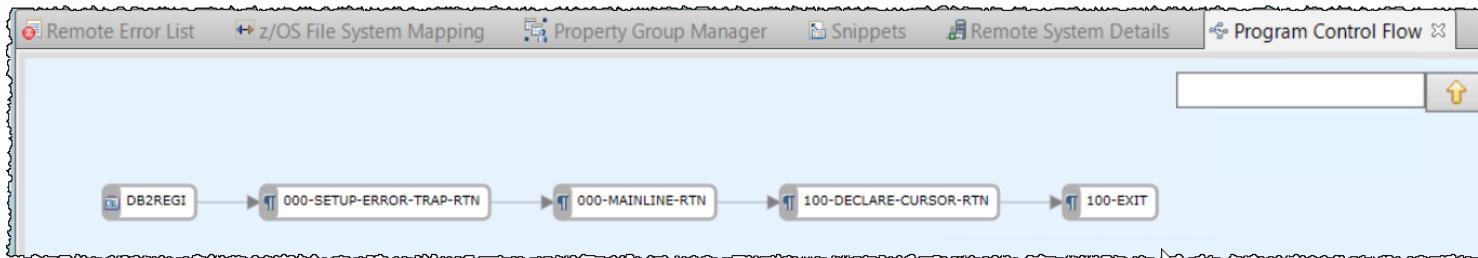
5.1.7 See the results under *Program Control Flow* view.

► 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**.



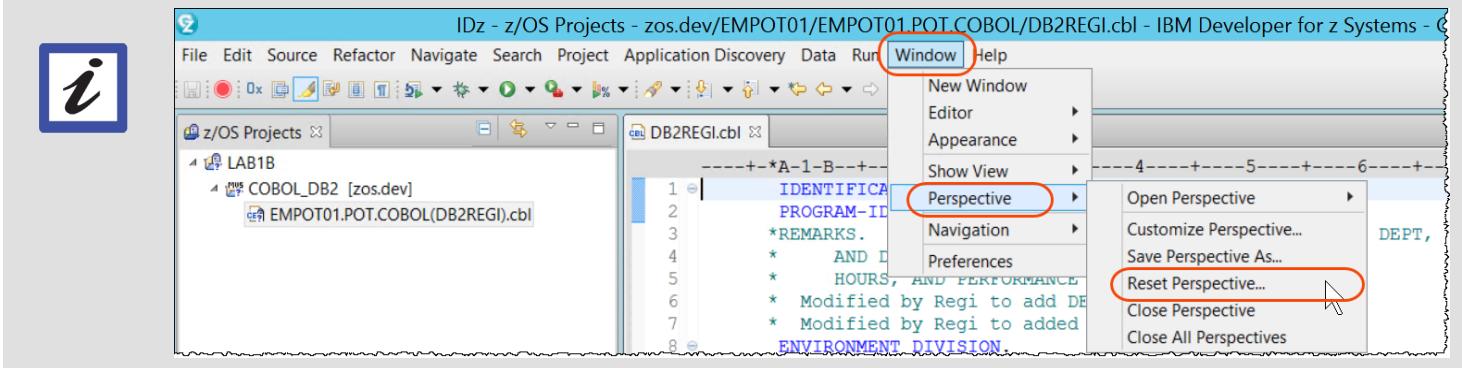
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



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

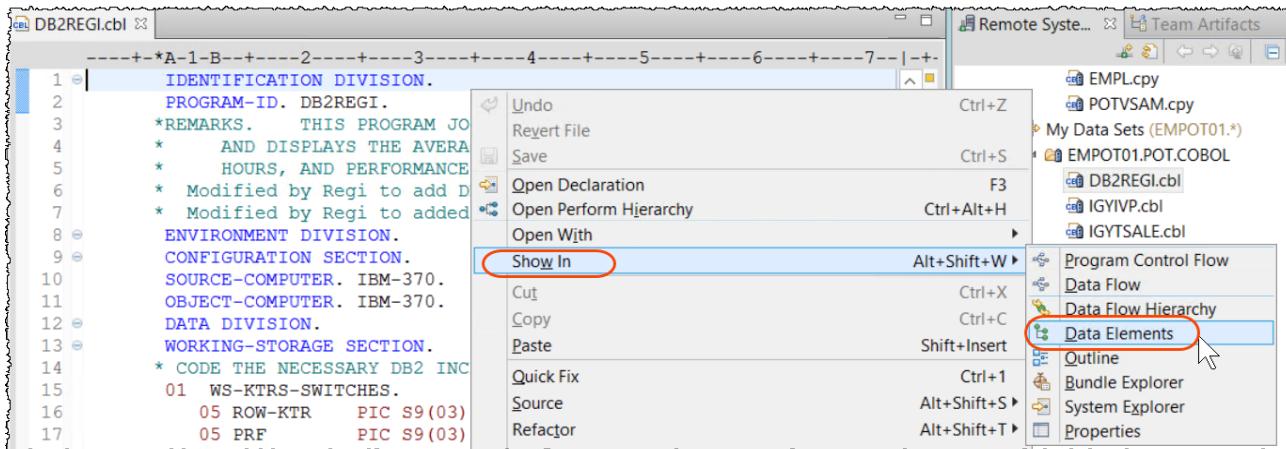
At any time, you may restore the perspective to the default. You may need that when you did mistakes and closed views you should not.

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

Showing data elements from DB2REGI.cbl								Type search text to filter by Name:
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			Open Declaration					
ROW-KTR			Occurrences in Compilation Unit					
ROW-MSG			Formatted Editor					
ROW-STAT			Search selection in Lookup					

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.

```

-----+--A-1-B-----2-----3-----4-----5-----6-----7-----8
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-SEEYA
253      ELSE
254        DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
255        PERFORM 540-GOODBYE.
256      530-SEEYA.
257      DISPLAY "EXECUTED SEEYA PARAGRAPH".
258      540-GOODBYE.

```

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

4 DB2REGI.cbl (3 matches)

8/103 RECEIVED-FROM-CALLED PIC 99.

244: 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 **REG10B** that is returning 0 on this variable as seen before with Fault Analyzer.

Exploring Data Flow

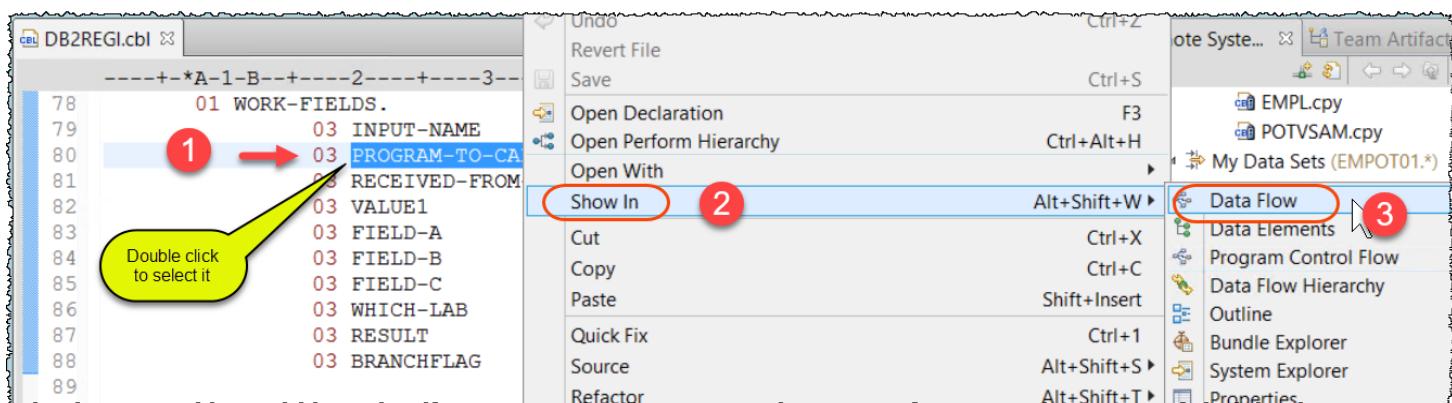
This capability is cool. *Program data flow* provides a graphical and hierarchical view of the data flow within a COBOL program. You can use this feature to examine how a data element is populated, modified, or written elsewhere.

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

The screenshot shows the Rational Developer for System z interface. The main window displays a COBOL source code editor for the unit DB2REGI.cbl. The code includes sections for WORK-FIELDS and PROCEDURE DIVISION, with several comments explaining error trapping and mainline routines. A red arrow points from the bottom of the editor area up towards the line '03 RECEIVED-FROM-CALLED'. The bottom navigation bar includes tabs for Remote Error, z/OS File Syst., Property Grou..., Snippets, Remote Syste..., Console, Remote Cons..., and Program Cont... . The status bar at the bottom shows the message 'RECEIVED-FROM-CALLED - 3 matches in compilation unit of 'DB2REGI.cbl''. The bottom-most tab is highlighted with a red circle around its label 'DB2REGI.cbl (3 matches)'. Below this tab, three specific matches are listed:

- 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).



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 IBM Rational Developer for z/OS interface. At the top, there's a code editor window titled "DB2REGI.cbl" containing COBOL code. Line 243 contains the statement "MOVE "REGI0B" TO PROGRAM-TO-CALL". Below the code editor is a graphical debugger pane. In the debugger, there's a variable node labeled "REGI0B" with a yellow icon. A red arrow points from the text editor's line 243 to this variable node in the debugger. The debugger pane also shows other nodes like "01 WORK-FIELDS". At the bottom of the interface, there are several toolbars and a status bar.

```

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
249      END-IF
250      IF BRANCHFLAG > 1
251          CALL 'REGI0C' USING Input-name
          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.

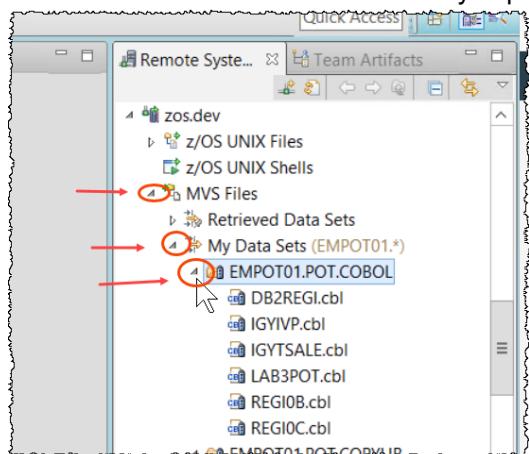
5.2 Fixing the program REGI0B

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

```

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

```

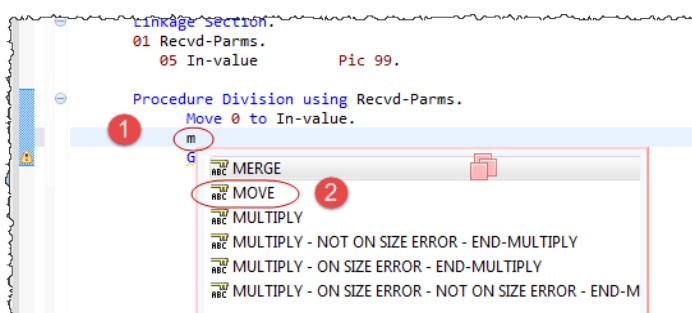
1 * Identification Division.
2   Program-ID. "REGI0B".
3   **** This program is called.
4   * It returns a value (0)
5   * This will cause an error when this value is used in a division
6   ****
7   Data Division.
8   Working-Storage Section.
9   Linkage Section.
10  01 Recvd-Parms.
11    05 In-value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

```

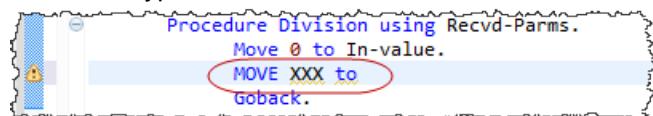
5.2.6 You can now take advantage of the **editor content assist...**

► 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)



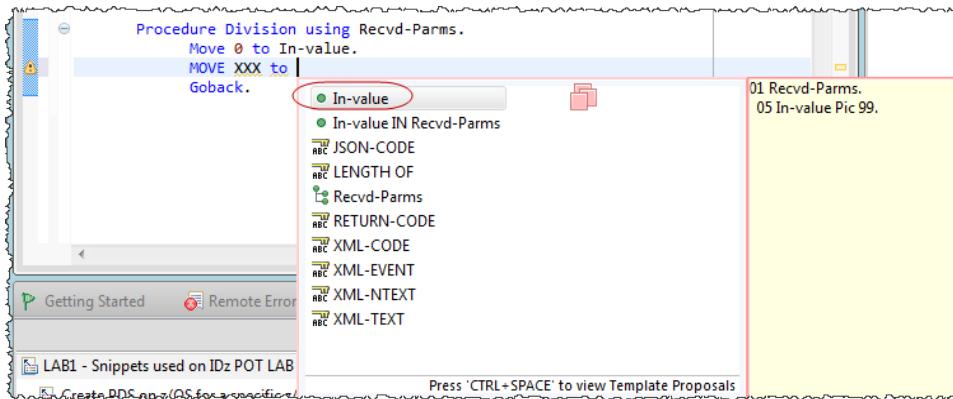
5.2.8 ► Type “XXX to”



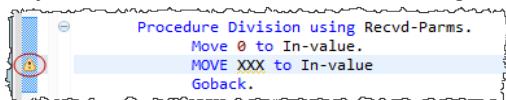
Procedure Division using Recvd-Parms.
Move 0 to In-value.
MOVE XXX to
Goback.

The screenshot shows a COBOL editor window. At the top left is a yellow warning icon with a red exclamation mark. The code is displayed in a monospaced font. The line 'MOVE XXX to' is circled in red, and the entire line 'MOVE XXX to Goback.' is underlined in red. The background of the editor has a light blue gradient.

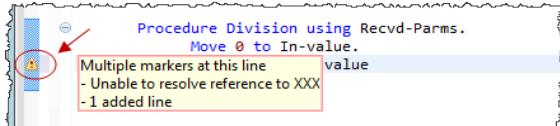
5.2.9 ►| Press **Ctrl + Space** to list the variables and select **In-value**.
 Notice that variables could be defined in copybooks.



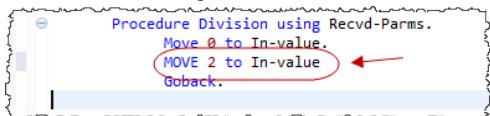
A yellow mark shows that something is wrong.



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.



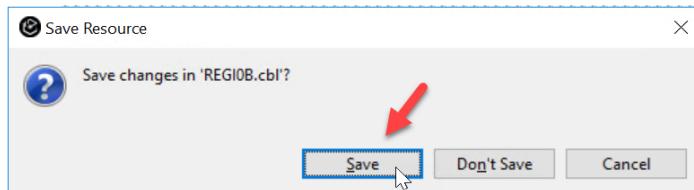
5.2.11 ►| Change from **XXX** to **2** as below. Note that the icon will go away.



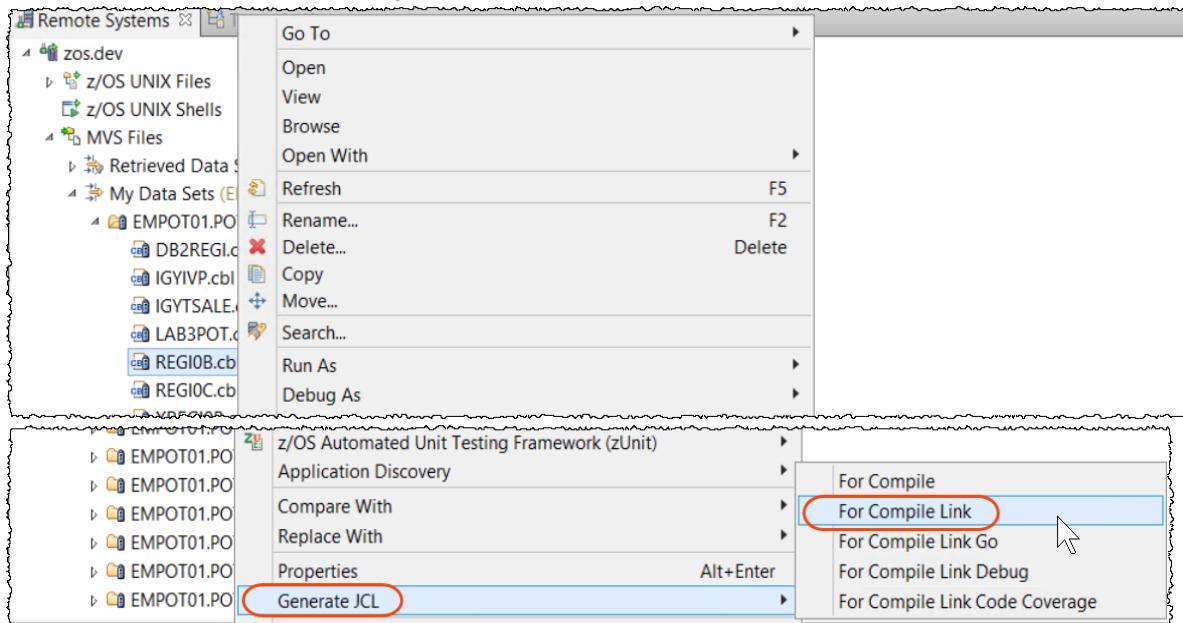
5.3 Compile and link REGI0B.

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.



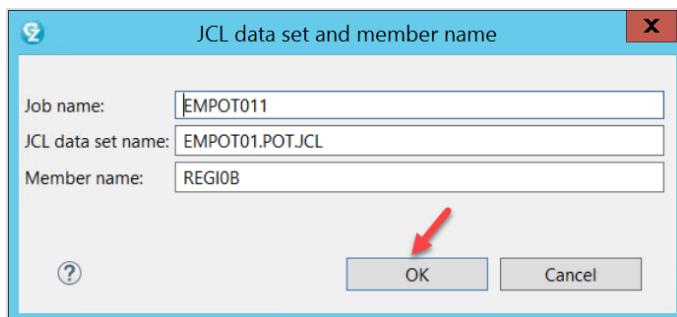
5.3.2 ► Using the Remote Systems view right click on **REGI0B.cbl** and select **Generate JCL > For Compile Link**.



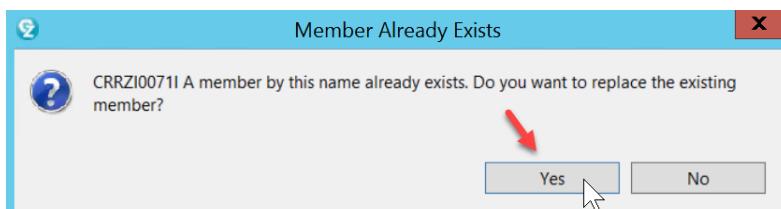
5.3.3 ► Accept the default member name and click **OK**



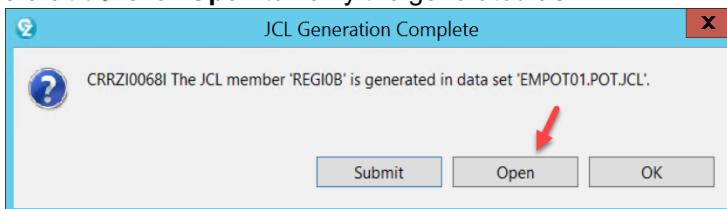
5.3.4 ► Accept the default and click **OK**



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



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



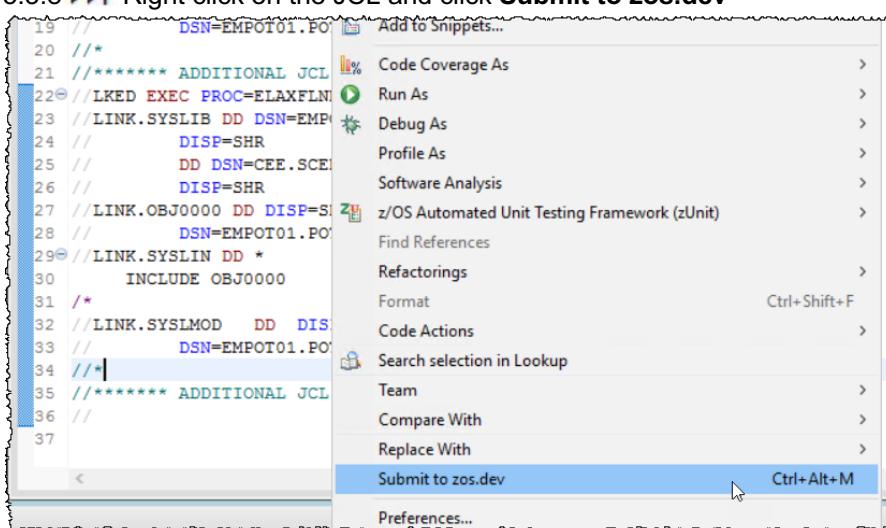
5.3.7 ➡ Scroll down and notice that the *Load module* will be created on the PDS: **EMPOT01.POT.LOAD**.

```

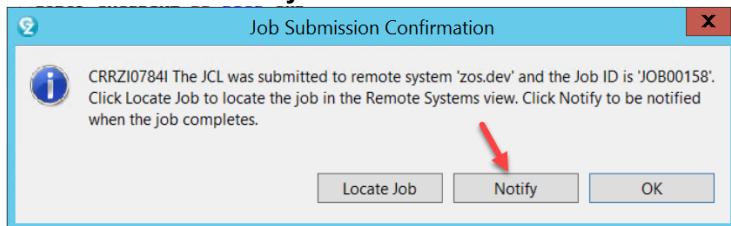
14 //      DD DSN=SHR,
15 //      DSN=EMPOT.ZPICL.COPYLIB
16 //      DD DUMMY
17 //      COBOL.SYSXMLSD DD DUMMY,
18 //      COBOL.SYSIN DD DISP=SHR,
19 //      DSN=EMPOT01.POT.COBOL(REGI0B)
20 //*****
21 //LKED EXEC PROC=ELAXFLNK
22 //LINK.SYSLIB DD DSN=EMPOT01.POT.OBJ,
23 //      DISP=SHR
24 //      DD DSN=CEE.SCEELKD,
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) 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,
23      DISP=SHR

```

[5/28/19, 10:34 AM] Job **JOB00158** is being submitted
[5/28/19, 10:34 AM] Job **EMPOT011: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**

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..

```

JCL pot01run.jcl
-----+-----+-----+-----+-----+-----+-----+
1 //POT01RUN JOB ,                                |-----8
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT) |
3 //* ZPICL Debug
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 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
8 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAUTH
9 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
10 //         DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
11 //         DD DISP=SHR,DSN=FEK910.SFEKAUTH
12 //SYSPRINT DD SYSOUT=*
13 //SYSOUT   DD SYSOUT=*
14 //SYSTSPRT DD SYSOUT=*
15 //SYSTSIN  DD *
16 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
17 DSN SYSTEM(DBDBG)
18 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
19 LIB('EMPOT.ZPICL.LOAD')
20 END
21 /*
22 //

```

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

► Expand CURSRAV4 EXEC PGM= and click on STEPLIB

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

```

z/OS Projects
  LAB1B
    COBOL_DB2 [zos.dev]
      EMPOT01.POT.COBOL(DB2REGI).cbt

JCL pot01run.jcl
-----+-----+-----+-----+-----+-----+-----+
1 //POT01RUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 //* ZPICL Debug
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 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
8 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAUTH
9 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
10 //         DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
11 //         DD DISP=SHR,DSN=FEK910.SFEKAUTH
12 //SYSPRINT DD SYSOUT=*
13 //SYSOUT   DD SYSOUT=*
14 //SYSTSPRT DD SYSOUT=*
15 //SYSTSIN  DD *
16 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
17 DSN SYSTEM(DBDBG)
18 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
19 LIB('EMPOT.ZPICL.LOAD')
20 END
21 /*
22 //

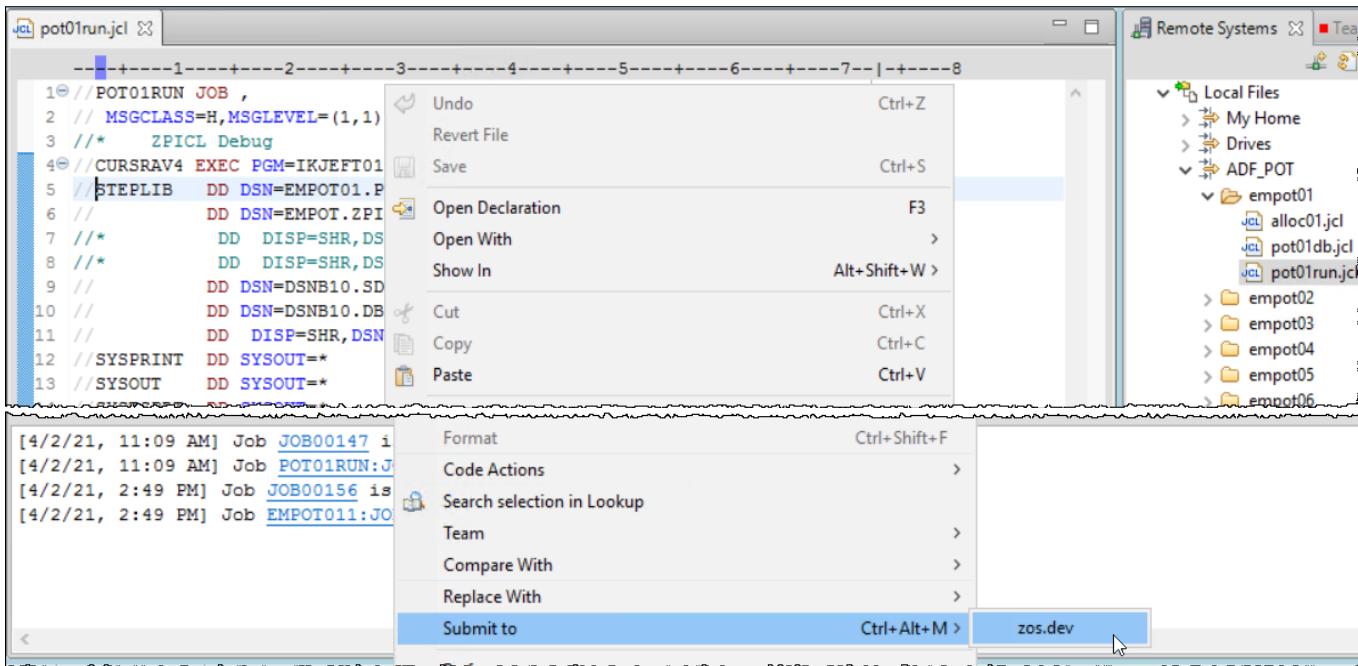
Properties
  POTO1RUN JOB
    CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
      STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
        DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
        DD DSN=DSNB10.SDSNLOAD,DISP=SHR
        DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
        DD DISP=SHR,DSN=FEK910.SFEKAUTH
        SYSPRINT DD SYSOUT=*
        SYSOUT   DD SYSOUT=*
        SYSTSPRT DD SYSOUT=*
        SYSTSIN  DD *

Outline
  1
  2

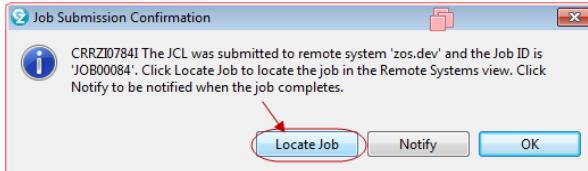
```

[5/28/19, 10:34 AM] Job [JOB00158](#) is being submitted
[5/28/19, 10:34 AM] Job [EMPOT011:JOB00158](#) ended with completion code

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



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



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

► Using *Remote Systems* view, verify the results double expanding **POT01RUN:JOB00xxx** (where **00xxx** can be any number) and clicking on **CURSRAV4 : SYSOUT**. Now you MUST have 000 as return code
Tip: You may need to refresh Retrieved Jobs to see it go from active to finished

The screenshot shows a software interface for managing z/OS environments. On the left, a terminal window displays the output of a JCL job named 'POT01.POT01RUN.JOB00113.D0000103.?spool'. The output includes various statistics and messages, such as:

```

33 *DEPT*      REG
34 *PERF-AVG*   0000000
35 *PERF-MIN*   0009
36 *PERF-MAX*   0009
37 *HOURS-AVG*  0002675
38 *HOURS-MIN*  0002675
39 *HOURS-MAX*  0002675
40 REG     .00  9.00  9.00    26.75   26.75   26.75
41 *DEPT*      N/A
42 *PERF-AVG*   0000000
43 *PERF-MIN*   0000
44 *PERF-MAX*   0000
45 *HOURS-AVG*  0003545
46 *HOURS-MIN*  0003545
47 *HOURS-MAX*  0003545
48 N/A     .00  .00  .00    35.45   35.45   35.45
49 *** END - OF - DATA ***
50 * * * ROWS READ --> 06
51 The result is ... 33
52 Thanks to ██████████ attending this
53 BRANCHFLAG GREATER THAN 1
54 EXECUTED SEEYA PARAGRAPH
55 EXECUTED SEEYA PARAGRAPH
56 EXECUTED GOODBYE PARAGRAPH

```

A red box highlights the last six lines of the output, which are the results of a COBOL program. A red arrow labeled '3' points from the 'Remote Systems' tree to the terminal window.

The 'Remote Systems' tree on the right shows the following structure:

- LAB2C
- LAB2D
- LAB7
- LAB8
- snippets
 - To_COPY_and_PASTE.txt
- My Favorites
- Local Shells
- zos.dev
 - z/OS UNIX Files
 - z/OS UNIX Shells
- MVS Files
- TSO Commands
- JES
 - Retrieved Jobs
 - POT01RUN:JOB00113 [CC 0000] (highlighted)
 - JES2:JESMSGLG
 - JES2:JESJCL
 - JES2:JESVMSG
 - CURSRAV4:SYSTSIN [0000]
 - CURSRAV4:SYSOUT [0000] (highlighted)
 - CURSRAV4:SYSTSPRT [0000]
 - POT01CC:JOB00092 [CC 0000]

Red circles labeled '1' and '2' point to the 'POT01RUN:JOB00113 [CC 0000]' entry and the 'CURSRAV4:SYSOUT [0000]' entry respectively, both of which are highlighted with red boxes.

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.

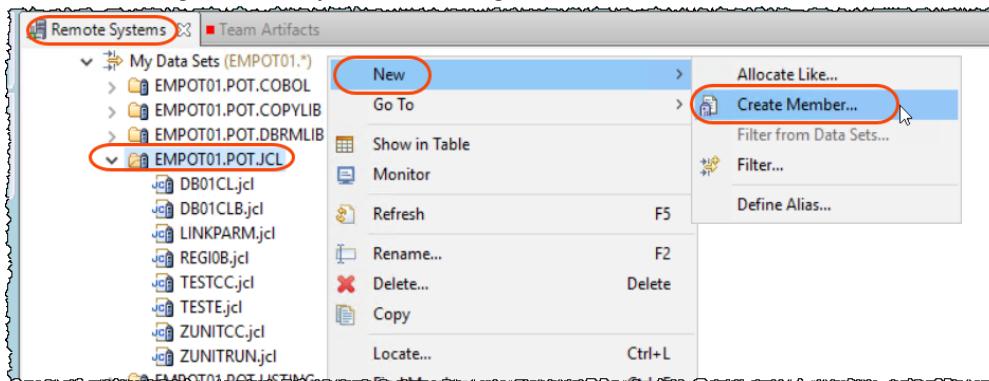
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode. A sample JCL procedure, to process a single program run, and a set of scripts to start and stop a permanently active Code coverage collector that can process multiple program runs.

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.

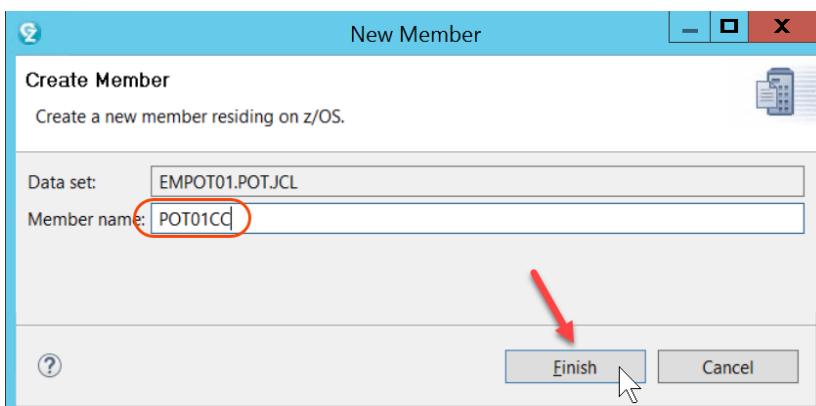
6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

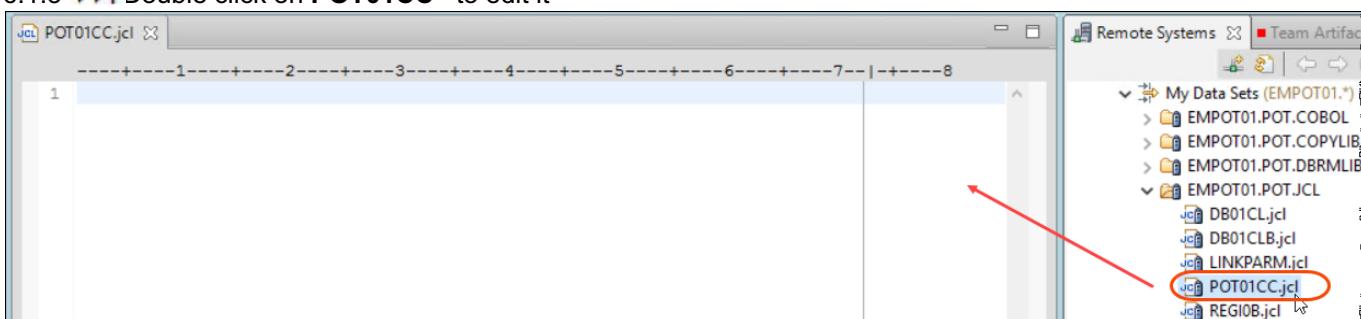
6.1.1 ►| Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member...**



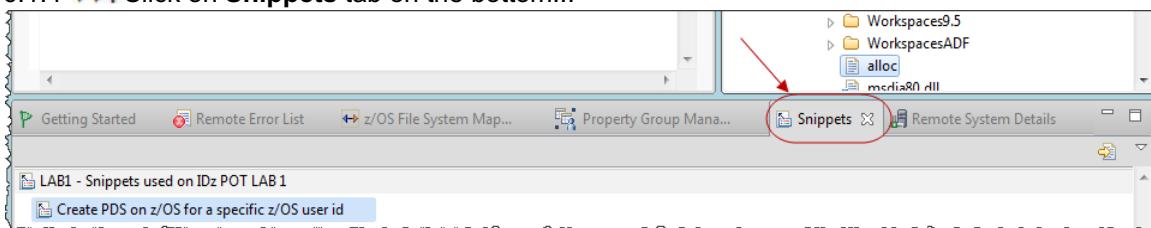
6.1.2 ►| Name it as **POT01CC** and click **Finish**



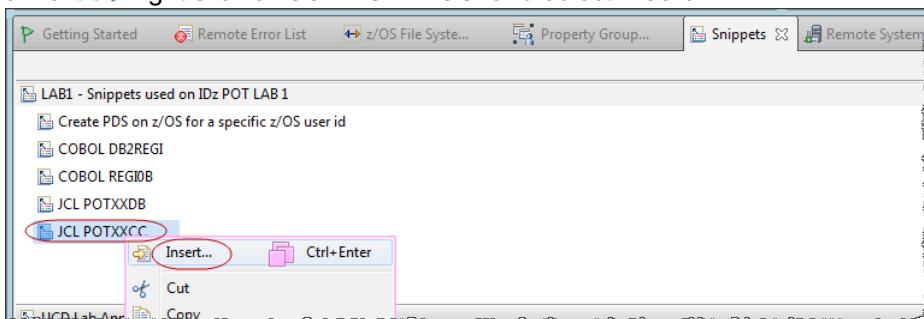
6.1.3 ► Double click on POT01CC to edit it



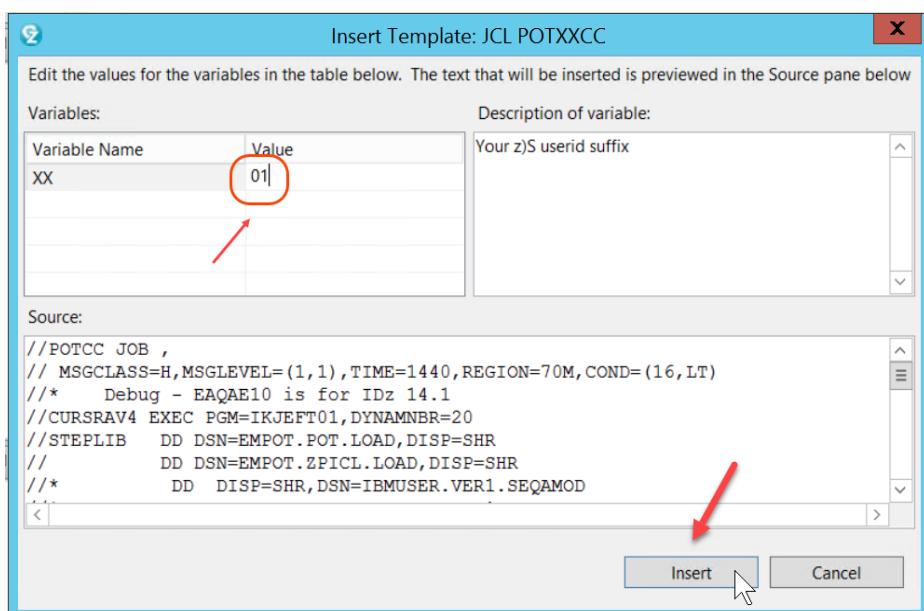
6.1.4 ► Click on Snippets tab on the bottom...



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.

```

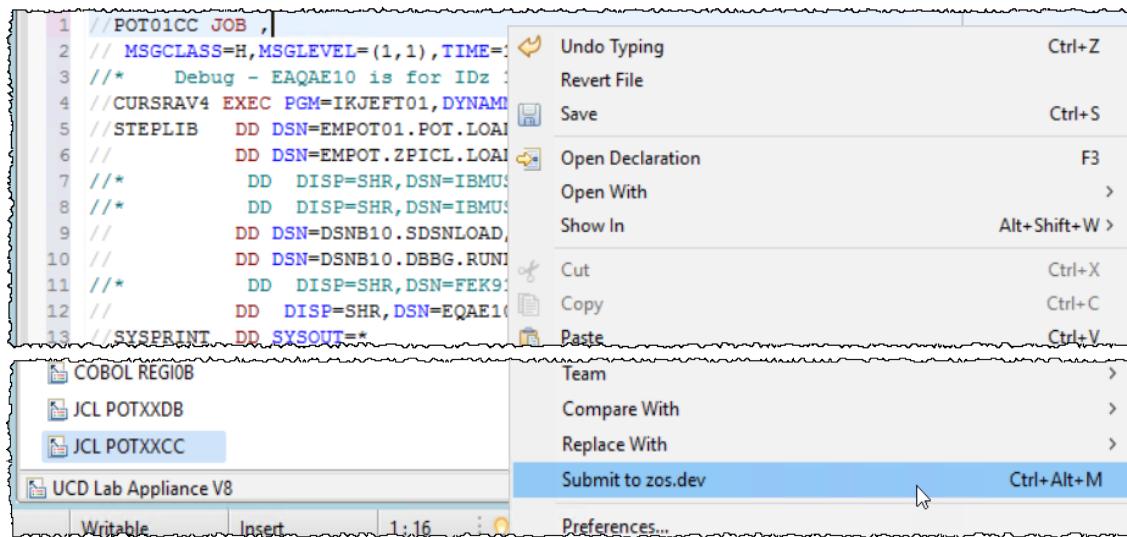
1 //POT01CC JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 //*
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
6 // DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
7 /**
8 /**
9 /**
10 /**
11 /**
12 /**
13 /**
14 /**
15 /**
16 /**
17 TSOLIB ACTIVATE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

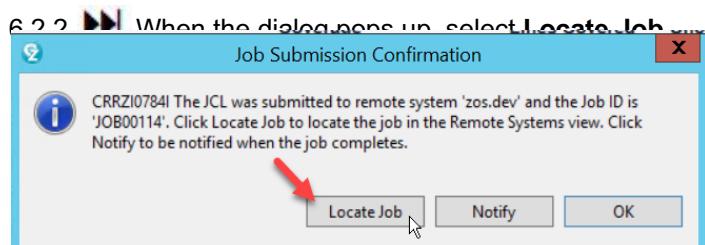
```

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**





6.2.3 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

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	

Below the table, a table lists workspace results with a red circle around the 'Coverage' column showing 73%.

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**.

```

351      EXIT.
352      MOVE ROW-KTR TO ROW-STAT.
353      DISPLAY ROW-MSG.
354      350-EXIT.
355      *      EXIT.
356      GO TO 500-SECOND-PART.
357      999-ERROR-TRAP-RTN.
358
359      **** WE HAVE A SERIOUS PROBLEM HERE ****.
360      999-ERROR-TRAP-RTN .
361      MULTIPLY SQLCODE BY -1 GIVING SQLCODE.
362      DISPLAY 'SQLCODE ==> ' SQLCODE.
363      DISPLAY SQLCA.
364      DISPLAY SQLERRM.
365      EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
366      EXEC SQL ROLLBACK WORK END-EXEC.
367
368      500-SECOND-PART.
369      MOVE 2 TO BRANCHFLAG.
370      MOVE 'AAAAAA' to FIELD-A.
371      MOVE 'BBBBBB' to FIELD-B.
372      MOVE 'CCCCCCCC' to FIELD-C.

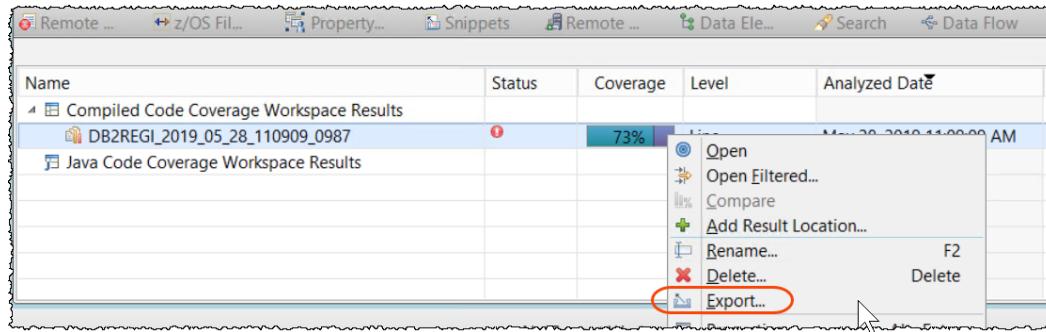
```

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

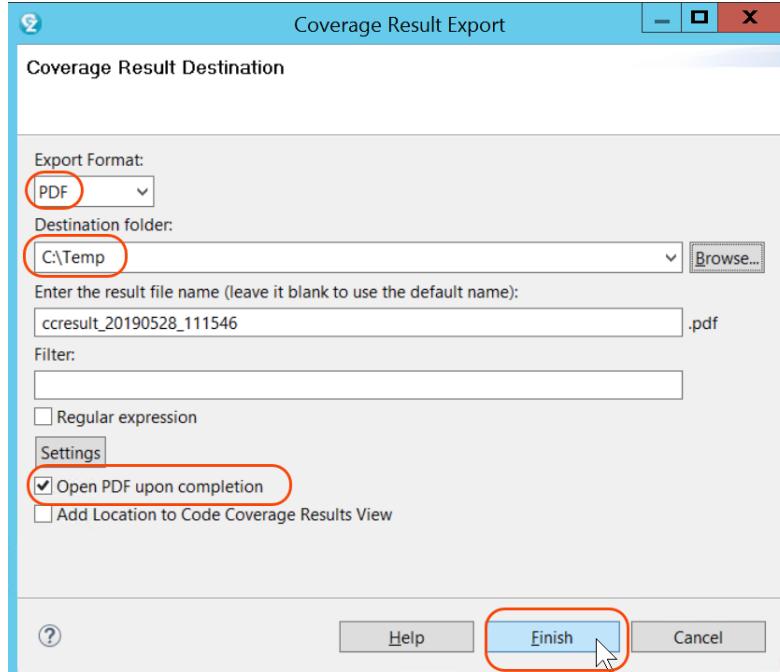
6.2.6 ► You could also create a PDF report as documentation or export this data to other products (like ADDI or SonarQube)

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.

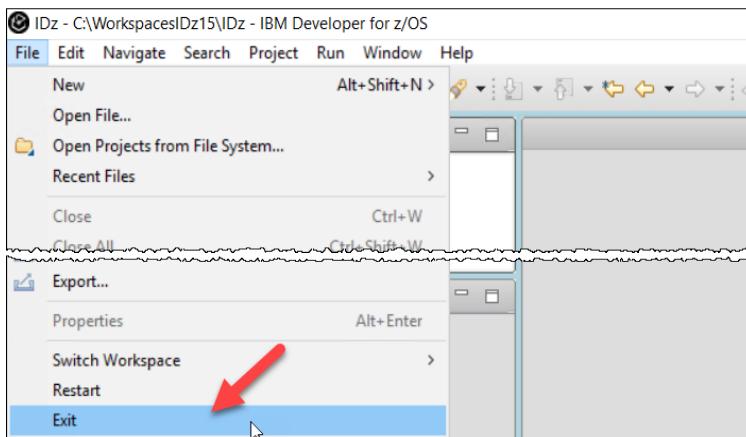
Data Studio is a foundational offering that includes support for key tasks across the data management lifecycle, including administration, application development, and query tuning.

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**

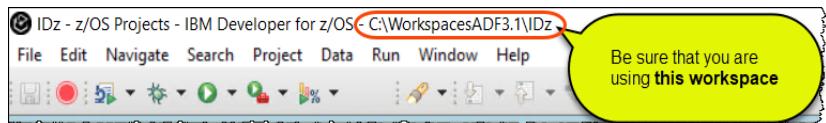


7.0.2 Start **IBM Developer for z Systems version 14**

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

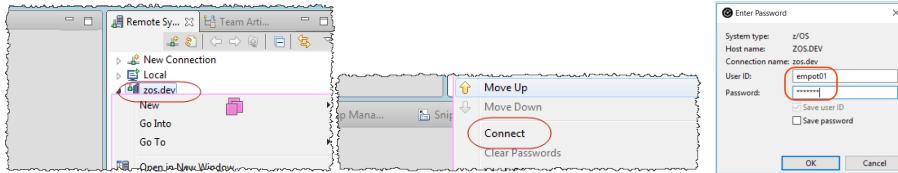
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.





7.0.3 To Connect to the z/OS system:

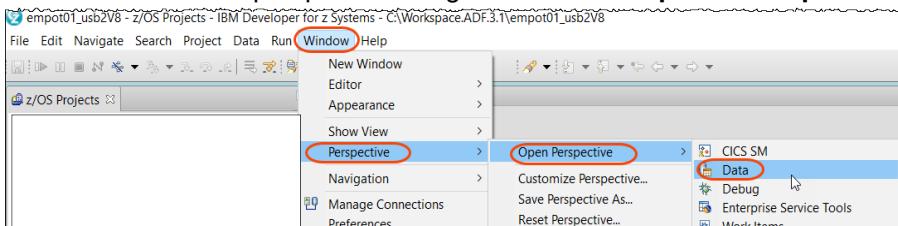
- ▶▶ Using the **Remote Systems** view, right-click on **zos.dev** and select **Connect**.
- ▶▶ Use **empot01** and password **empot01**



7.1 Creating a connection to the DB2 on z/OS

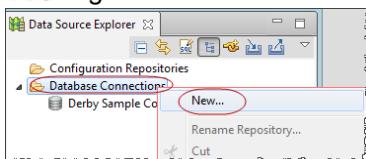
You will need to have authorization to connect to the DB2. Now you will use **IBMUSER** as a new z/OS ID

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

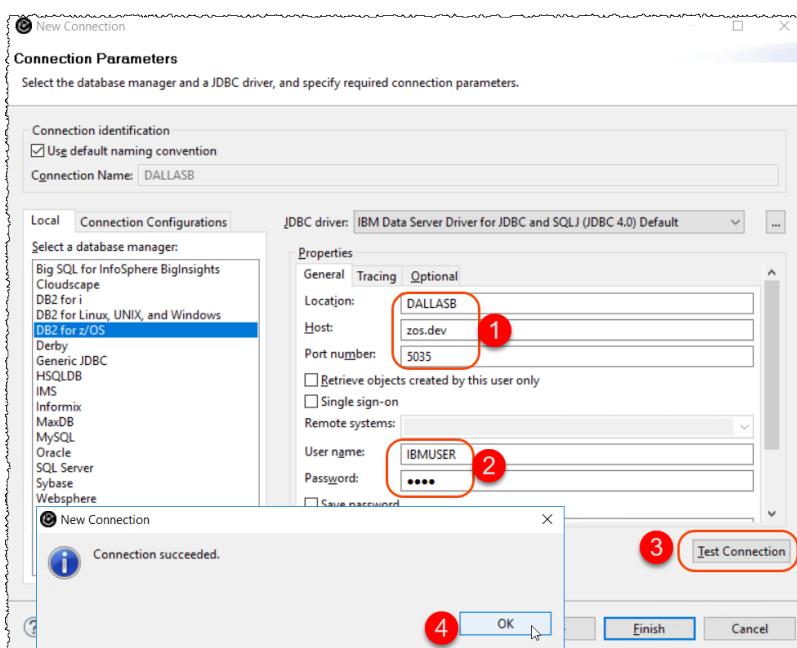


7.1.2 Using the Data Source Explorer view, create a new z/OS DB2 connection:

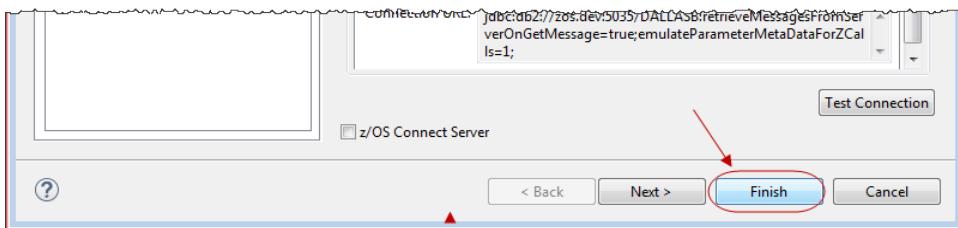
- ▶▶ Right click on **Database Connections** and select **New..**



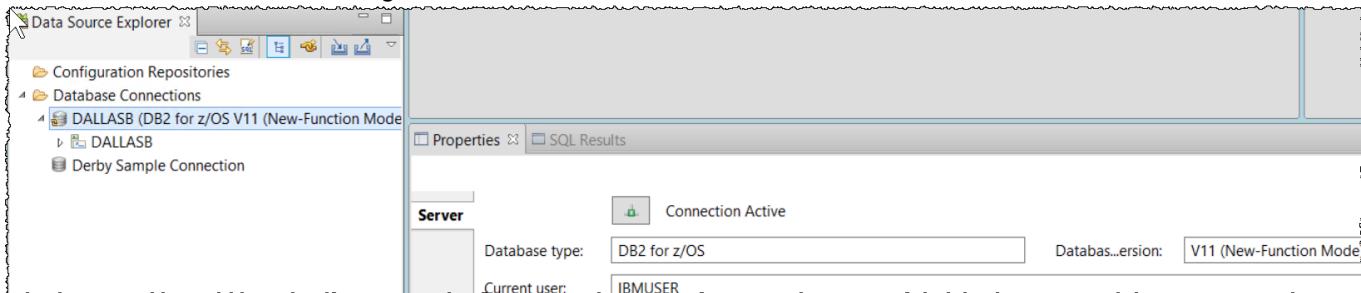
7.1.3 ▶▶ Select **DB2 for z/OS**, use **Location** as **DALLASB**, Host is **zos.dev** and port is **5035**, **IBMUSER** and password **SYS1**. Click **Test Connection** to be sure you can successfully connect to the DB2.



7.1.4 ► Click OK and Finish



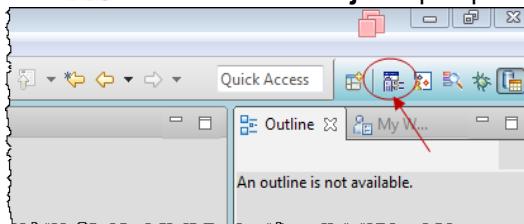
The connection is created. You could see tables, modify contents etc..
But we want to focus on the integration of DB2 and the COBOL/DB2 editor.



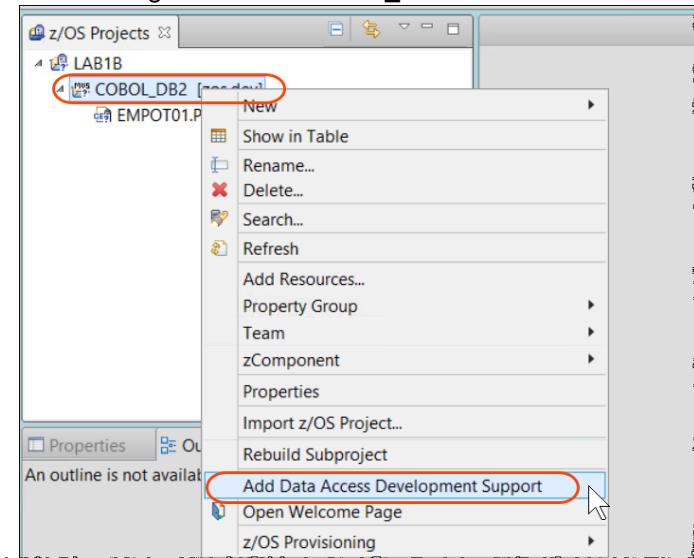
7.2 Running a SQL query from COBOL program

You will need to have authorization to run the queries.

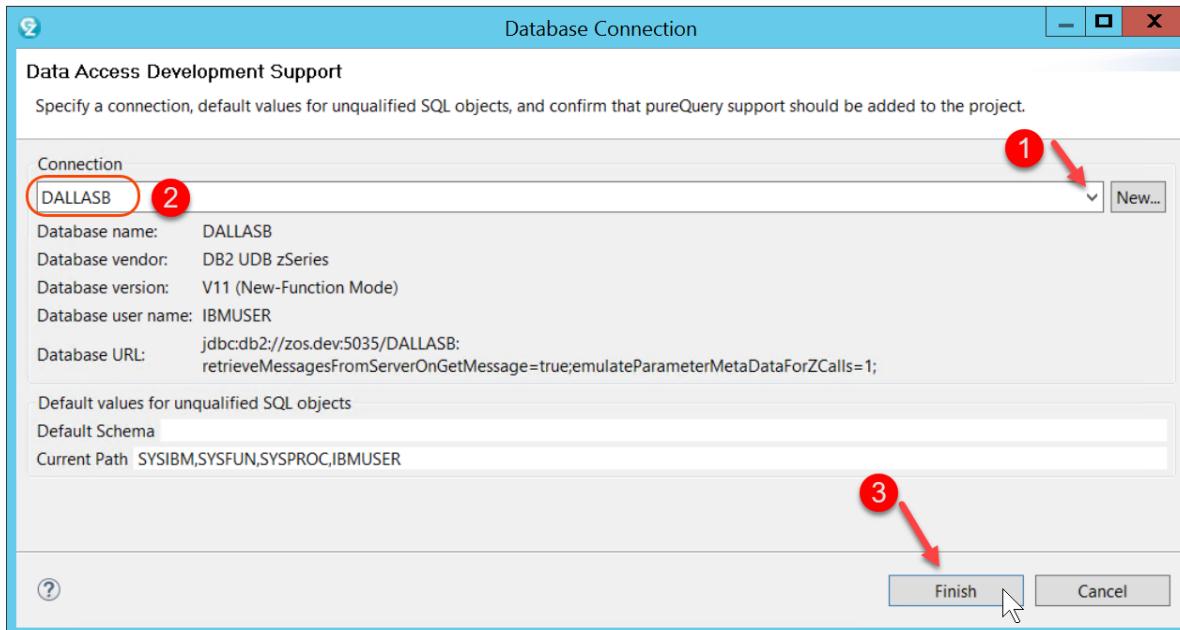
7.2.1 ► Go to the z/OS Projects perspective



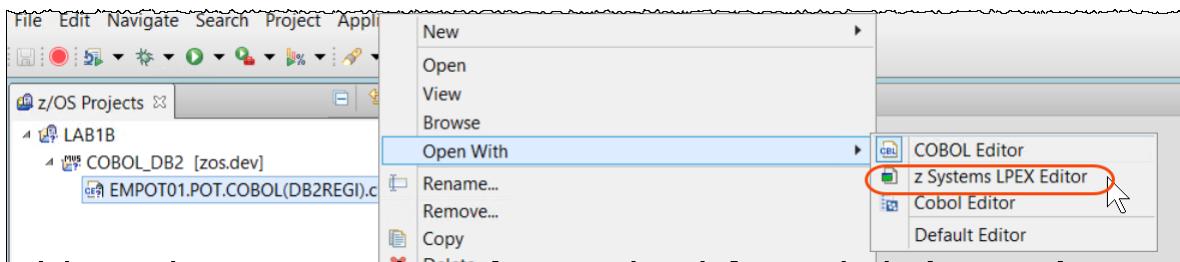
7.2.2 ► Right click on COBOL_DB2 and select Add Data Access Development Support



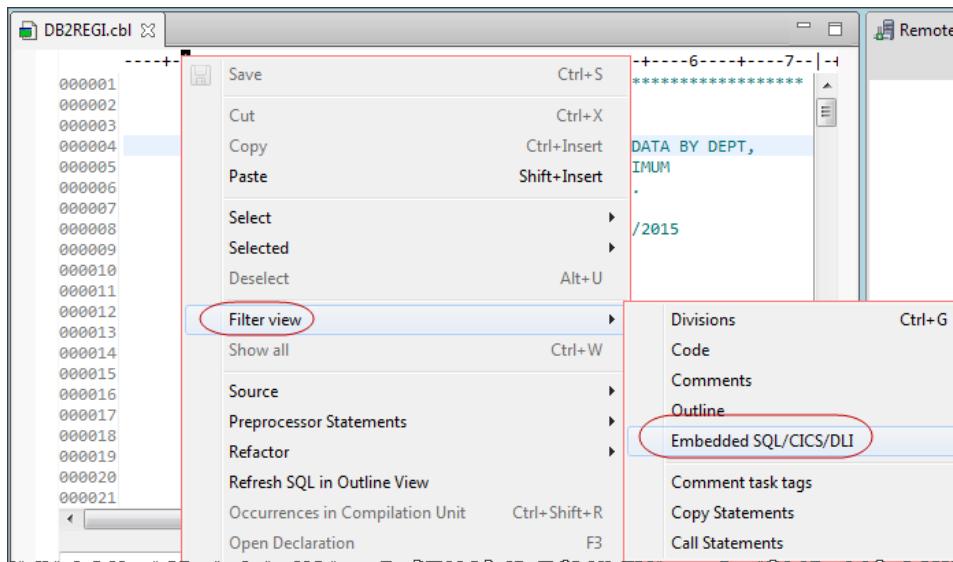
7.2.3 ➡ Select the connection created **DALLASB** and click **Finish**



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 'Run SQL' option is highlighted with a red circle. Other options in the menu include Paste, Select, Selected, Deselect, Filter view, Show all, Source, Preprocessor Statements, Refactor, Tune SQL, 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.
000135      EXEC SQL OPEN C1
000136      END-EXEC.
000187      EXEC SQL FETCH C1 INTO
000188      :DEPT-TBL:DEPT-NULL,
000189      :PERF-TBL-MIN:PERF-NULL,
000190      :PERF-TBL-MAX:PERF-NULL,
000191      :PERF-TBL-AVG:PERF-NULL,
000192      :HOURS-TBL-MIN:HOURS-NULL,

```

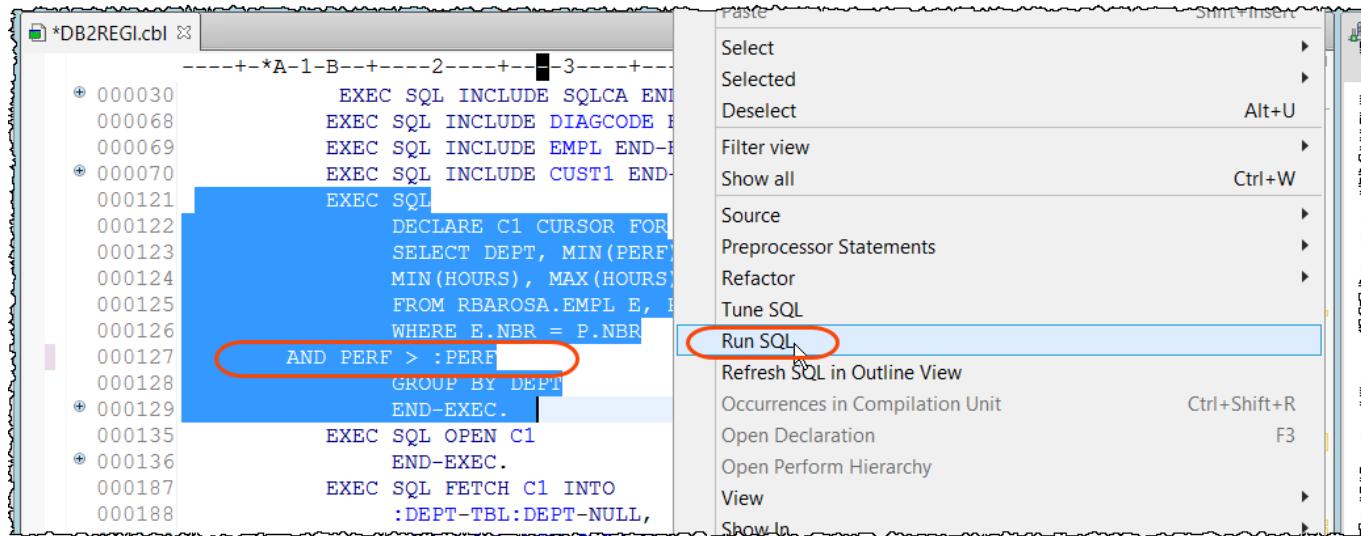
7.2.7 ► Click as below and you will have the query results

The screenshot shows the DB2REGL.cbl editor window after running the SQL. The status bar at the bottom left shows a success message: 'Success SELECT D... 8/12/19, ... DALLASB'. The status bar at the bottom right has a red circle with the number '2' and the label 'Result1' pointing to the results grid. The results grid displays data from the query, including columns DEPT, 2, 3, 4, 5, 6, and 7.

	DEPT	2	3	4	5	6	7
1	ACC	8	8	8	15.99	15.99	15,990...
2	FIN	3	9	5	8.89	32.45	22,696...
3	MKT	1	3	1	13.23	32.41	22,820...
4	R&D	1	1	1	NULL	NULL	NULL
5	REG	9	9	9	26.75	26.75	26,750...
6	NULL	0	0	0	35.45	35.45	35,4500...

7.2.8 Modifying the SQL query..

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

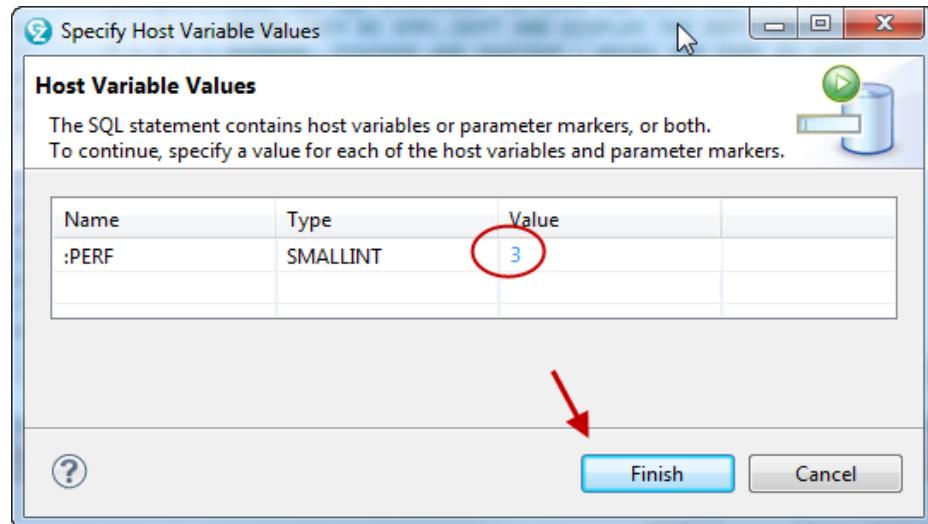


```

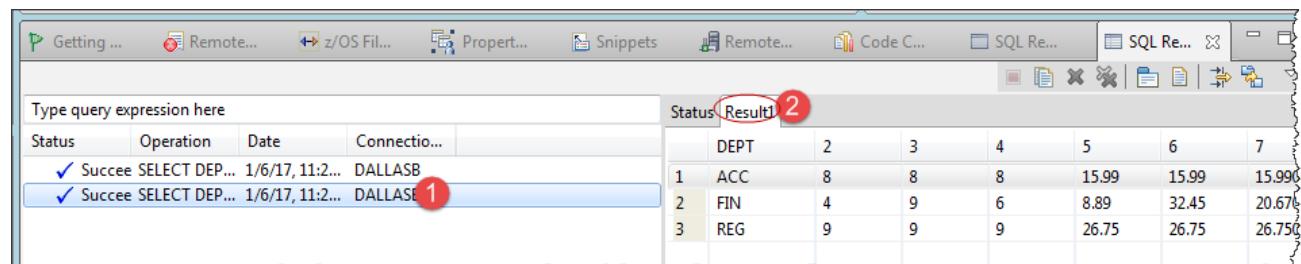
*DB2REGL.cbl
-----+-----+-----+-----+
+ 000030      EXEC SQL INCLUDE SQLCA ENI
 000068      EXEC SQL INCLUDE DIAGCODE I
 000069      EXEC SQL INCLUDE EMPL END-I
+ 000070      EXEC SQL INCLUDE CUST1 END-
 000121      EXEC SQL
 000122      DECLARE C1 CURSOR FOR
 000123      SELECT DEPT, MIN(PERF)
 000124      MIN(HOURS), MAX(HOURS)
 000125      FROM RBAROSA.EMPL E, I
 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
 000188      :DEPT-TBL:DEPT-NULL,

```

7.2.9 ► Specify a value like 3 and click Finish



7.2.10 The new results now will be:



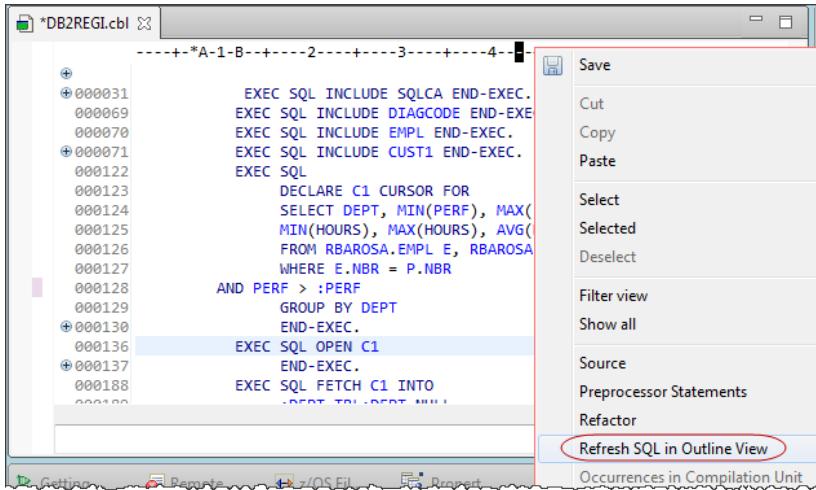
Type query expression here

Status	Result
1	Succee SELECT DEP... 1/6/17, 11:2... DALLAS
2	Succee SELECT DEP... 1/6/17, 11:2... DALLAS

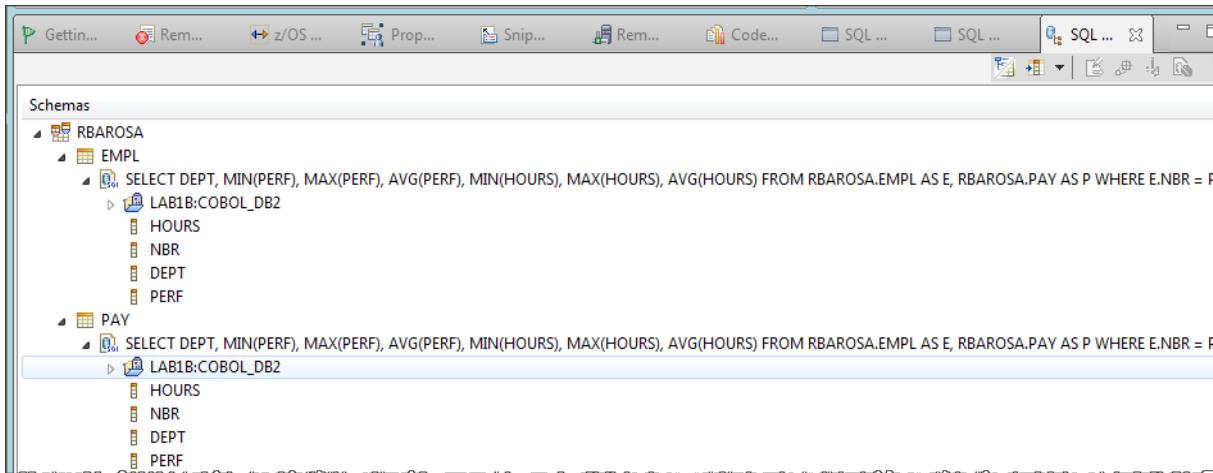
	DEPT	2	3	4	5	6	7
1	ACC	8	8	8	15.99	15.99	15.99
2	FIN	4	9	6	8.89	32.45	20.67
3	REG	9	9	9	26.75	26.75	26.75

7.3 Using the SQL Outline view

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

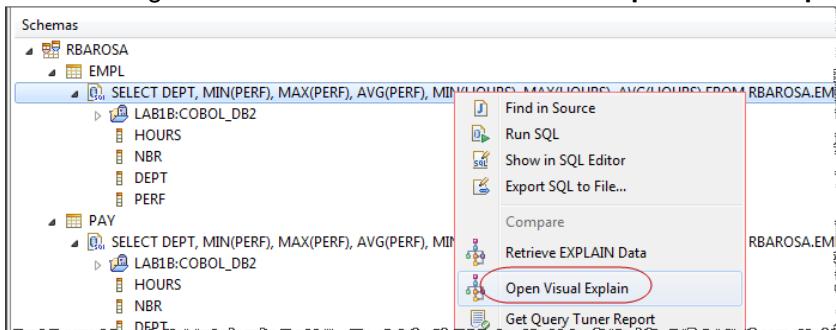


7.3.2 Using SQL Outline View, expand RBAROSA, EMPL, PAY and the SQL statements and you will have more details as below

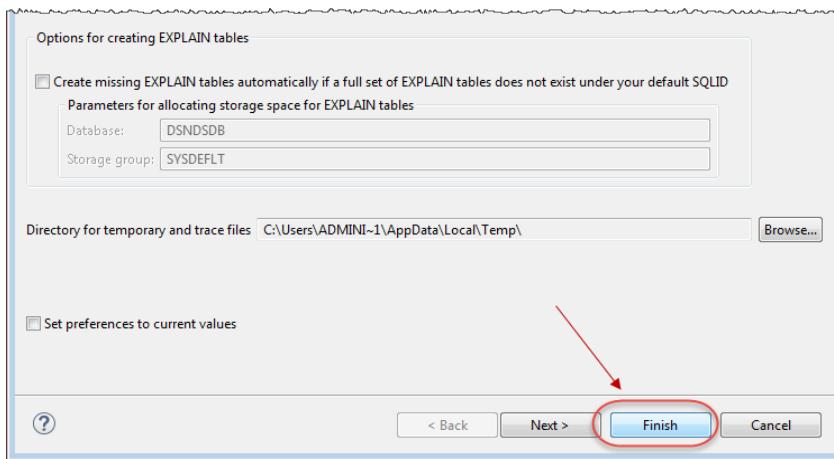


7.4 Using SQL Visual Explain

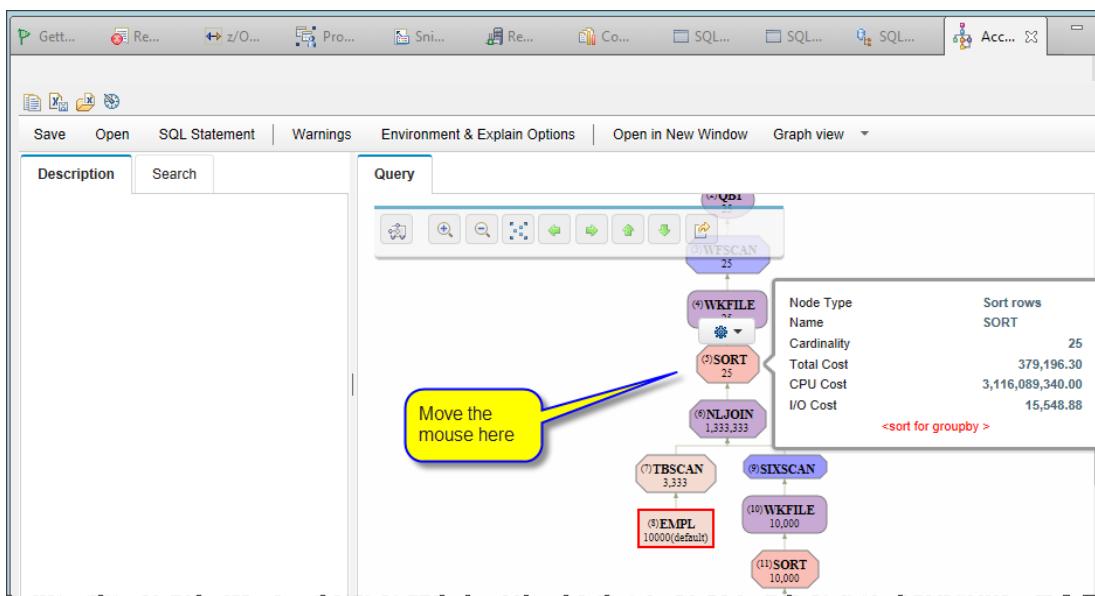
7.4.1 Right click on the first Select and choose Open Visual Explain



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

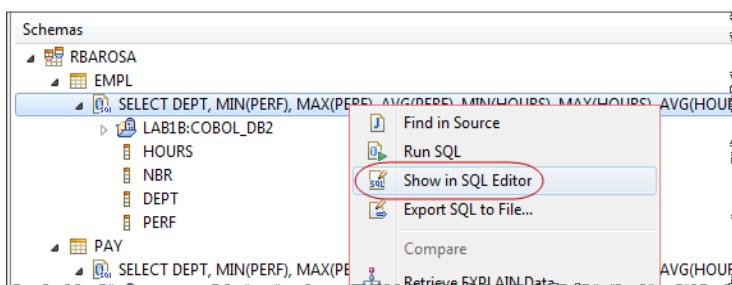


7.4.3 ► Double click on the tab to have a bigger view.
You will have the report below to work with the SQL explain

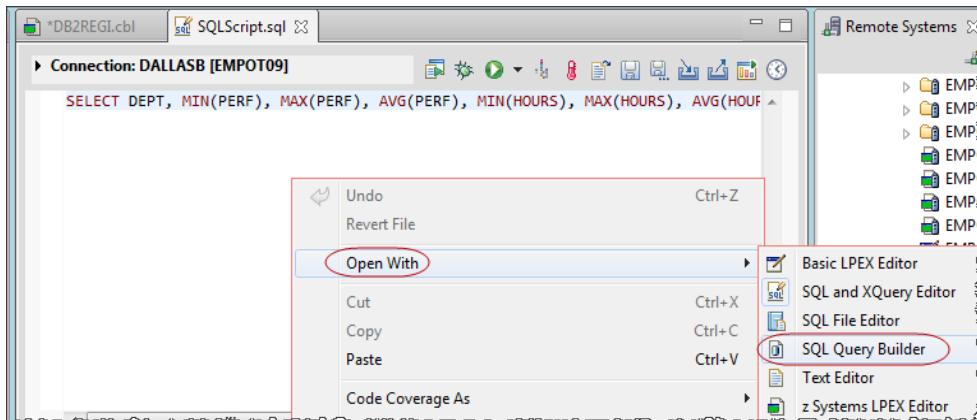


7.5 Reverse engineer the Query

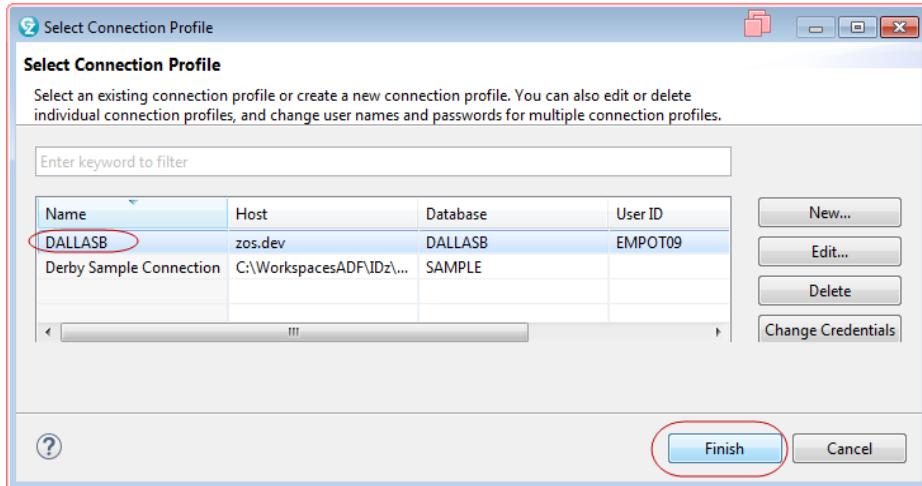
7.5.1 ► Using the SQL Outline view, right click on the Select and choose **Show in SQL Editor**



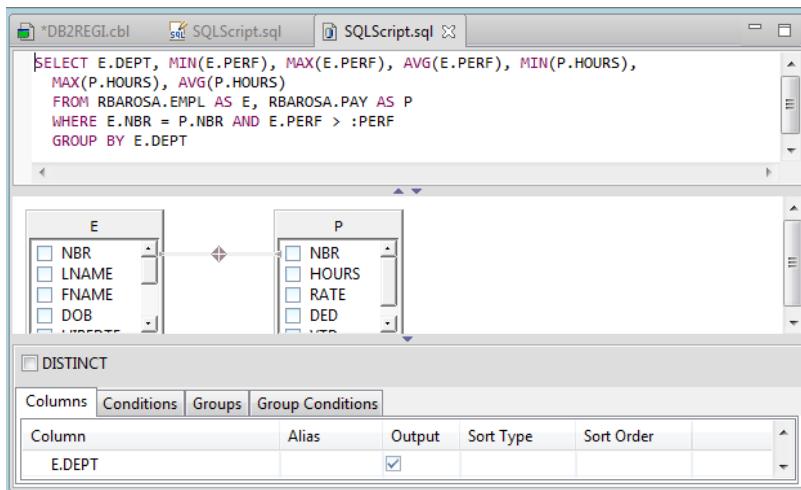
7.5.2 ► When opened, select Open With > SQL Query Builder



7.5.3 ► Choose DALLASB and click Finish



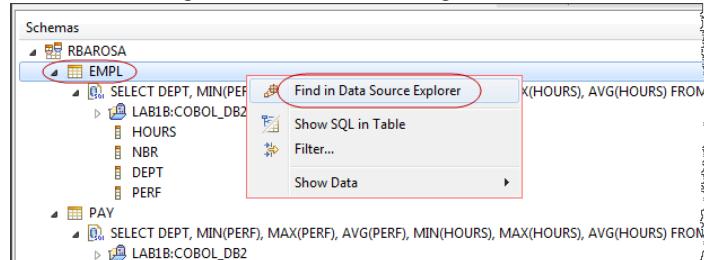
7.5.4 Now we have the graphical editor



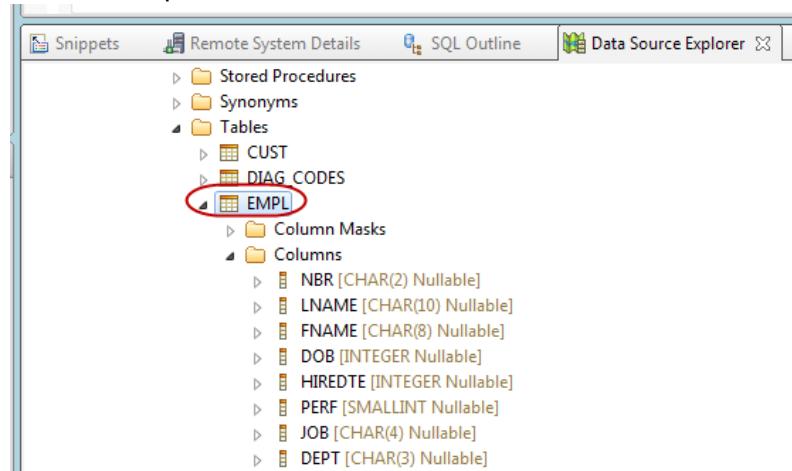
7.5.5 ► You might use this graphical editor to make changes on SQL statements. Notice that the changes reflects on the COBOL code..

7.6 Displaying DB2 table content

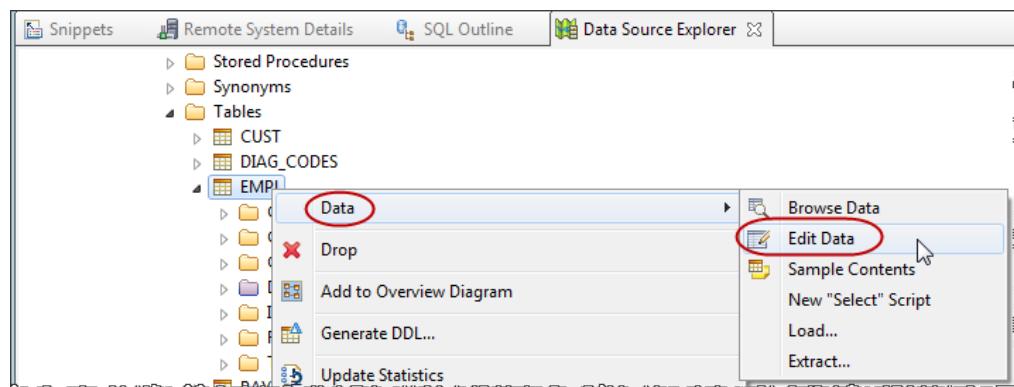
7.6.1 ► Using SQL Outline view, right click on EMPL and select Find Data Source Explorer



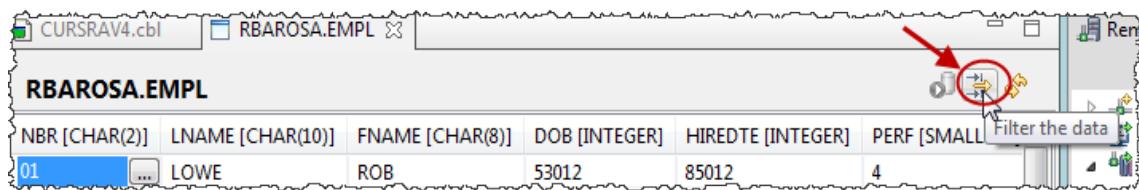
7.6.2 ► Expand Tables and EMPL



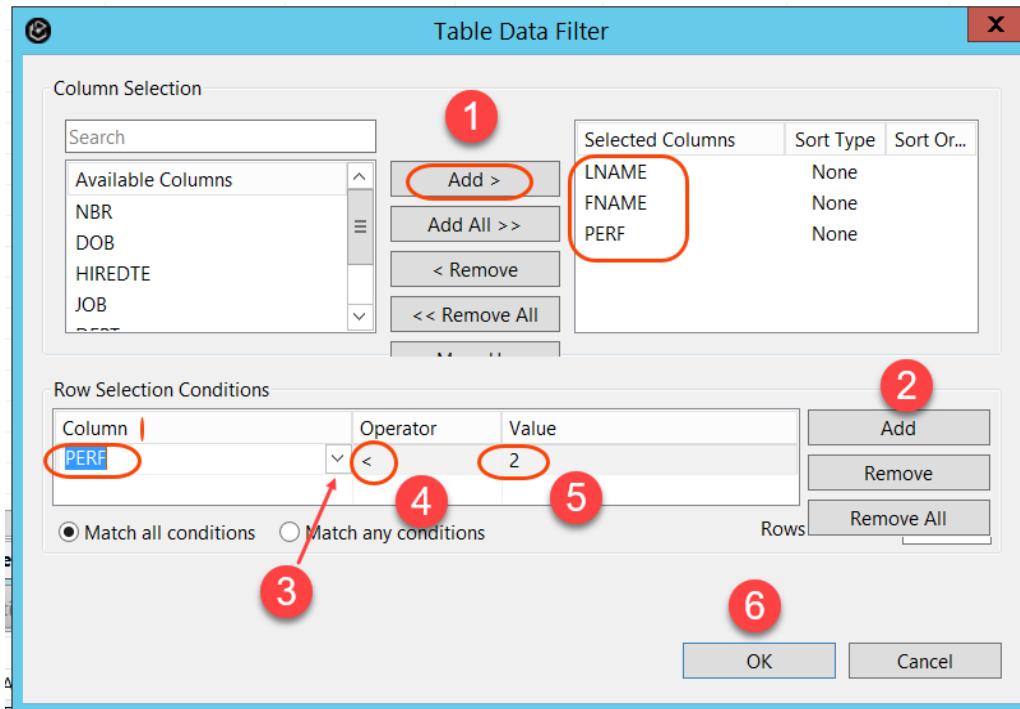
7.6.3 ► Right click on EMPL and select Data > Edit Data



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.

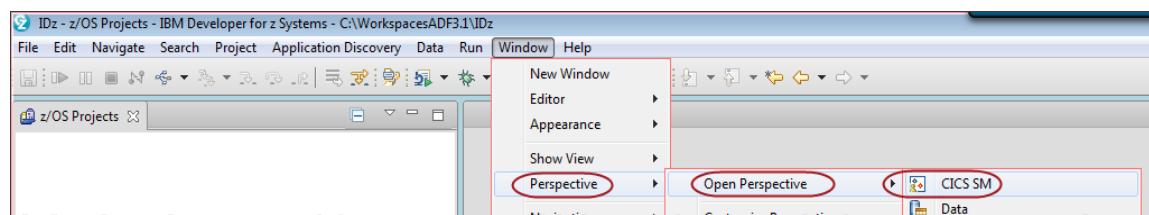
Note that this exercise could be also done with either IDz version 14 or version 15. Here we are using version 14, but instructions will be the same.

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.

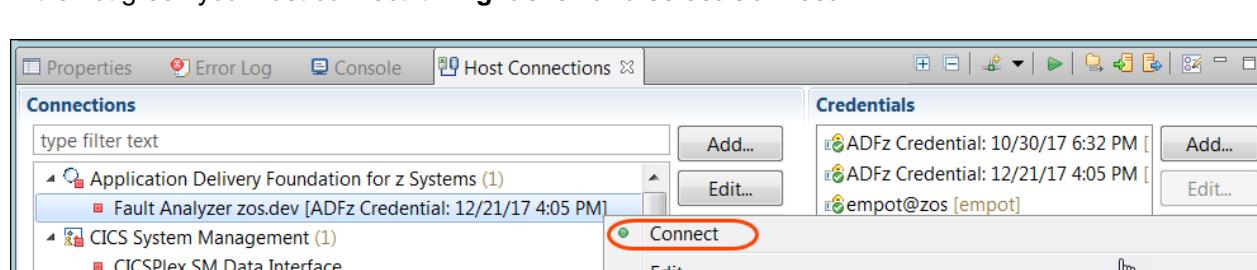
8.1.1 ► Go to **CICS SM** perspective using

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**

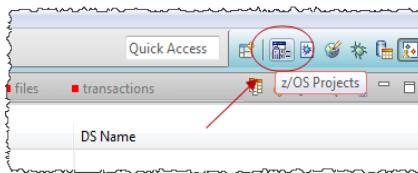


8.2 Using View Load Module utility

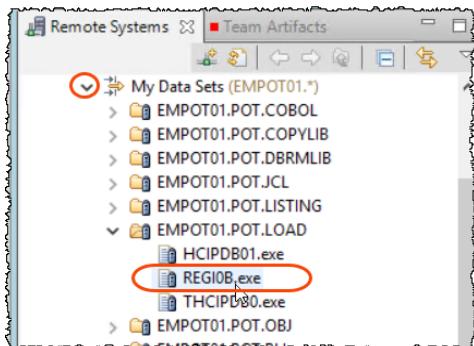
You can use the **View Load Module** utility function to print a list of the symbols (CSECTs, common sections, entry points, and ZAPs) in a load module. You also could compare load modules, using the Compare wizard.

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](#).

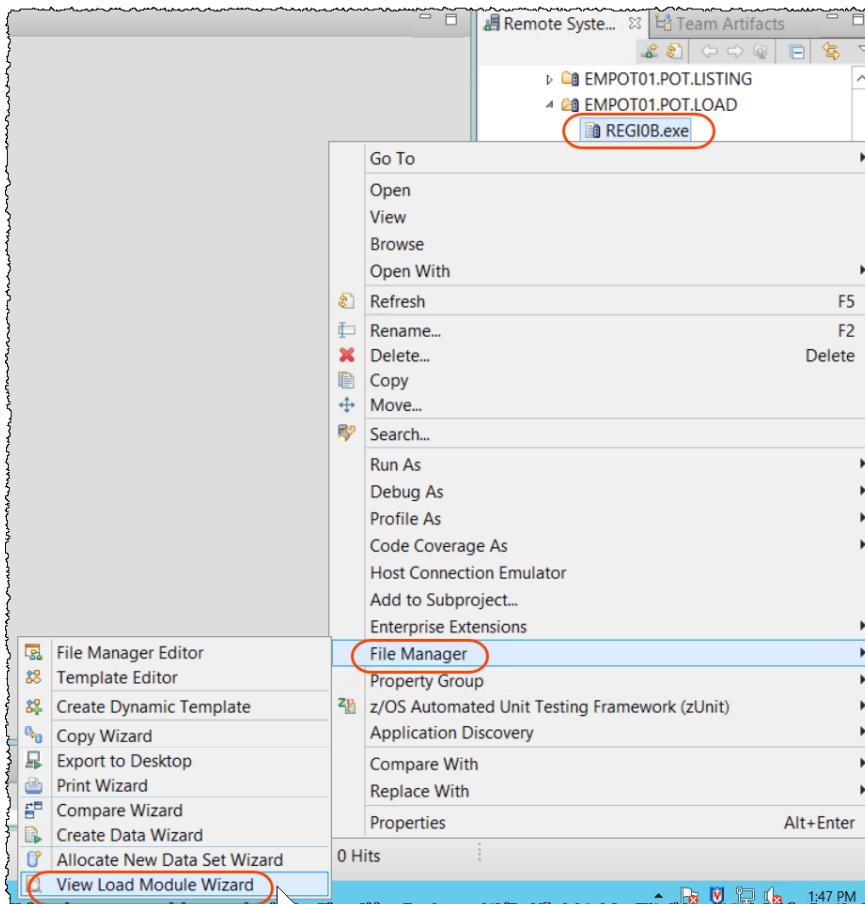
8.2.1 ► Go to **z/OS Projects** perspective clicking on the upper top right icon as below



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..

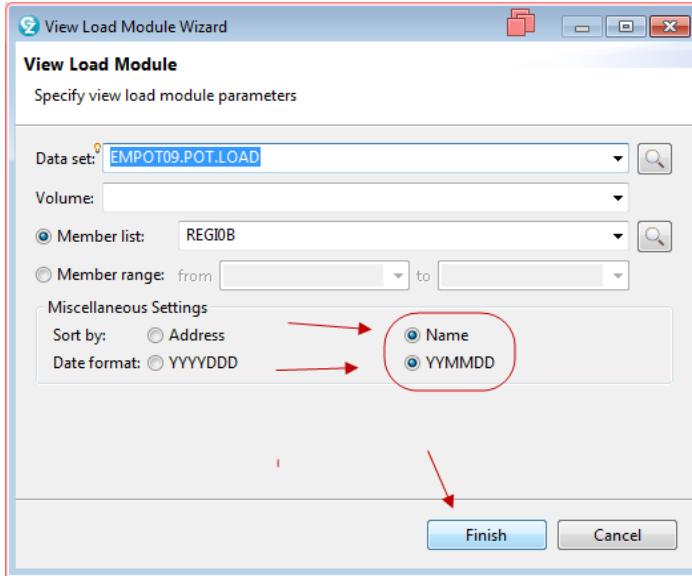


8.2.3 ► Right click on **REGI0B.exe** and select **File Manager** and **View Load Module Wizard**.



- ▶ 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.

```

000018 b
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

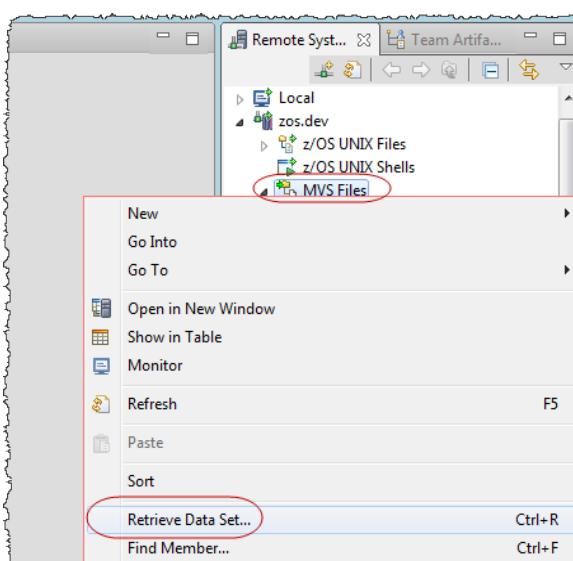
In this section, you will open a variable blocked sequential dataset in the editor or viewer, using a copybook as a layout.

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.

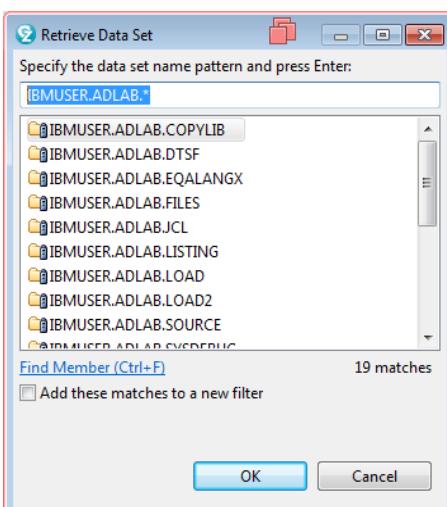
8.3.1 Before you start mapping the dataset using a COBOL COPYBOOK, let us copy the copybook that maps the file to your PDS.

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

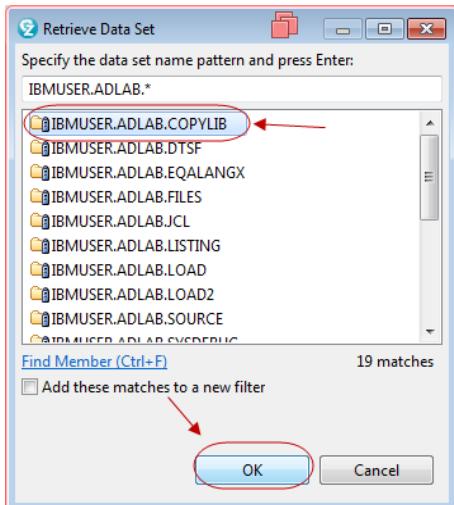
This is equivalent at the TSO 3.4 function.



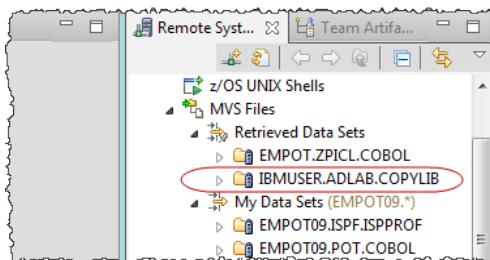
8.3.2 ► Type **IBMUSER.ADLAB.*** and press **Enter**.



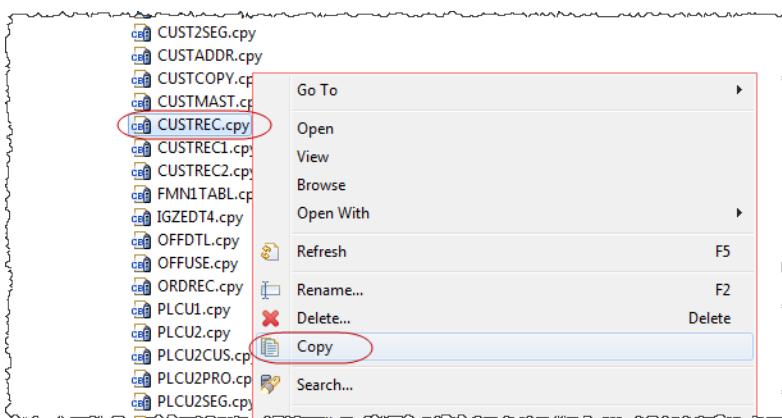
8.3.3 ► Select **IBMUSER.ADLAB.COPYLIB** and click **OK**.



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

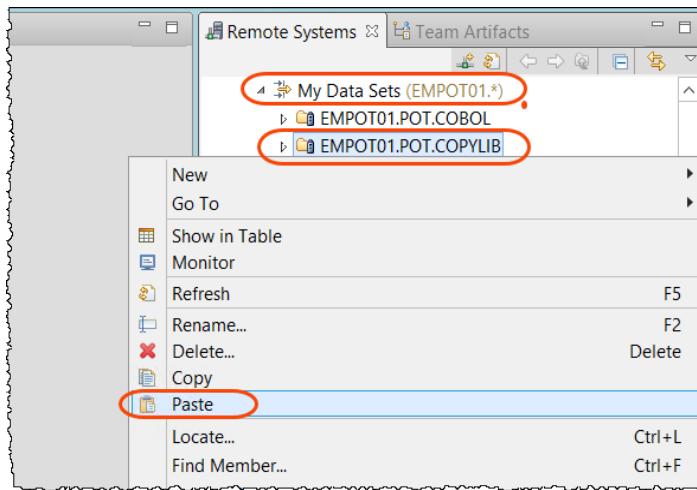


8.3.5 ► Expand **IBMUSER.ADLAB.COPYLIB**, right click on **CUSTREC.cpy** and select **Copy**. You want to copy this member to your PDS.

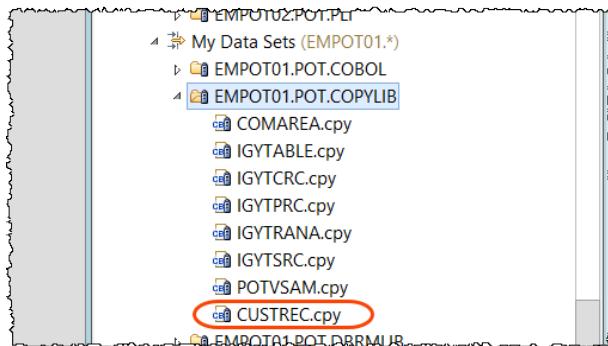


8.3.6 ► Navigate to **My Data Sets**, right click on **EMPOT01.POT.COPYLIB** and select **Paste**

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.

The screenshot shows the COBOL editor with the file 'CUSTREC.cpy' open. The code is as follows:

```

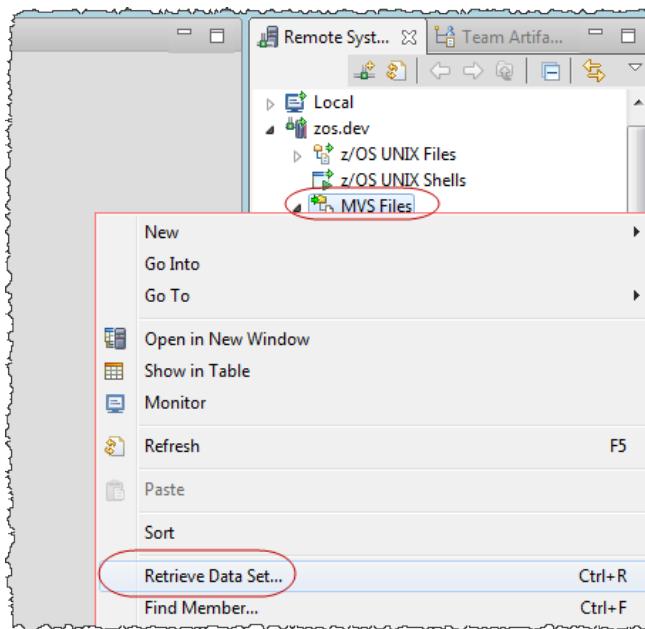
1   *A-1-B-----2----+---3---+---4---+---5---+---6---+---7---+---8
1   *** ++++++++-----+
2   * Sample COBOL Copybook for IBM PD Tools Workshops
3   *
4   * The sample data described by this copybook
5   * is <USERID>.ADLAB.CUSTFILE
6   *** ++++++++-----+
7   01 CUST-REC.
8     05 CUSTOMER-KEY.
9       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 TAB-DATA-1       PIC X(05).

```

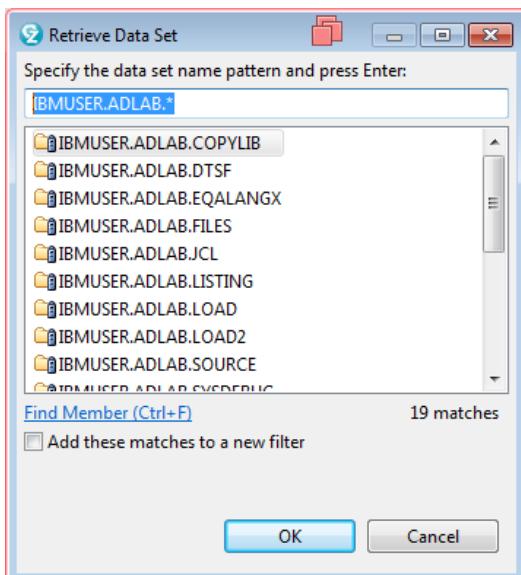
A red arrow points from the right margin of the code window to the 'CUSTREC.cpy' file in the 'My Data Sets' tree view on the right.

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

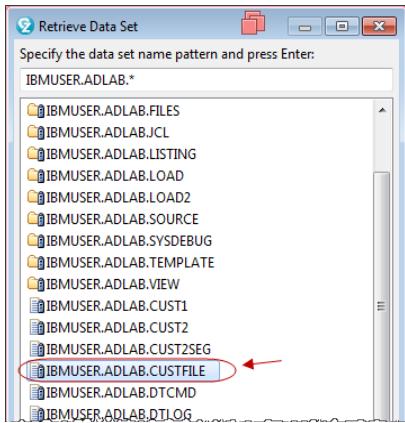
This is equivalent at the TSO 3.4 function.



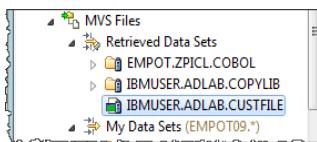
8.3.10 ► Type **IBMUSER.ADLAB.*** and press **Enter**.



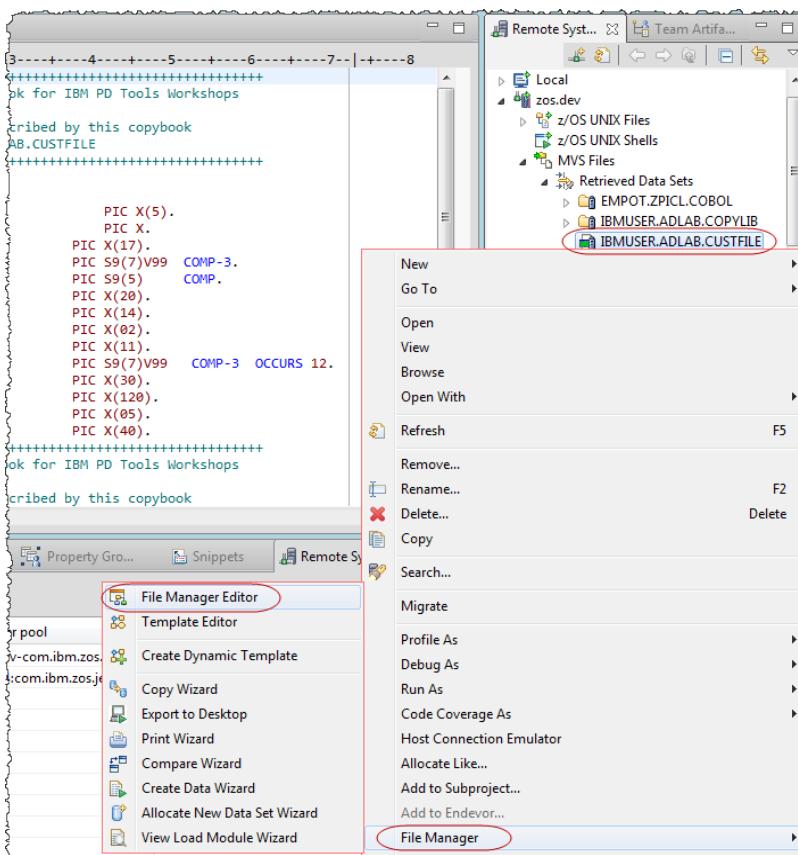
8.3.11 ► Scroll down, select **IBMUSER.ADLAB.CUSTFILE** and click **OK**.



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

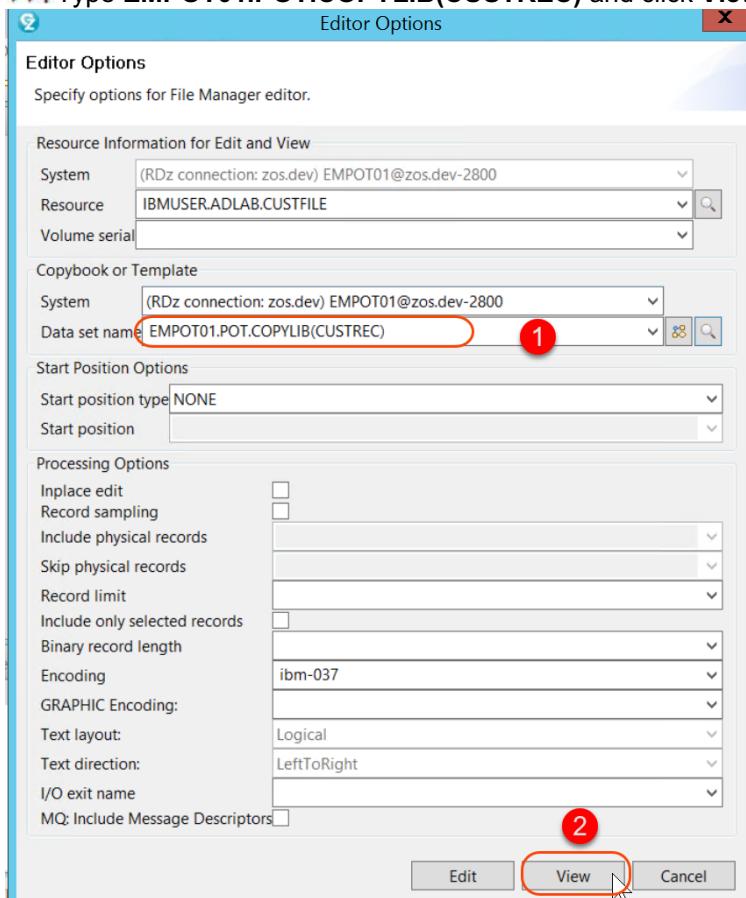


8.3.13 ► Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**. You want to map this file against the COBOL COPYBOOK that you copied.



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.

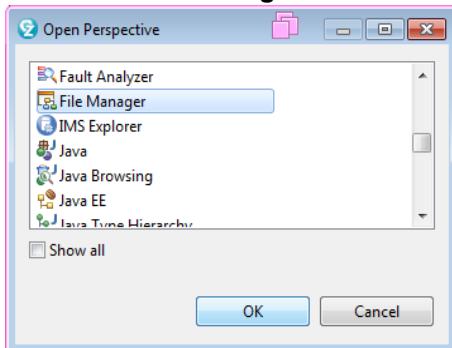
The screenshot shows the File Manager editor interface. The left pane displays a table of customer records (CUST-ID, REC-TYPE, NAME, ACCT-BALANCE, ORDERS-YTD, ADC) in a formatted view. The right pane shows the file structure under 'Remote Syst...' and 'Team Artifacts', including files like REGIDC.cbl, EMPOT03.POT.COPYLIB, and various copybook members (CUSTREC.cpy, IGYTABLE.cpy, etc.).

	CUST-ID	REC-TYPE	NAME	ACCT-BALANCE	ORDERS-YTD	ADC
1	03115	A	Graham, Holly	254.53	1	316
2 CONTA...						
3 CONTA...						
4	05580	A	Moore, Adeline	498.95	3	470
5 CONTA...						
6 CONTA...						
7 CONTA...						
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 CONTA...						
13 CONTA...						
14 CONTA...						
15	11204	A	Ness, Luke	513.06	3	516
16 CONTA...						
17 CONTA...						

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	Y(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)..

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.

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

The screenshot shows a DB2 command line processor window titled '(RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE' with a file name 'CUSTREC.cpy'. The table has columns: CUST-ID, REC-TYPE, NAME, DESCRIPTION, and CONTACT-INFO. The data includes records for customers like Graham, Holly, Moore, Adeline, Marx, Audrey, etc., with various phone numbers. Records 2 and 3 are circled in red.

	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**

The screenshot shows a DB2 command line processor window with a context menu open over a record. The menu item 'Switch Mode' is highlighted with a red circle. Other menu items include 'Compare With', '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', and 'Replace With...'. The main table view shows customer records.

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 a DB2 command line interface window titled 'CUSTREC.cpy'. It displays a list of 19 records from a file named 'CUSTREC.cpy'. The data is presented in character mode, where fields are not automatically formatted. The first few records show fields like 'NAME', 'ADDRESS', 'CITY', 'STATE', and 'ZIP' containing names and addresses. The last record, record 19, shows a single field 'Cell Phone138-555-2410' with a length of 10 characters. The bottom of the window has a toolbar with icons for search, copy, and paste, and a dropdown menu labeled 'CHARACTER' which is circled in red.

	NAME	ADDRESS	CITY	STATE	ZIP	
1	03115AGraham, Holly	...á.....3100 Oaktree Ct	Raleigh	NCUSA		
2	03115B Graham, Holly	Home Phone112-555-6736	1234 5678			
3	03115CGraham, Holly	Cell Phone135-555-2338	9012 3456			
4	05580AMoore, Adeline	..ñi*....4700 S. Syracuse	Denver	COUSA		
5	05580BMoore, Adeline	Work Phone161-555-4024	7890 1234			
6	05580CMoore, Adeline	Home Phone221-555-7598	5678 9012			
7	05580DMoore, Adeline	Cell Phone138-555-2410	3456 7890			
8	06075ADubree, Dustinð.....9229 Delegate's Row	Indianapolis	INUSA		
9	06927ABuchs, Jillian	...ræ.....41 Avendale Drive	Carlisle	PAUSA		
10	07008AHouston, Roger	...Ñ@.....4111 Northside Pkwy	Atlanta	GAUSA		
11	07025AMarx, Audrey	...á.....90 South Cascade	Boulder	COUSA		
12	07025BMarx, Audrey	Cell Phone232-555-7244	1234 5678			
13	07025C Marx, Audrey	Home Phone240-555-4245	9012 3456			
14	07025DMarx, Audrey	Work Phone232-555-8753	7890 1234			
15	11204ANeiss, Luke	..é.%....5166 Sprinkle Road	Portage	MIUSA		
16	11204BNess, Luke	Work Phone552-555-2975	5678 9012			
17	11204CNess, Luke	Cell Phone558-555-1854	3456 7890			
18	11544AGraf, Polly	...à.....1551 S. Washington	Dowagiac	MIUSA		
19	11544BGraf, Polly	Home Phone613-555-0518	1234 5678			

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

The screenshot shows the DB2 command line interface with a table layout showing 175 records. Below the table, a dropdown menu is open under 'View Mode'. The menu items are: Single Mode, Dump Mode (which is highlighted with a blue background), DB2 Single Mode, Structure Mode, and Unstructured Mode. A red arrow points from the 'Dump Mode' item to the 'View Mode' dropdown button.

Field	Picture	Type	Start	Length	Data
CUST-ID	X(5)	AN	1	5	05580
REC-TYPE	X	AN	6	1	D
NAME	X(17)	AN	7	17	Moore, Adeline
DESCRIPTION	X(10)	AN	24	10	Cell Phone
CONTACT-INFO	X(20)	AN	34	20	138-555-2410
LAB-DATA-3	X(05)	AN	54	5	3456
LAB-DATA-4	X(05)	AN	59	5	7890

The screenshot shows the DB2 command line interface with a dump mode output. The top part shows the same table layout as before. Below it, a hex dump of the data is shown. The dump includes offsets, byte values in hex and ASCII, and column headers for each byte position.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2
+0	F0	F5	F5	F8	F0	C4	D4	96	96	99	85	6B	40	C	0	5	5
+10	93	89	95	85	40	40	40	C3	85	93	93	40	D7	8	1	i	n
+20	85	F1	F3	F8	60	F5	F5	60	F2	F4	F1	F0	4	e	1	3	
+30	40	40	40	40	F3	F4	F5	F6	40	F7	F8	F9	F				

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

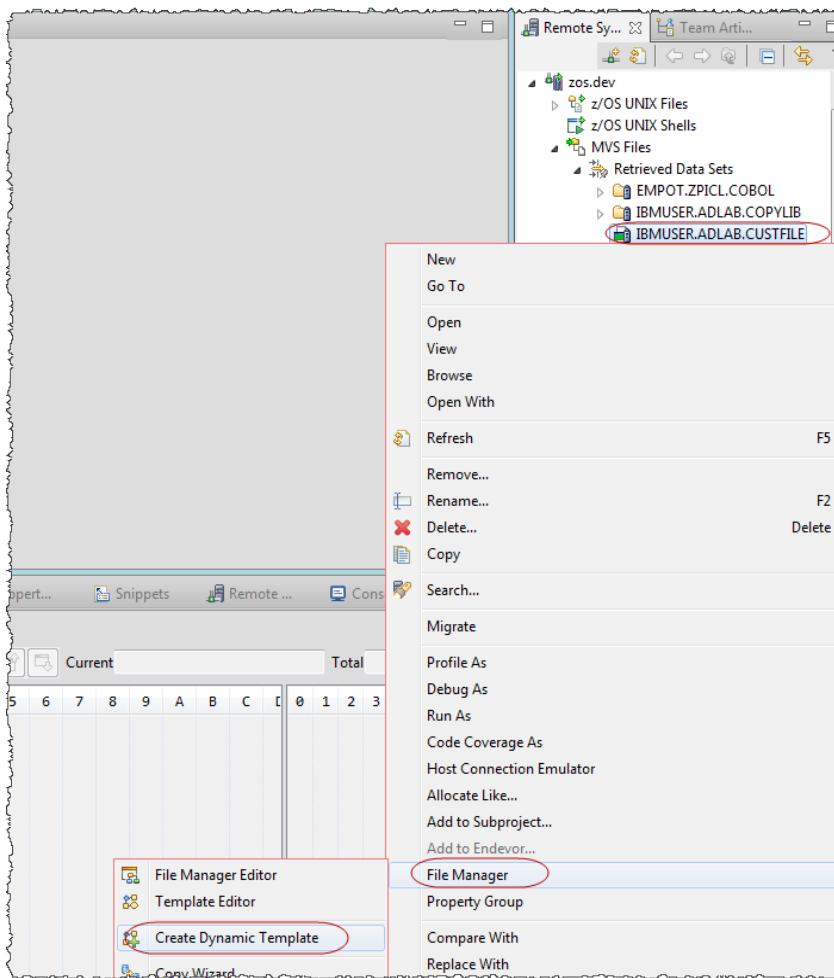
When you use the File Manager editor or viewer, you can specify a layout in the form of either a copybook or a template. If you specify a copybook, the editor/viewer can display records formatted according to the fields in the copybook. While a copybook defines the record layout, it cannot be used to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

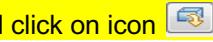
You can use an existing copybook as the basis for a new template. All of the fields in the copybook are copied into the new template, and then you save the template. Templates are stored in PDS or library data sets. After you have saved a template, you can re-use it again whenever you need it. Templates 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.

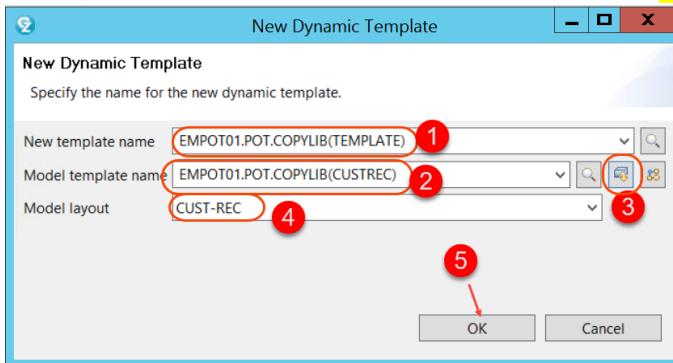
8.4.1 ► Go back to **z/OS Projects** Perspective and using *Remote Systems Explorer* view, right click **IBMUSER.ADLAB.CUSTFILE** file and select **File Manager -> Create Dynamic Template**



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**.

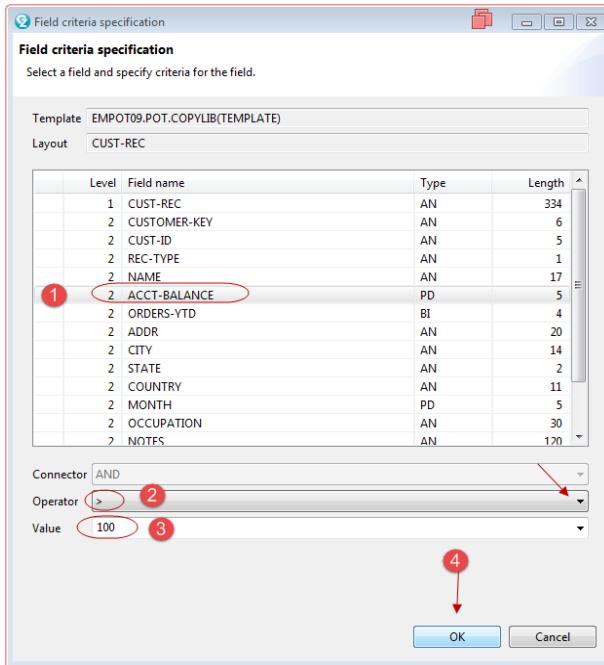
► 1 On **Selection criteria** click on the icon .

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

Sequence	Field name	Type	Length
1	CUST-REC	AN	334
2	CUSTOMER-KEY	AN	6
2	CUST-JD...	AN	5

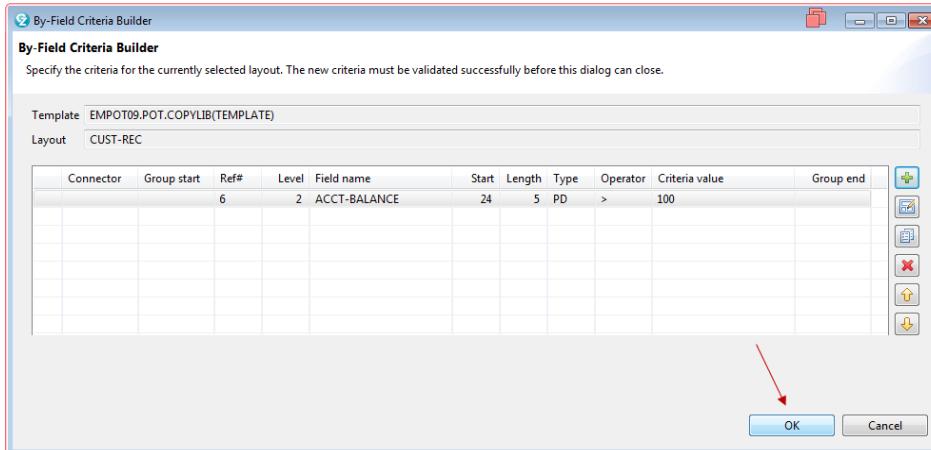
8.4.4 ►| ① On the *Field criteria specification* select **ACCT-BALANCE**

►| 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**

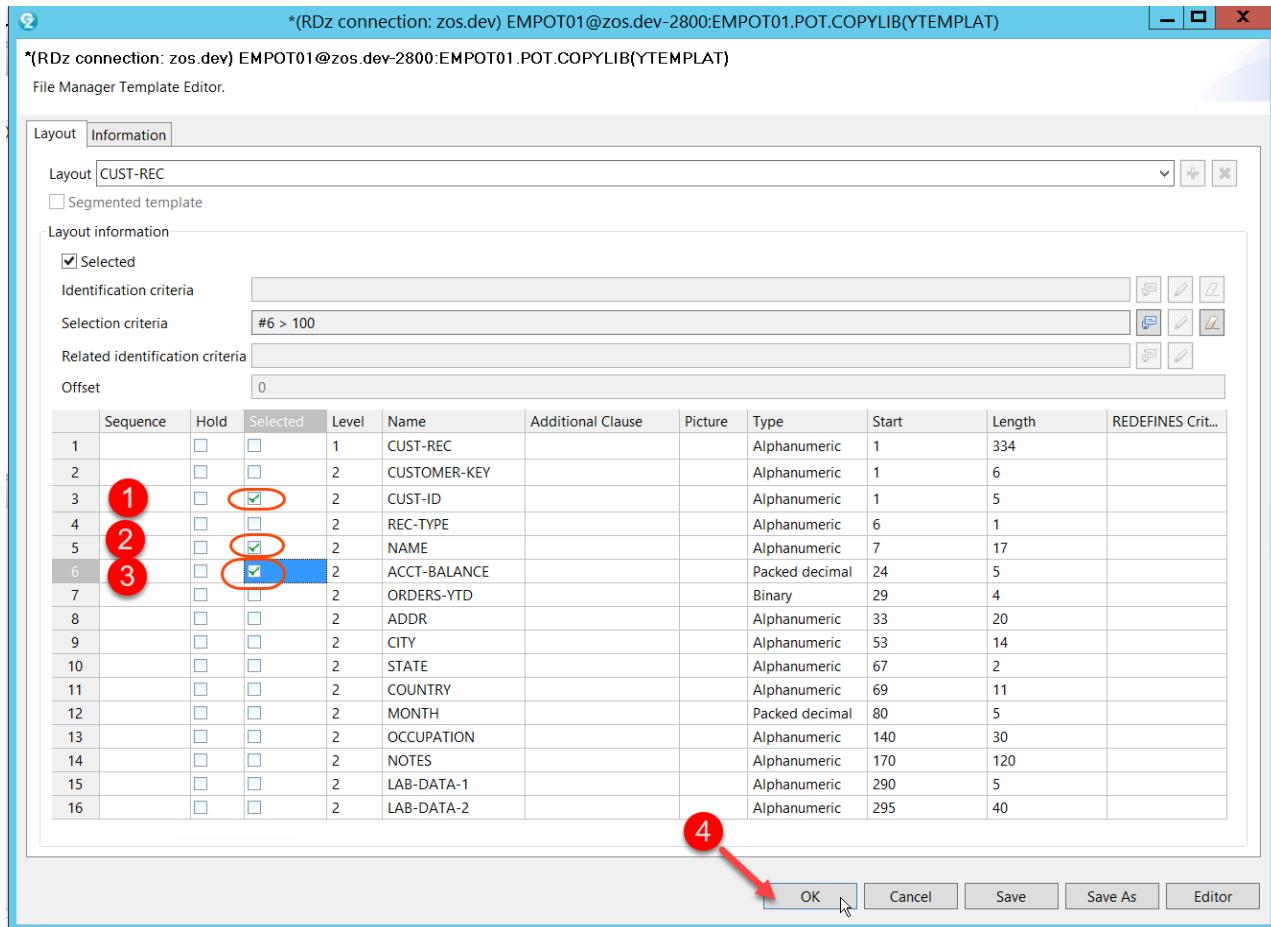


8.4.6 ► Using the column **Selected** double-click on the boxes

① CUST-ID, ② NAME and ③ ACCT-BALANCE.

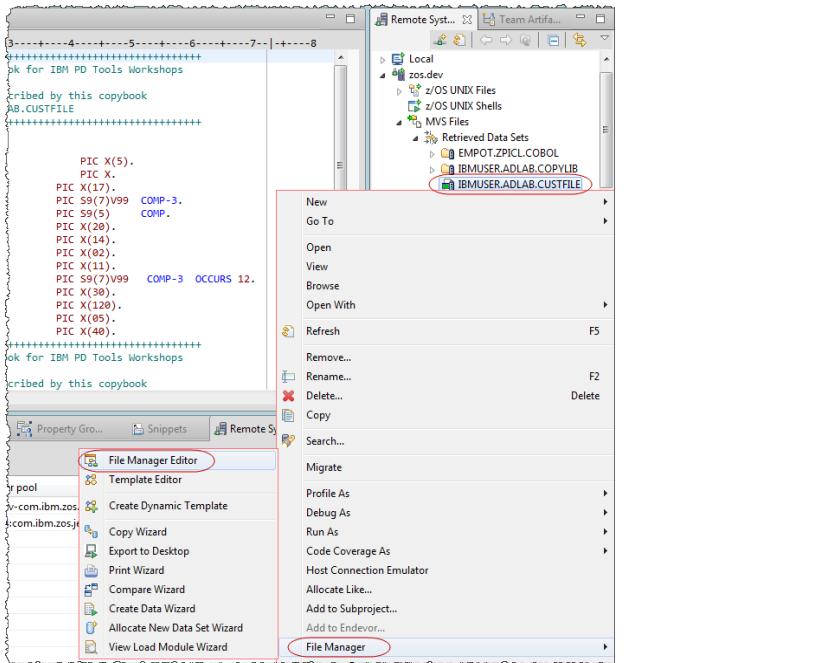
► Click OK.

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



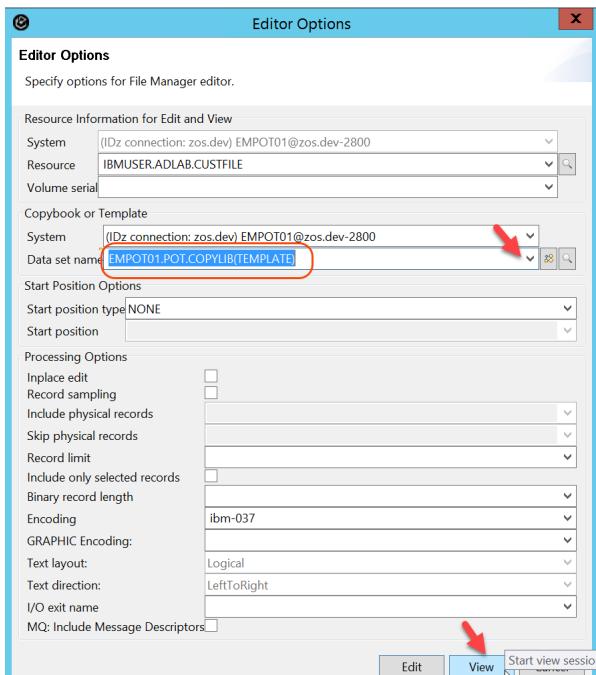
8.4.7 Now that the template is created you can use against the data set.

► 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 View' editor titled '(RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE'. It displays a table with columns CUST-ID, NAME, and ACCT-BALANCE. The data includes rows for customers 1 through 14, with names like Graham, Holly, Moore, Adeline, etc., and account balances ranging from 19298 to 25453. Below this editor is a 'Layout CUST-R' editor showing the same data in a different format. On the right, there is a 'File System' view titled 'zos.dev' which lists various z/OS UNIX files, MVS datasets, and temporary files. The 'zos.dev' node is expanded to show 'z/OS UNIX Files', 'z/OS UNIX Shells', and 'MVS Files' sections.

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

Updated December 16 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.



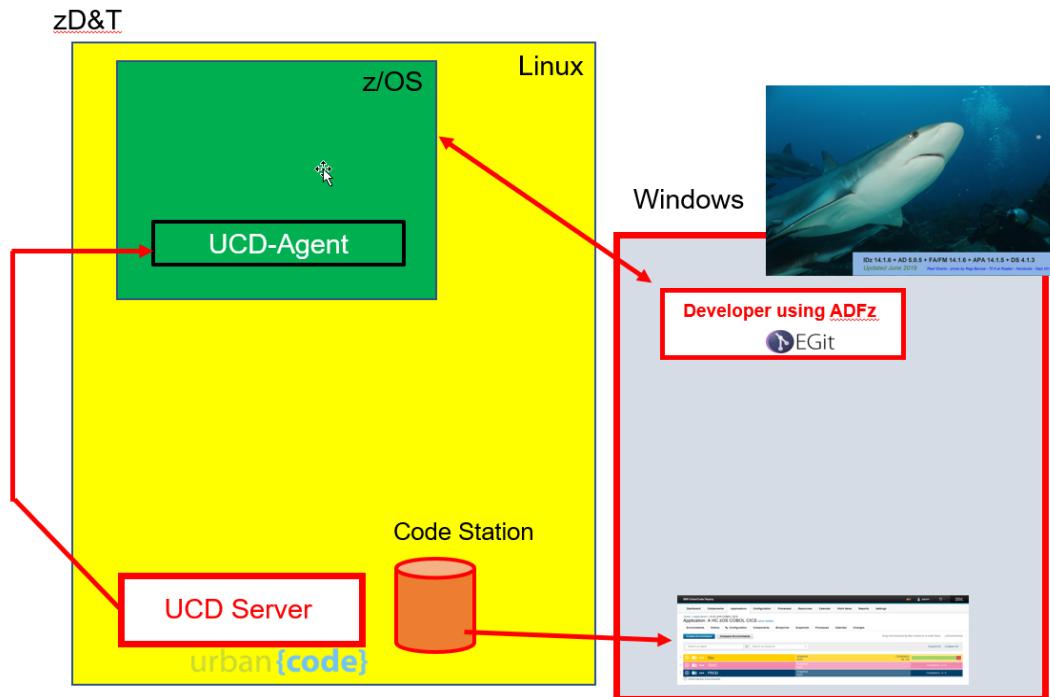
Each time you see the symbol it means that you have to "do" something on your computer – not merely read the document.

Lab Architecture and proposed scenario

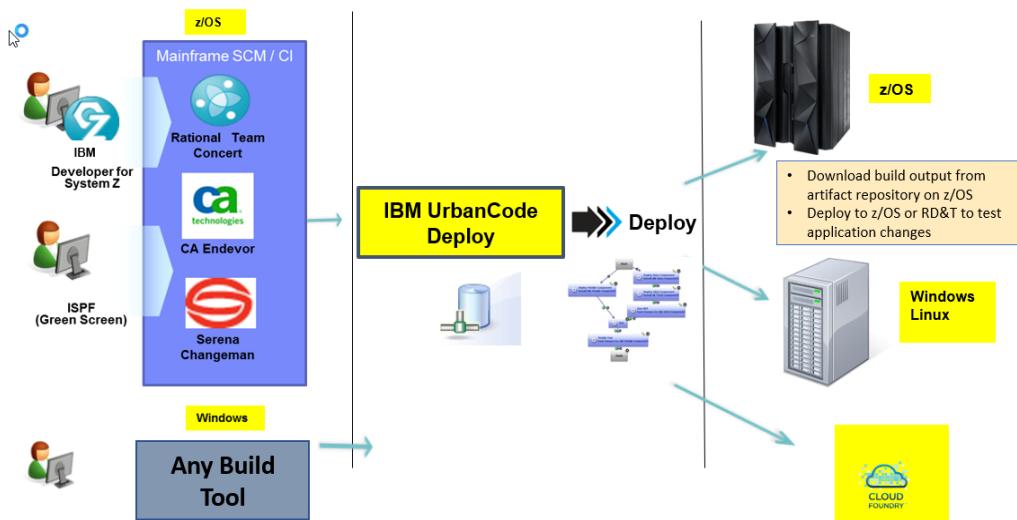
The architecture used in this lab consists of.

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:

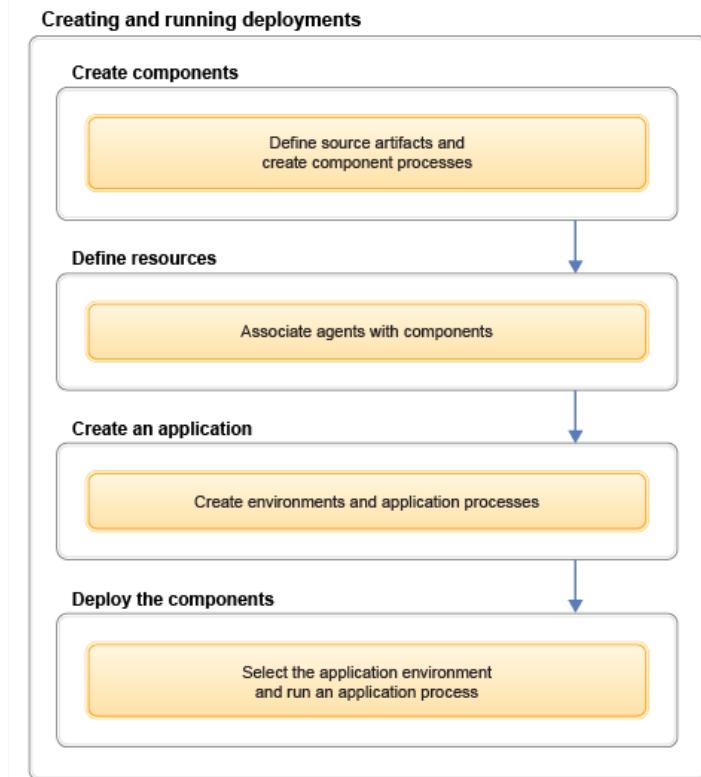


PART 1 – Create the UrbanCode application infrastructure

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

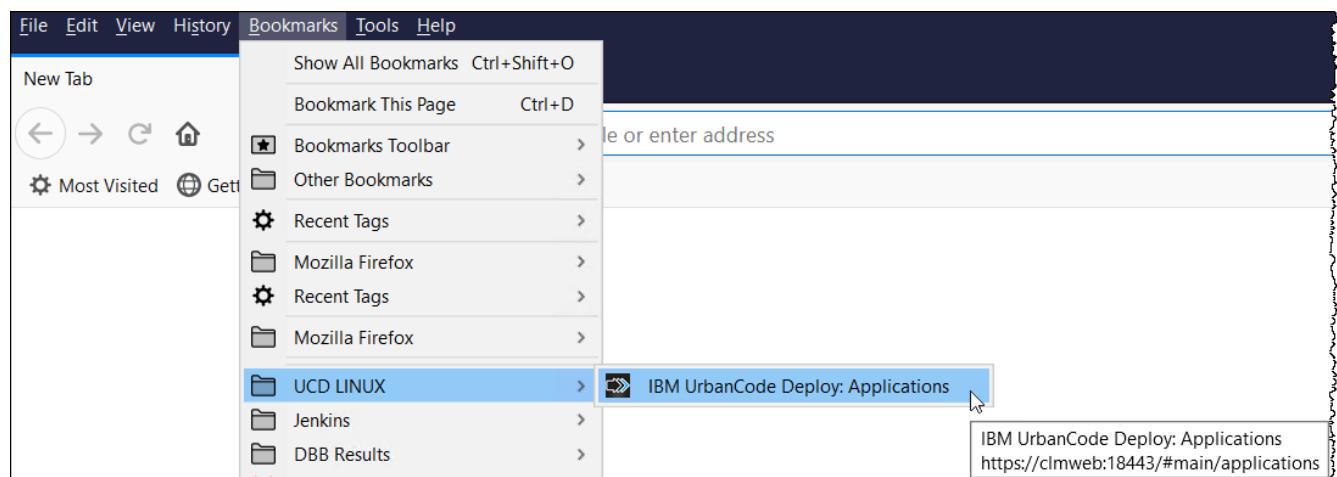
You will now connect to the UrbanCode Deploy (UCD) server running on a LINUX (same Linux that ZDT is running) and that each student have access to it
Here you will verify the UCD agents that the UCD server has access to.

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



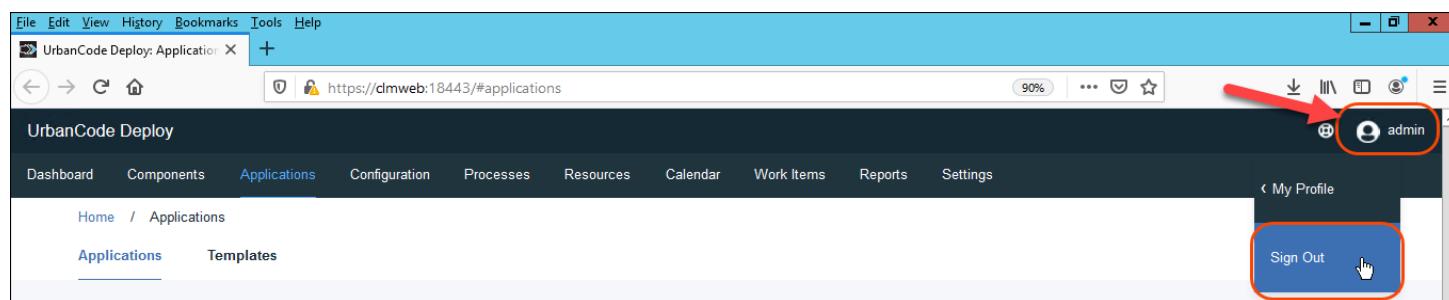
► Look for the **bookmark** below



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



- 1.4. ➡️ Verify that the UCD agents are running 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**)

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		● Upgrade Recommended	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	

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:

The screenshot shows the UrbanCode Deploy interface. At the top, there's a navigation bar with tabs: Dashboard, Components (which is highlighted with a red circle and arrow), Applications, Processes, Resources, Calendar, Work Items, Reports, and Settings. Below the navigation bar, the URL is shown as Home / Components. Under Components, there are two sub-tabs: Components (which is highlighted with a blue underline) and Templates. The main area is titled 'Components' and contains a table with columns: Name, Latest, Latest Version, Template, Description, Created, and By. There are filters at the top of the table: Flat list, Actions..., Import Components, and Create Component. The 'Create Component' button is highlighted with a red circle and arrow. The entire screenshot is framed by a thick black border.

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

1. Name: J02Mortgage

2. Component Template: MVSCOMPONENT

3. Description: (empty)

4. Save button

Additional configuration options shown below:

- 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

2.3 The component **J02Mortgage** is created

Dashboard Components Applications Processes Resources Calendar Work Items Reports Settings

Home / Components / J02Mortgage

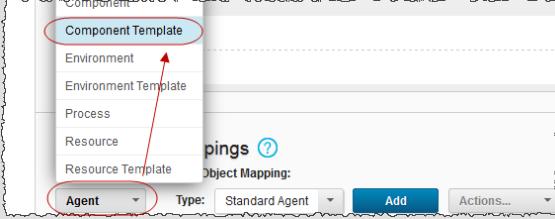
Component: J02Mortgage [Show details](#)

Dashboard Usage Configuration Calendar Versions Processes Changes

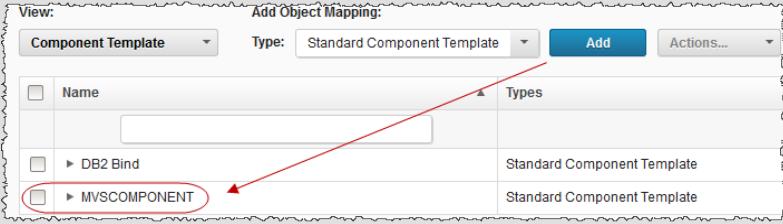
The MVSCOMPONENT is not in the drop down?
 Your UCD server must have this template defined there. If NOT call the instructor..

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

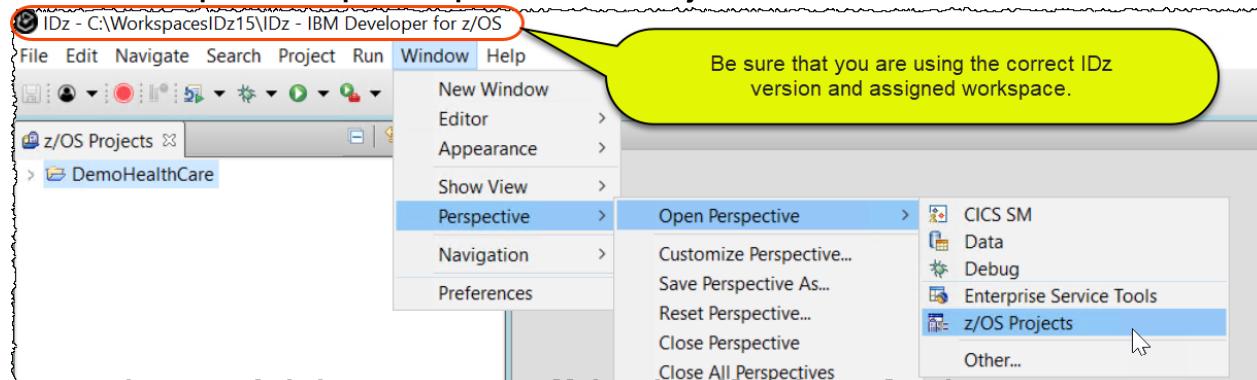
►► Using the desktop double click on **IDz V15** icon.

►► 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.

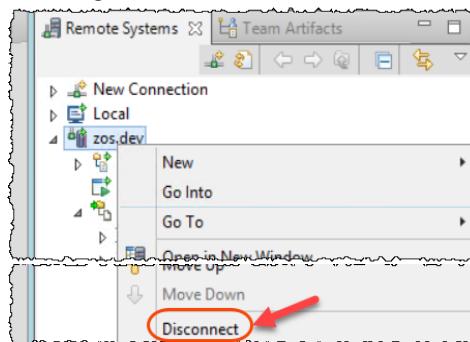


3.2 ► Open the z/OS Projects perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

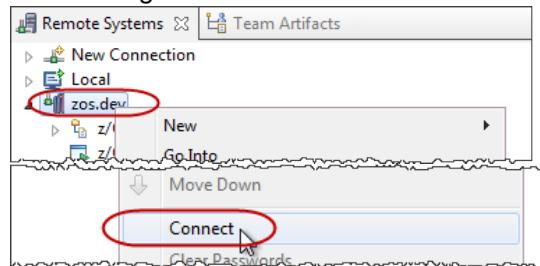


3.3 Before you used `empot01`, `empot05` or `ibmuser` and you need to disconnect. Your new userid now will be **empot02**.

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

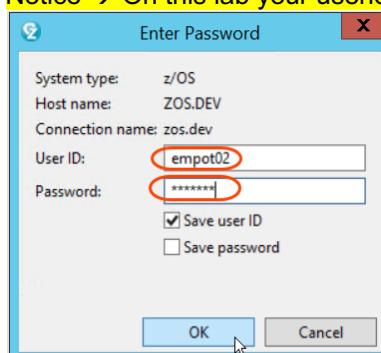


3.4 ► Right click on **zos.dev** and select **Connect**

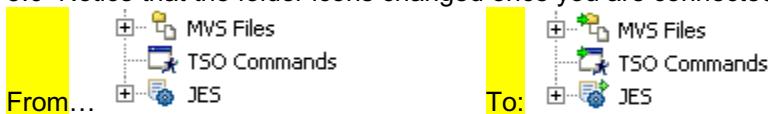


3.5 ► Type **empot02** as userid and **empot02** as password.
Click **OK** to connect to z/OS.

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



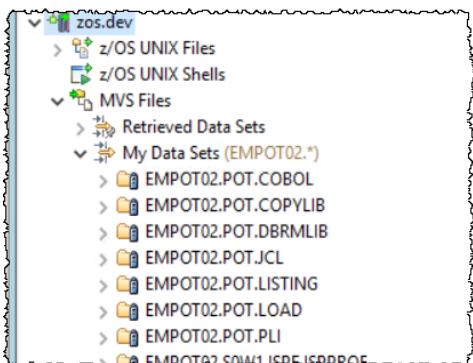
3.6 Notice that the folder icons changed once you are connected to z/OS. A small green arrow is added.,,



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. .

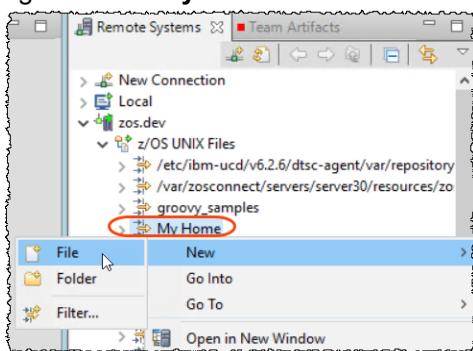
What is the Ship list file?

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.

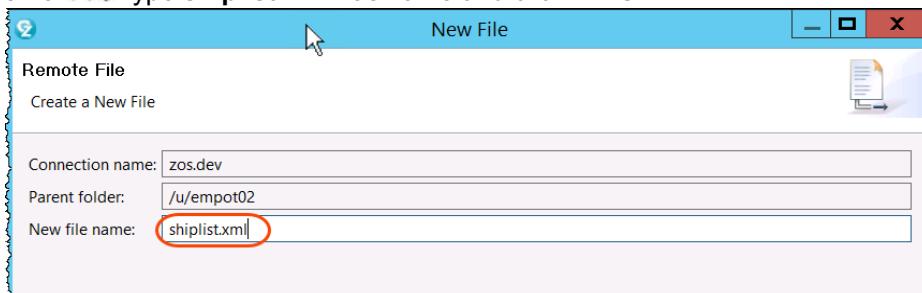


Ship list files are XML files that contain a list of files. The container type that is used to identify partitioned data set (PDS) resources is PDS and the resource type is *PDSmember*. You can use an asterisk (*) as a wildcard for the resource name, if you want all members in a partitioned data set (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.

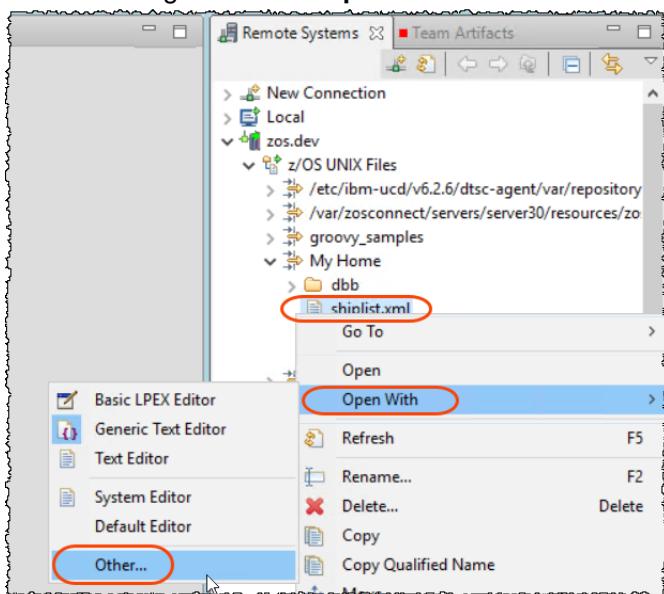
3.9 ►| Using the *Remote Systems* view (on top right), expand **z/OS UNIX Files**, right click on **My Home** → **New** → **File**



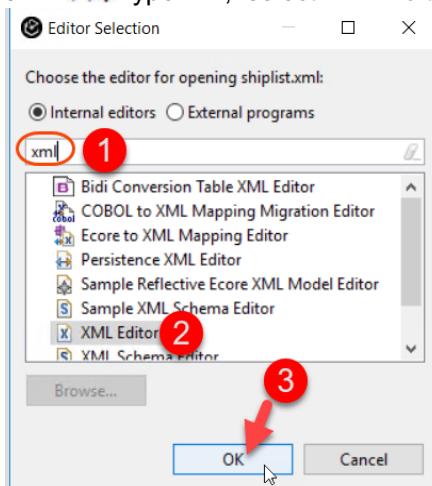
3.10 ► Type **shiplist.xml** as name and click **Finish**



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**.



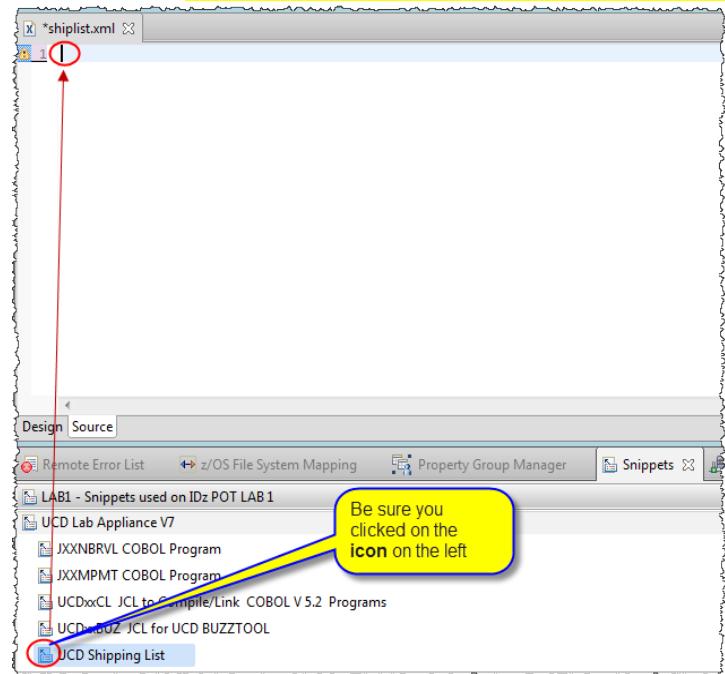
3.13 ► Click on **Source** tab



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

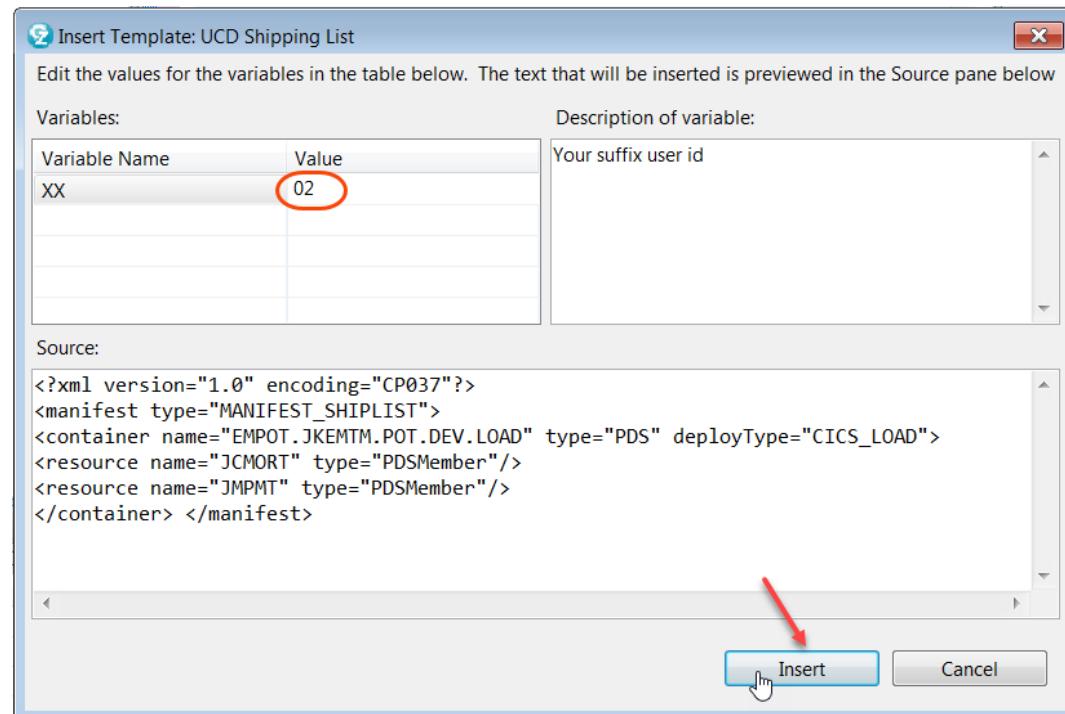
► Scroll down and use the **UCD Shipping List** snippets to drag and drop to the **FIRST position** of the file

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

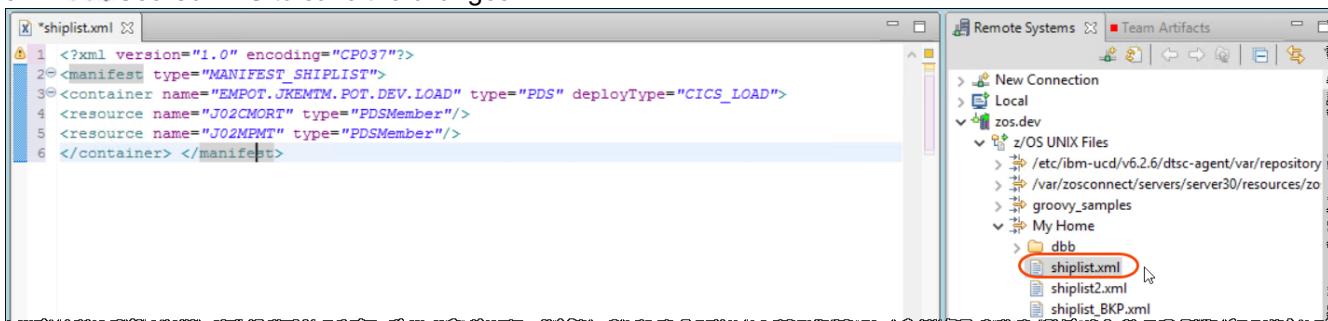


3.15 ► When the dialog opens type **02** ...

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



3.17 ➡ Use Ctrl + S to save the changes



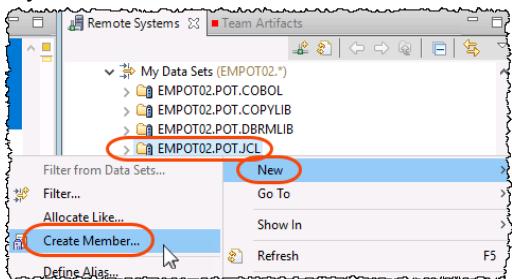
Task 4 - Create the UCD component version from JCL

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).

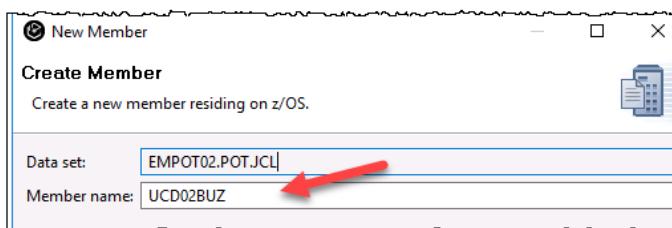
4.1 ➡ Using Remote System view, expand **My Data Sets**, right click on **EMPOT02.POT.JCL** and create a member .

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

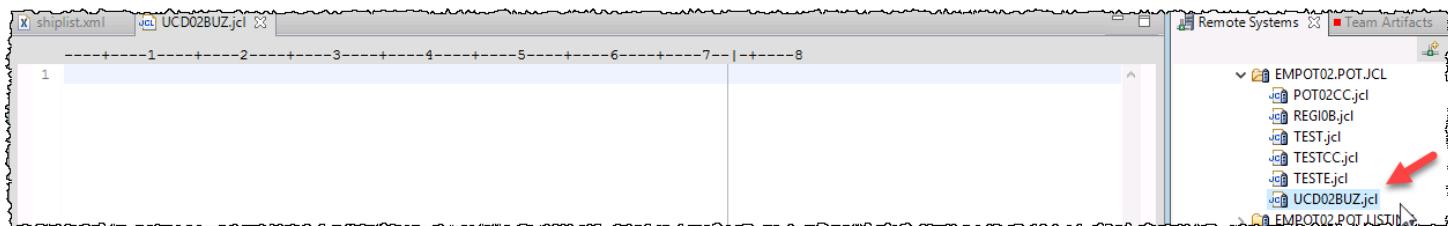


4.2 ➡ Name it **UCD02BUZ** and click **Finish**

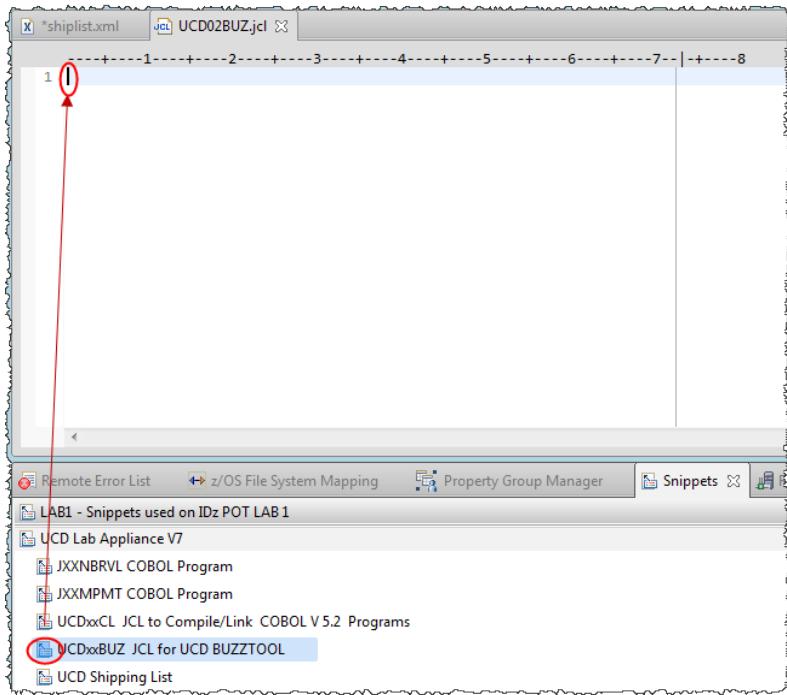
NOTE: If you have already this member there, right click and delete it.



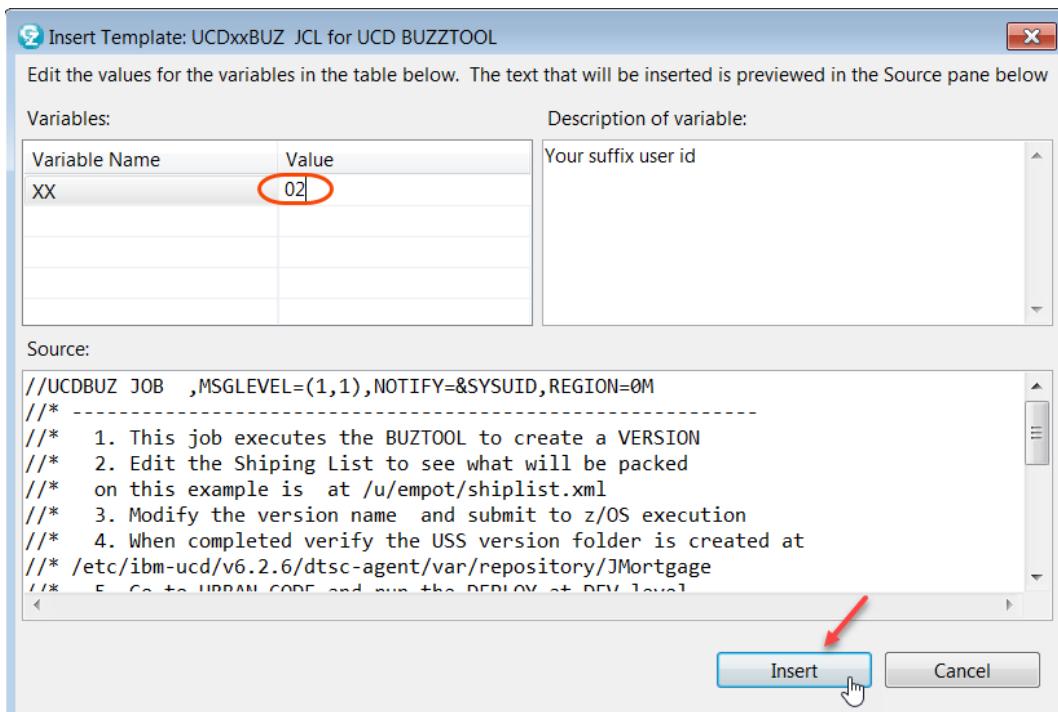
4.3 ➡ Double click on **UCD02BUZ.jcl** to edit.



- 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

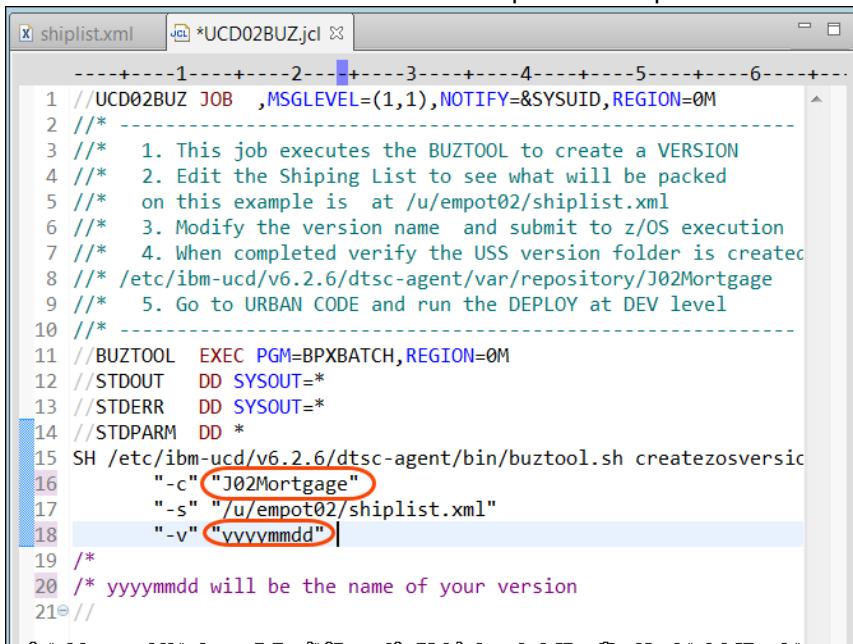


- 4.5 ►| When the dialog opens type **02** ..
 ►| Click **Insert** to insert the lines. Note that the variables are replaced



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

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



```

x shiplist.xml  *UCD02BUZ.jcl x
-----+-----2-----3-----4-----5-----6-----+
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" "vvvvmmdd"
19 /*
20 /* yyyymmdd will be the name of your version
21 */

```

Possible arguments for the *buztool createzosversion* command

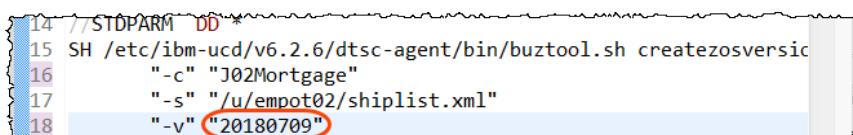


Parameter	Required	Description
-c	true	The name of the component in IBM UrbanCode Deploy. The component name can contain only letters, numbers, and spaces.
-v	false	The name of the version to create. If a version is not specified, a version name is generated from the current time stamp. The version name 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.



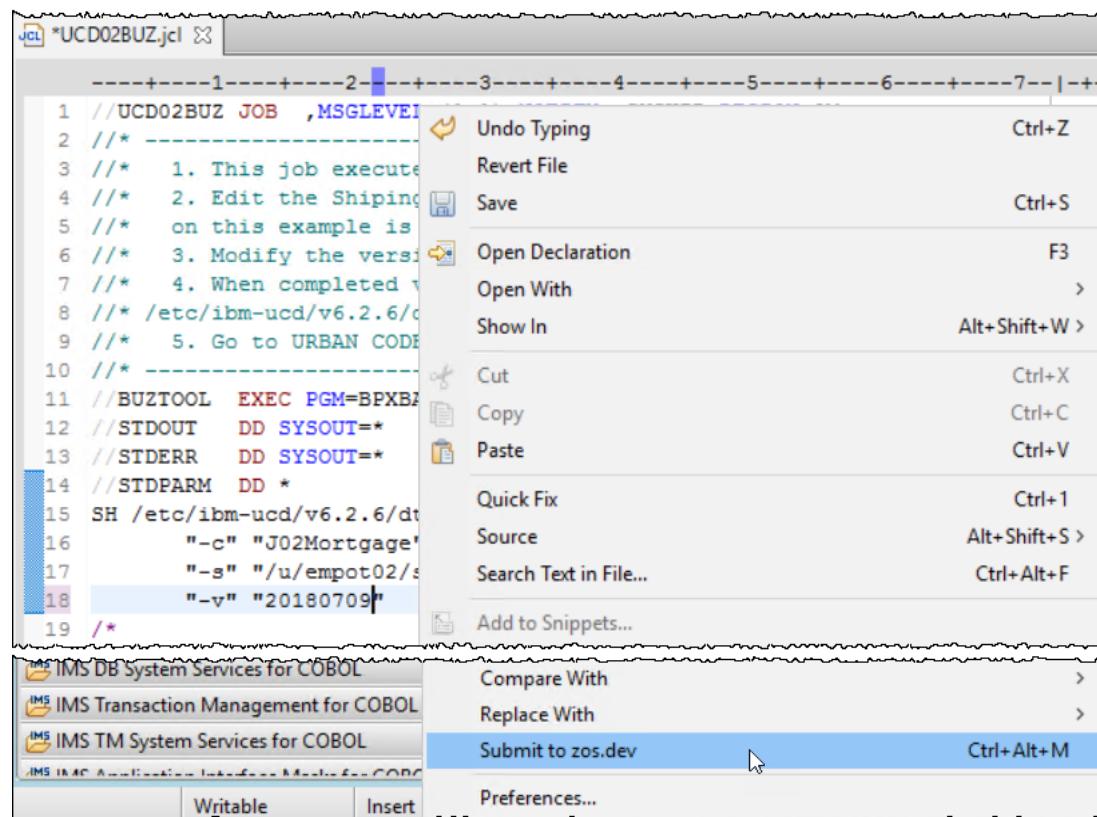
```

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**

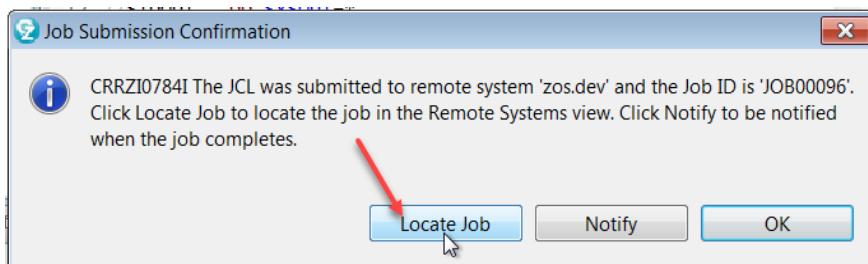


You do not find the option Submit?...



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..

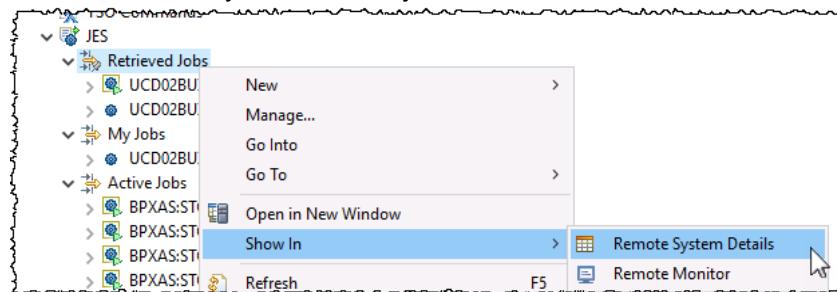
4.9 ►► Click **Locate Job** to get the job results



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 > Remote System Details**.

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*

Remote system filter Retrieved Jobs										
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

```

53 CPU: 0 HR 00 MIN 00.08 SEC SRB: 0 HR 00 MIN 00.00 SEC
54 SH /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh createzosversion
55 "-c" "J02Mortgage"
56 "-s" "/u/empot02/shiplist.xml"
57 "-v" "20180709"
58 zOS toolkit config : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,20170906-2200)
59 zOS toolkit binary : /etc/ibm-ucd/v6.2.6/dtsc-agent/ (6.2.6,20170906-2200)
60 zOS toolkit data set : BUZ626 (6.2.6,20170907-0249)
61 Reading parameters:
62 ....Command : createzosversion
63 ....Component : J02Mortgage
64 ....Version : 20180709
65 ....Shiplist file : /u/empot02/shiplist.xml
66 Verifying version
67 ....Repository location : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage/20180709
68 Pre-processing shiplist:
69 ....Shiplist after processing :/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage/20180709/shiplist.xml
70 Packaging data sets:
71 ....Location to store zip : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage/20180709
72 ....Zip name : package.zip
73 ....EMPTO.JKEMITM.POT.DEV.LOAD.bin
74 ....Elapsed time for data set package or deploy operation : 3.935909
75 Create version and store package:
76 ....Version artifacts stored to UCD server CodeStation
77 ....Version:20180709 created
78 Elapsed time 39.0 seconds.

```

- 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.

```

КАМУТ
ISPSTART CMD(BUZTOOL "createzosversion" "-c" "J09Mortgage" "-v" "20150601" "-s" "/u/empot09/shiplist.xml"
Component:J09Mortgage
Version to create:20150601
Ship list file:/u/empot09/shiplist.xml
UCD SERVER URL:https://clmweb:8443
The value given for 'component' (J09Mortgage) could not be resolved.
Component not found on server.

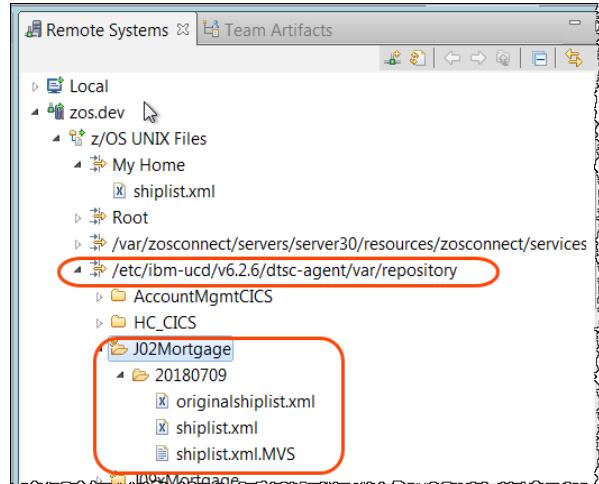
```

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**

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, 7:30 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	MVSCOMPONENT		7/9/2018, 6:11 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: 5/7/2020 4:05 PM
Repository Type: CodeStation

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):

Artifacts

Total add: 2 members in 1 data sets

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.

Save

You should have:

Resource Tree					
			Inventory	Status	Description
..	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

Name*

Description

Inherit Teams From Parent

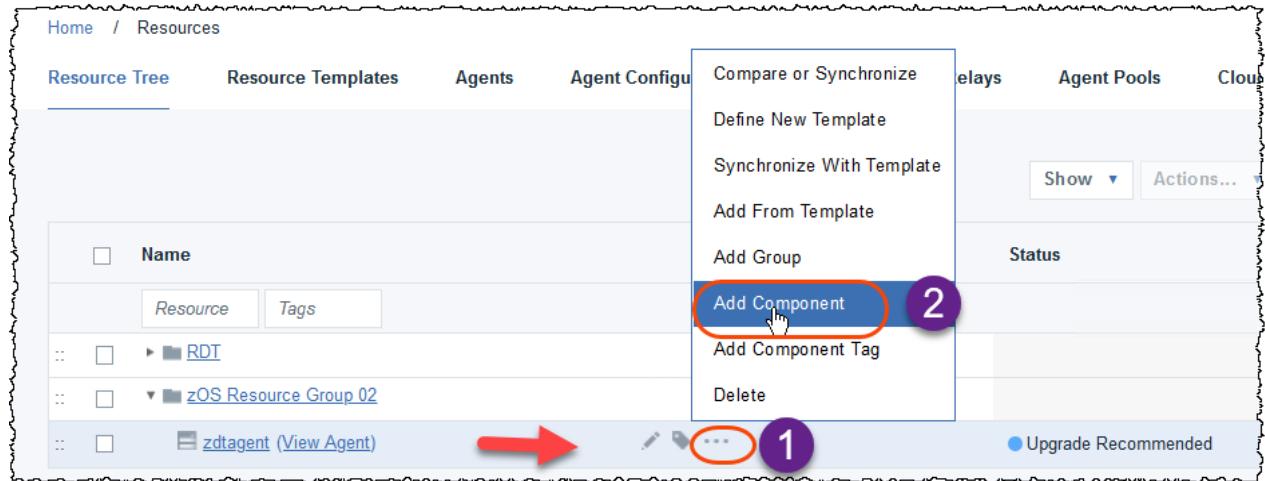
Teams

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**

- 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*

Name *

Description

Inherit Teams From Parent

Teams

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 Home, Resources, Resource Templates, Agents, Agent Configuration Templates, Agent Relays, Agent Pools, and Cloud. Below the tabs is a search bar with 'Name' and 'Filter' buttons. The main area displays a tree view of resources. A red box highlights the 'zOS Resource Group 02' node, which has two children: 'zdtagent (View Agent)' and 'J02Mortgage (View Component)'. To the right of the tree, there is a status indicator for 'Upgrade Recommended'.

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 that 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}

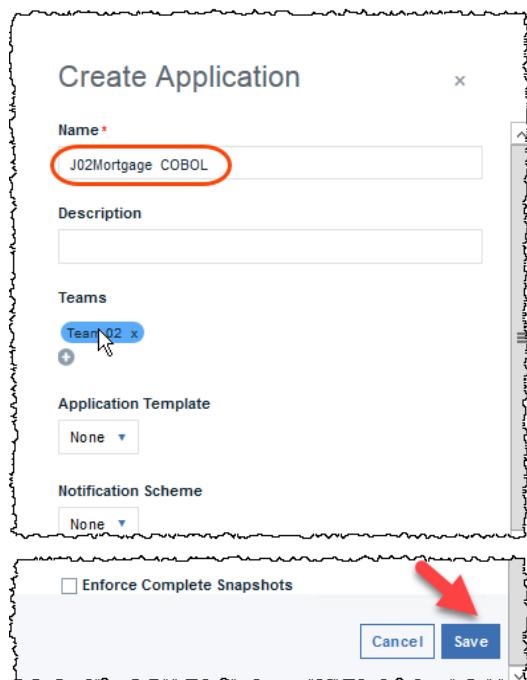
You define objects (Application, component, process, resource, etc) properties in the object's Configuration Tab of UCD browser interface.

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 of the UrbanCode Deploy interface. The tab bar includes Dashboard, Components, Applications (which is highlighted), Processes, Resources, Calendar, Work Items, Reports, and Settings. Below the tab bar, there is a navigation bar with Home / Applications, Applications (which is highlighted), and Templates. At the bottom of the screen, there is a toolbar with 'Flat list', 'Actions...', 'Import Applications', and a large blue 'Create Application' button. A purple circle labeled '1' is positioned over the 'Applications' tab, and a purple circle labeled '2' with a red arrow points to the 'Create Application' button.

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



The result should be:

Task 7 - Create environment

On this task you will you 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?

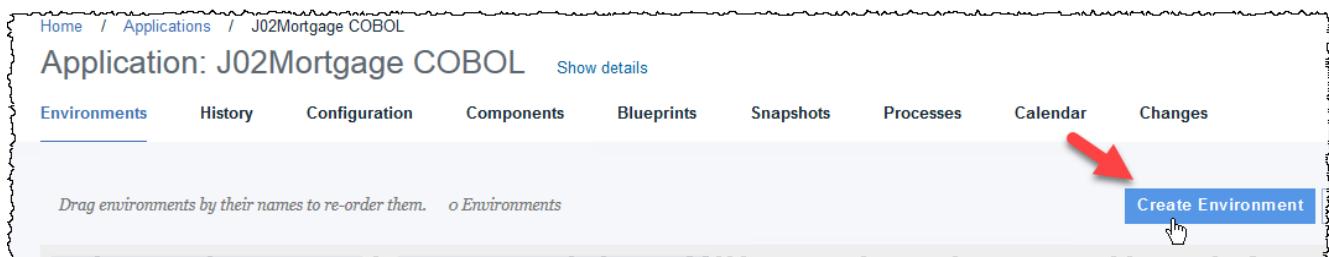


An environment is the application's mechanism for bringing together components with the agent that deploys them. Environments are typically modeled on some stage of the software project lifecycle, such as development, test, or production.

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**:



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

The screenshot shows the 'Create Environment' dialog box. Step 1 highlights the 'Name' field with the value 'Test'. Step 2 highlights the color picker with a yellow circle. Step 3 highlights the 'Save' button. The dialog includes fields for 'Description', 'Blueprint', 'Teams', and checkboxes for 'Require Approvals', 'Exempt Processes', 'Lock Snapshots', and 'Require Snapshot'. A color palette is also present.

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.' and '1 Environment'. A search bar at the top has 'Search by Name' and 'Search by Blueprint'. A blue button on the right says 'Create Environment'. The main area lists environments with a yellow header row. One environment, 'Test', is highlighted with a yellow background. A red arrow points to the 'Test' entry in the list.

7.4 ► Click **Add Base Resources**:

The screenshot shows the 'Resources' tab selected. At the top, there's a message 'No Desired Inventory'. Below it is a table with columns: Name, Inventory, Status, and Description. A red arrow points to the blue 'Add Base Resources' button in the top right corner of the table area.

7.5 ► Expand **zOS Resource Group 02** and **zdtagent**.

► 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. On the left, there's a tree view showing 'Name' and 'Tags' filters, and a 'Filter' button. The tree view shows 'RDT', 'zOS Resource Group 02' (expanded), 'zdtagent' (expanded), and 'J02Mortgage' (selected with a checked checkbox). A red circle with '1' points to the 'J02Mortgage' checkbox. A red circle with '2' points to the blue 'Save' button at the bottom right. Other buttons include 'Cancel', 'Refresh', and 'Print'.

You must have now:

Name	Inventory	Status	Description
Resource	Filter	Filter	
zOS Resource Group 02 / zdtagent (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 (circled in red). A red arrow points from the text 'Click on tab Components' to this tab. Below the navigation bar, there are two tabs: 'Components' (selected) and 'Templates'. A purple circle with the number '1' is placed over the 'Components' tab. In the main content area, a table lists components. One row for 'J02Mortgage' is highlighted with a yellow background and circled in red. A red arrow points from the text 'find the component created earlier (J02Mortgage)' to this row. A purple circle with the number '2' is placed over the 'J02Mortgage' row.

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 Windows x	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 emp02 (emp02)	

8.2 ► Click the **Processes** tab

The screenshot shows the 'Processes' tab selected in the top navigation bar of the component details page (circled in red). A blue arrow points from the text 'Click the Processes tab' to this tab. Below the navigation bar, there are six tabs: 'Dashboard', 'Usage', 'Configuration', 'Calendar', 'Versions', and 'Processes'. A blue circle with a hand icon is placed over the 'Processes' tab. In the main content area, a table lists various processes. The first process, 'Deploy (MVSCOMPONENT)', is highlighted with a yellow background and circled in red. A red arrow points from the text 'there is a small collection of processes that was created automatically for your component' to this process.

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.

Component: J02Mortgage [Show details](#)

Dashboard Usage Configuration Calendar Versions Processes Changes

Process Description

Create Process

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

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

Create Process

Name* 1
Deploy JKE to CICS

Description

Process Type*
Deployment

Inventory Status*
Active

Default Working Directory*
\${p:resource/work.dir}/\${p:component.name}

Required Role 2

Cancel Save

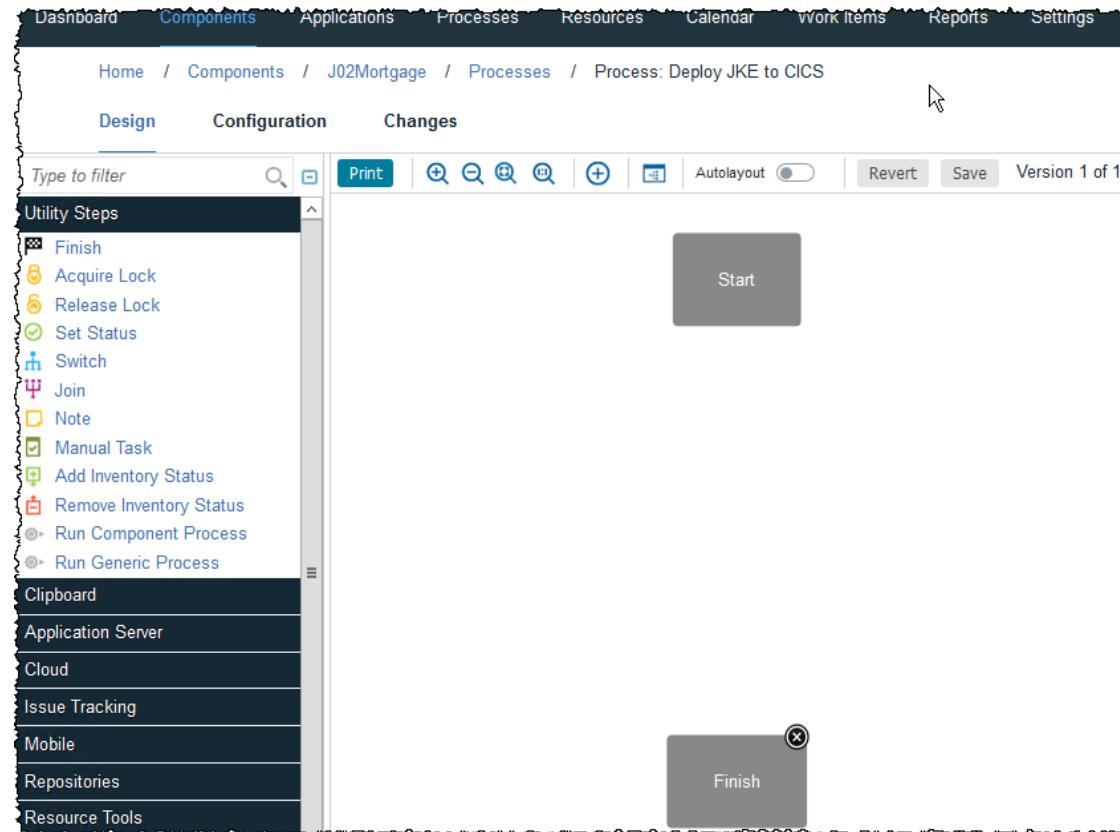
8.5 The Process Editor will open.

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.udeploy.doc/topics/comp_workflow.html

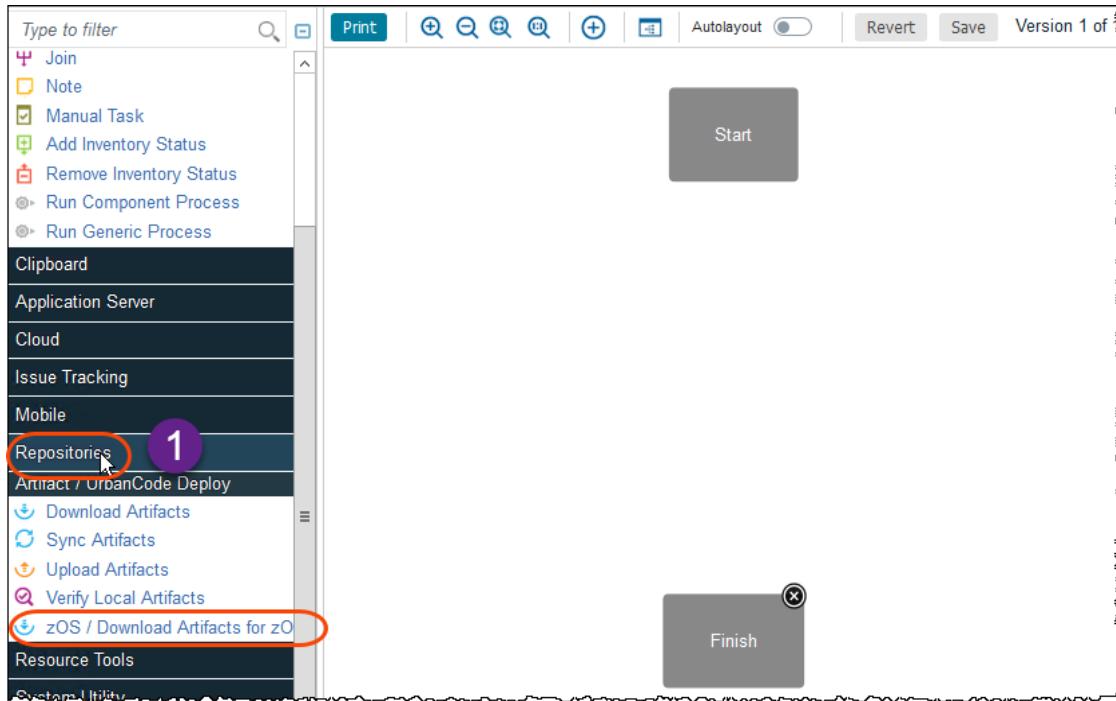


8.6 We are going to create a simple process for updating an existing CICS application.

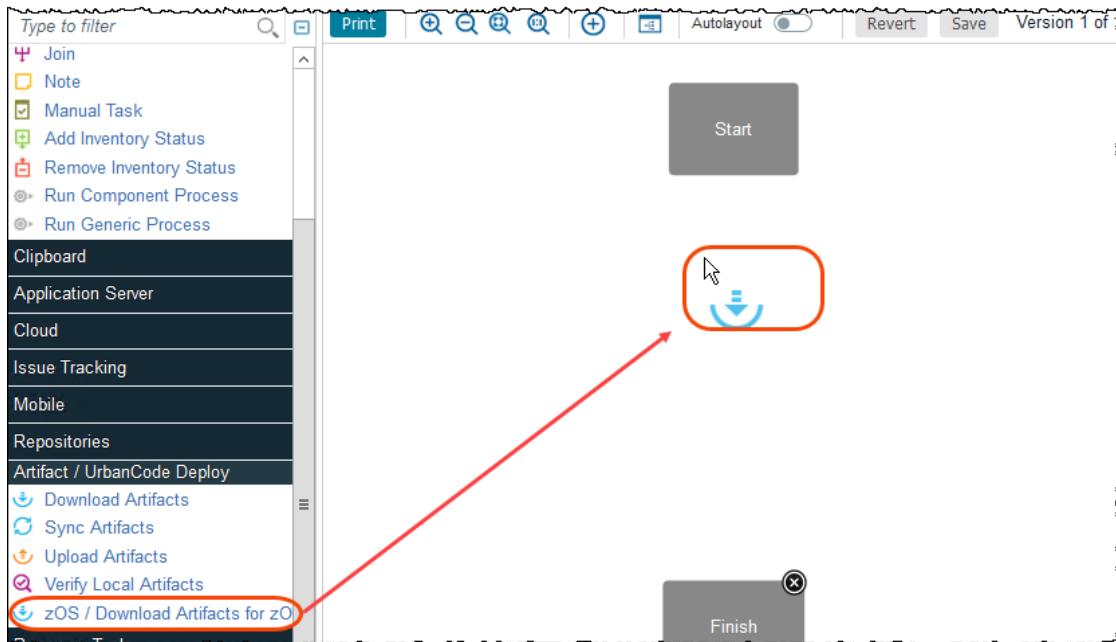
► On the left-hand side of the editor you will find an extensive **Palette** of process steps.

► Scroll down and expand **Repositories**.

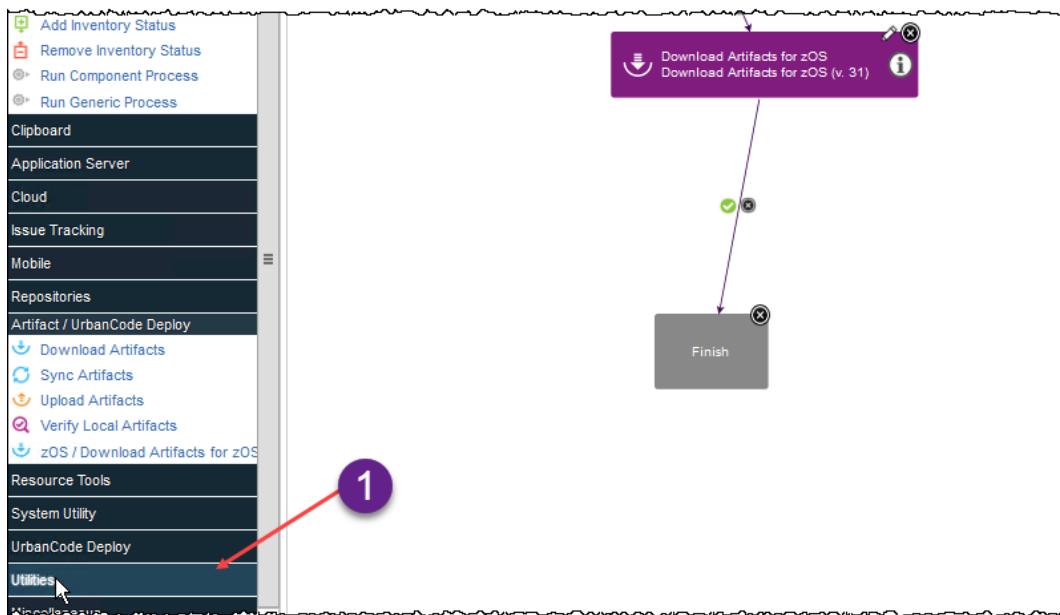
First you are going to use the step **zOS /Download Artifacts for zOS**:



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



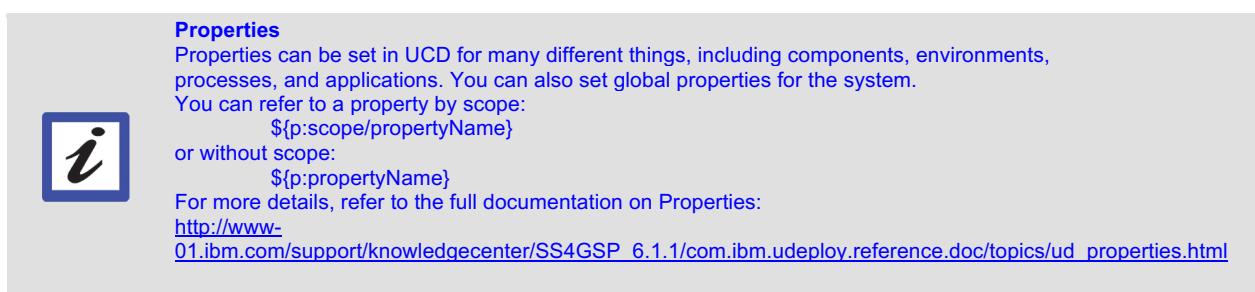
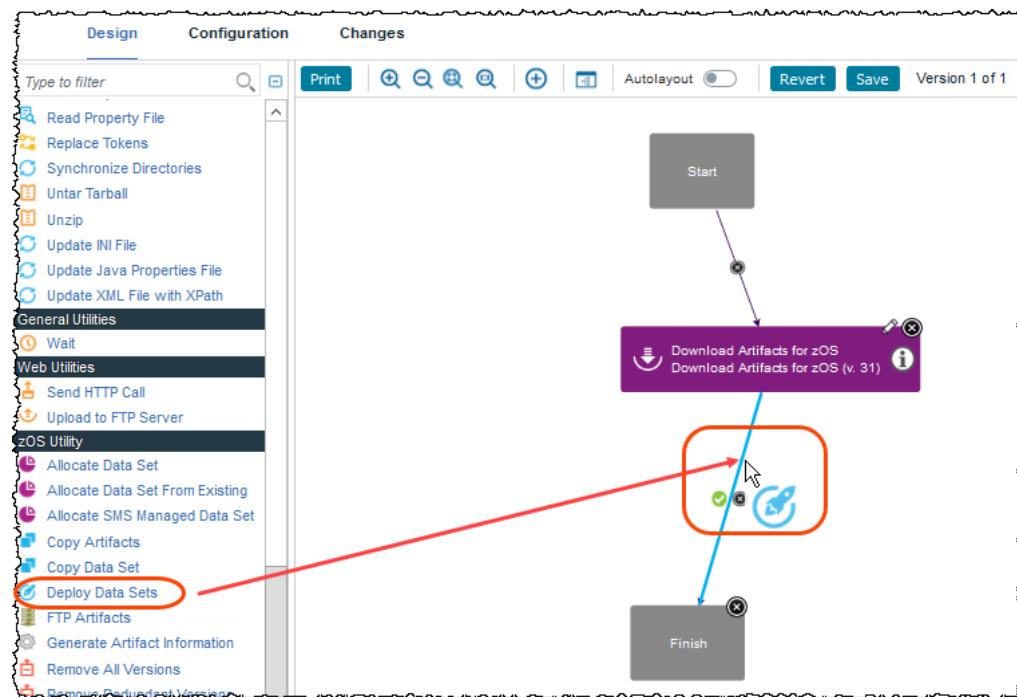
8.8 ► 1 Using the Design palette on left expand **Utilities**, clicking on it



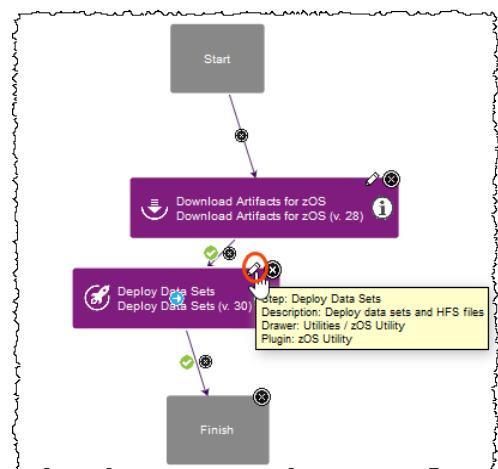
► 2 Scroll down and 3 expand **zOS Utility**: The **Deploy Datasets** step will also be needed to load component artifacts from the z/OS repository and to bring them into z/OS work area for later deploy.



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



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

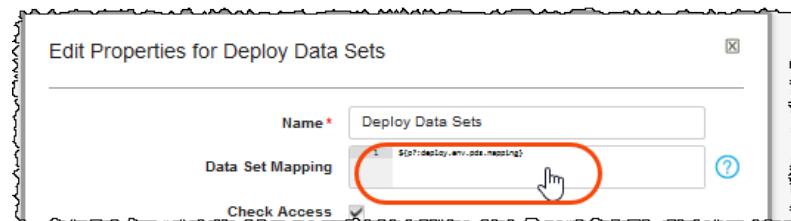


8.11 ► Enter the value for the *Data Set Mapping* property:
EMPOT.JKEMTM.POT.DEV.LOAD, EMPOT05.DEMO.DEV.LOAD

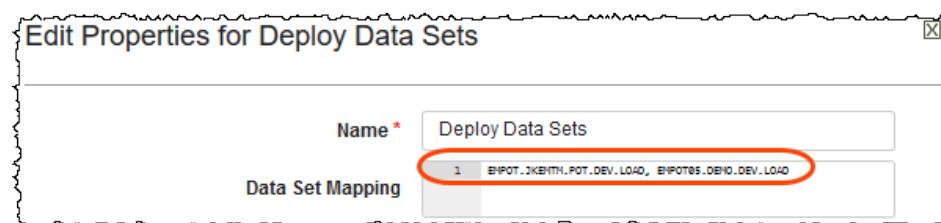
Data Set Mapping specifies which PDS members packaged with the component to deploy, and where to deploy them.

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

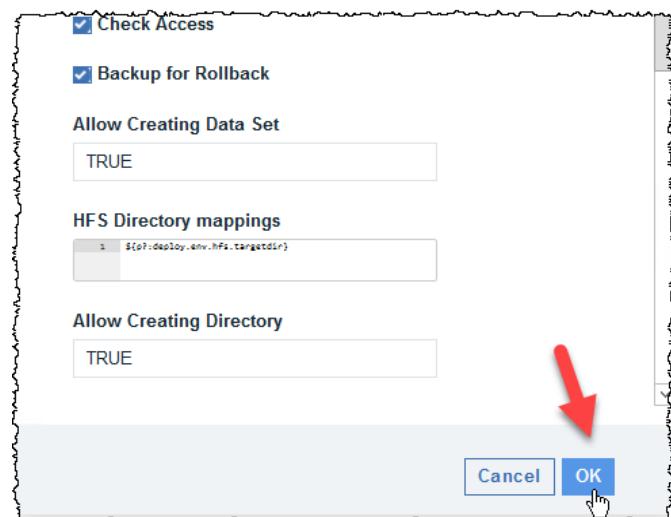
We could have that as variable but we will fix here to save time.



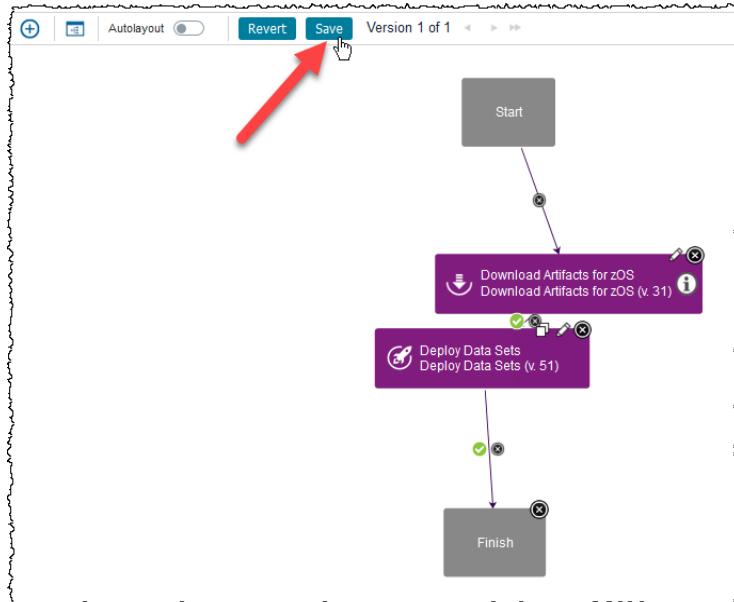
► After entering the values click **Save**



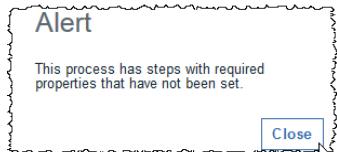
8.12 ► Click **OK** to save the properties updated



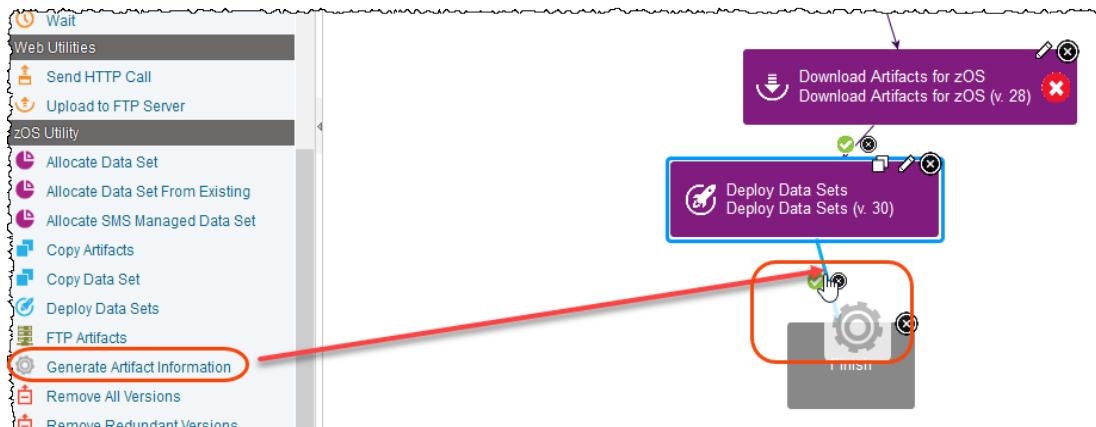
8.13 ► Click the **Save** button to save what you had done so far.



8.14 ► If there is a warning about missing property ignore it, clicking **Close**. You will fix that later.



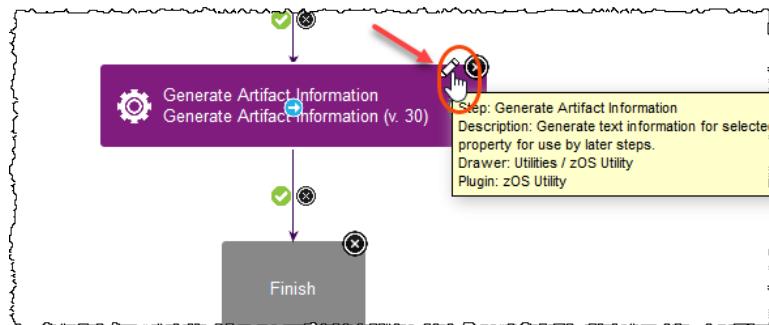
8.15 ► Scroll down on left pallet and drag and drop **Generate Artifact Information** on the line before Finish. You will need to specify a list of the load modules that CICS needs to do the *Newcopy*. This step generate text information for selected version artifacts. Information is put to output property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



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

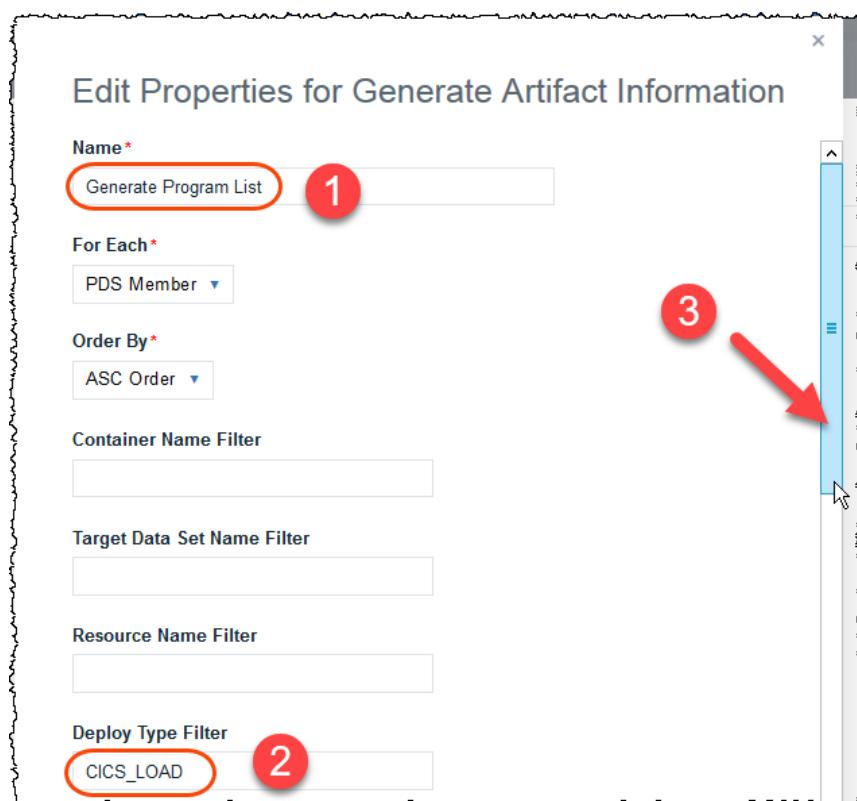


- 8.17 ►| Click on the pencil icon  on **Generate Artifact Information** to change the UCD properties for this step.



- 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*
 1

For Each*

Order By*

Container Name Filter

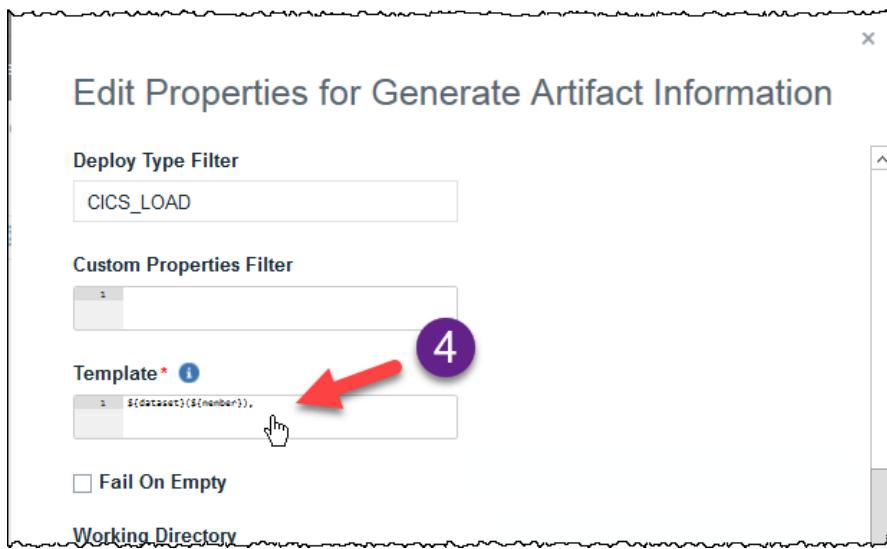
Target Data Set Name Filter

Resource Name Filter

Deploy Type Filter
 2

3

8.19 ► 4 Click on **Template**

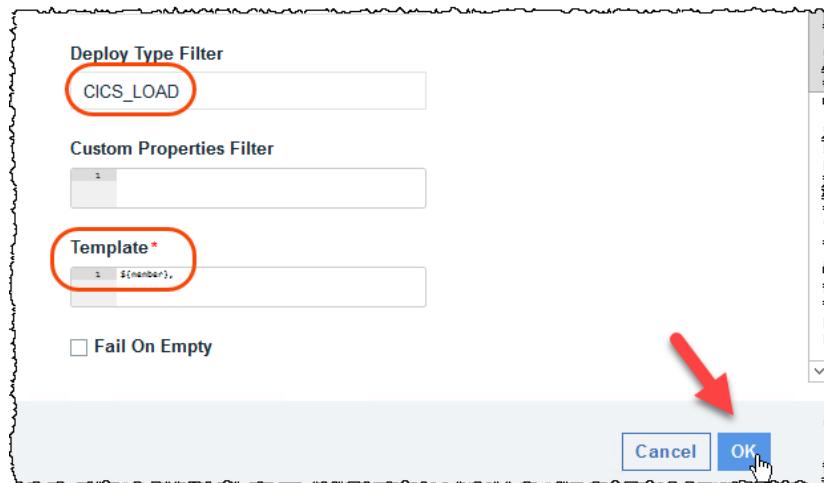


► Change the value from “
 `${dataset}(${member}),`” to `${member},`
IMPORTANT → Be sure that you add a , (comma) after the `${member},`



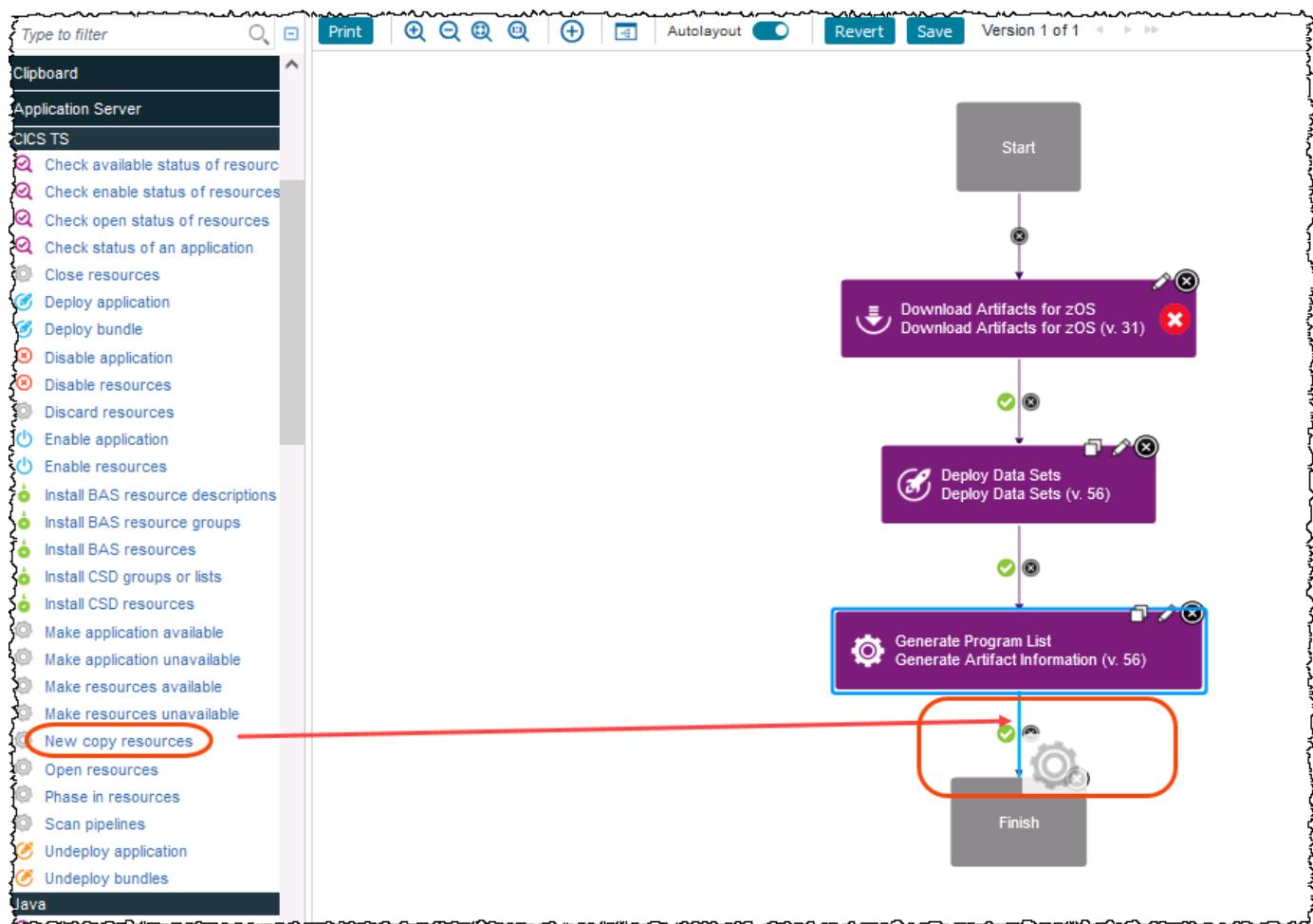
► Click **Save** to save the modified value

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

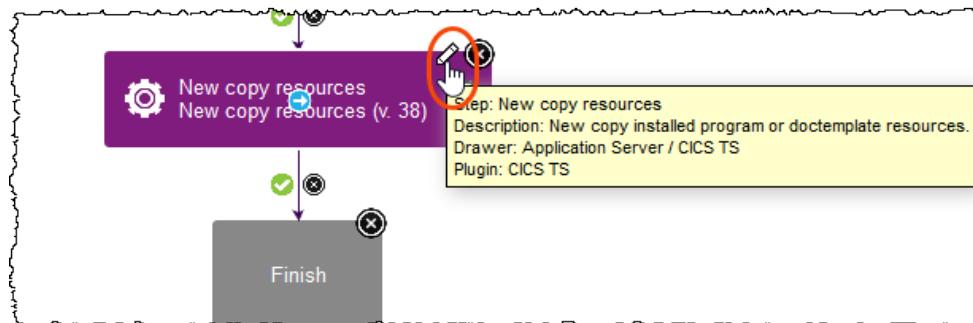


8.21 ►| On the Palette (left side) expand **Application Server** (scroll up) and **CICS TS**.

►| Drag and drop **New copy resources** to the line that connects to the *Finish* box..

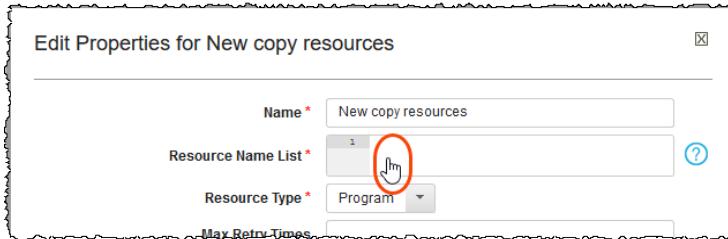


8.22 ►| Click on the pencil icon on **New copy resources** to change the UCD properties for this step.

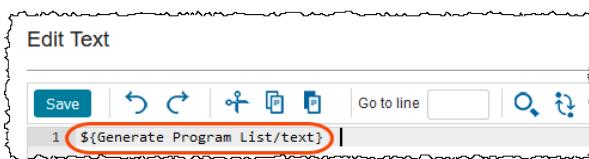


8.23 You need to specify a list of the load modules that CICS needs to do the *Newcopy*.

► Click on **Resource Name List**

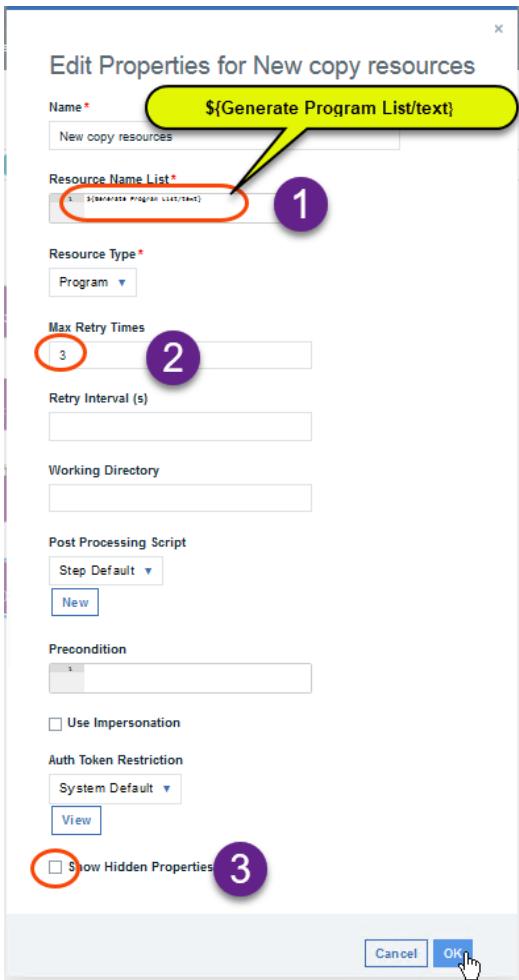


► Type **\${Generate Program List/text}** (This is the step name that you created on 8.18) and click **Save**



► 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

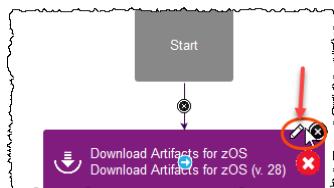
Show Hidden Properties

Host *	<input type="text" value="\${p:cics.host}"/>
Port *	<input type="text" value="\${p:cics.cmciport}"/>
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 *

Directory Offset *

Working Directory

Post Processing Script

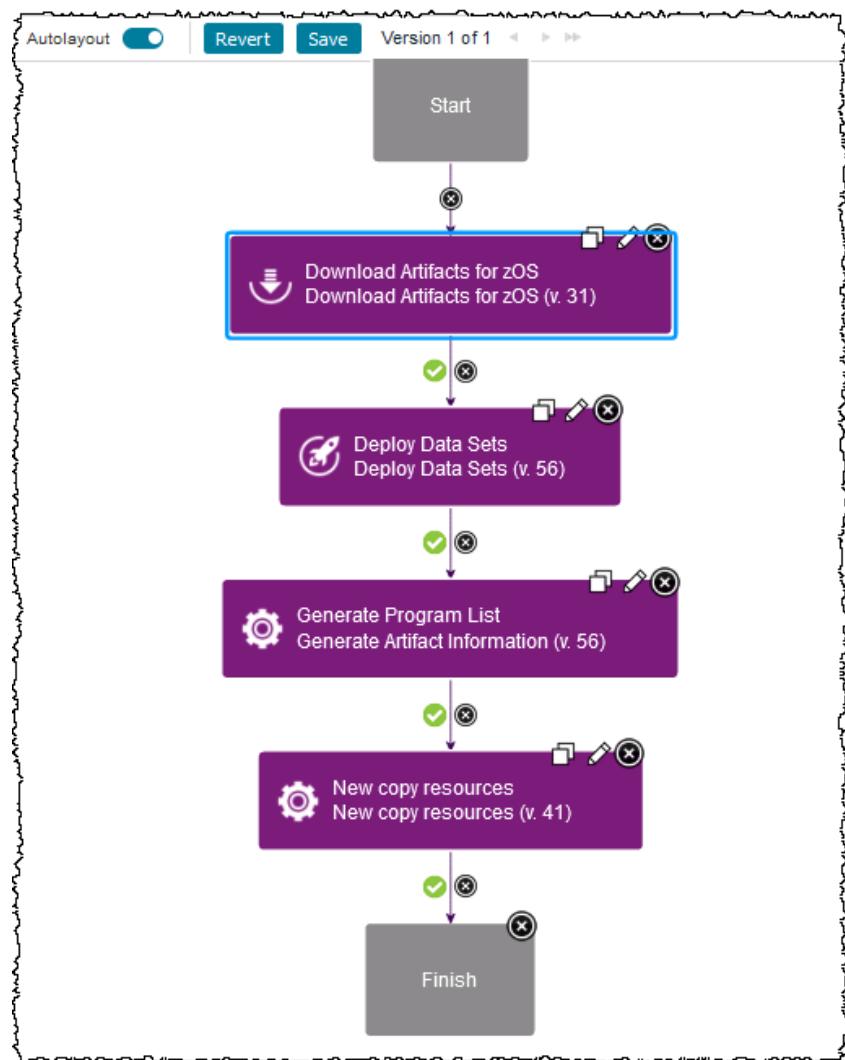
Precondition

Use Impersonation

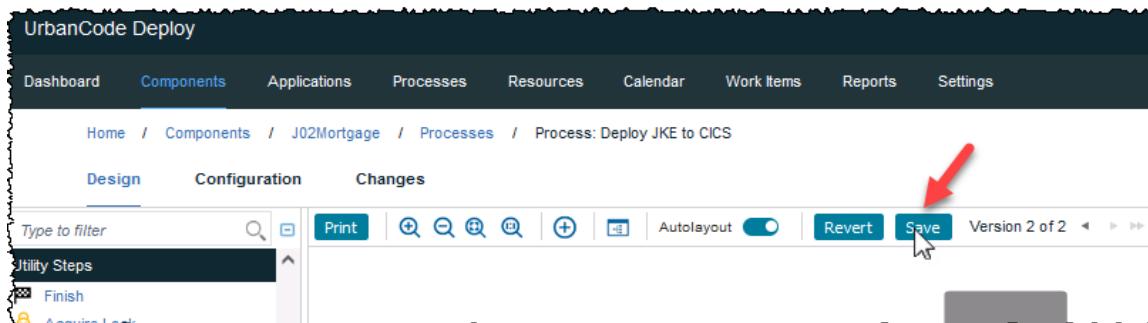
Auth Token Restriction

Show Hidden Properties

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		

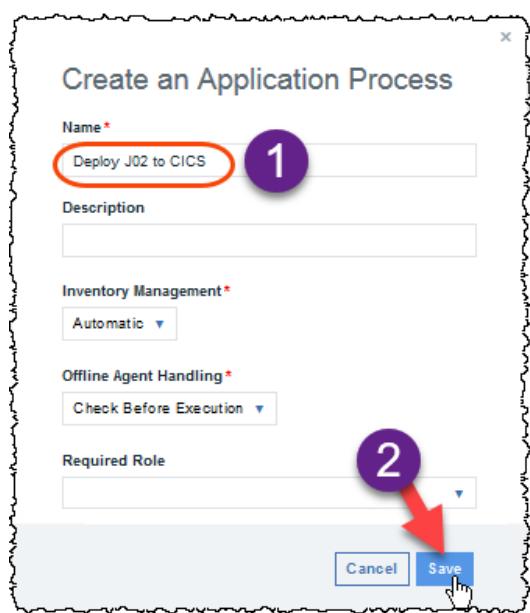
- 9.2 ► Click Components tab and click Add Component:

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

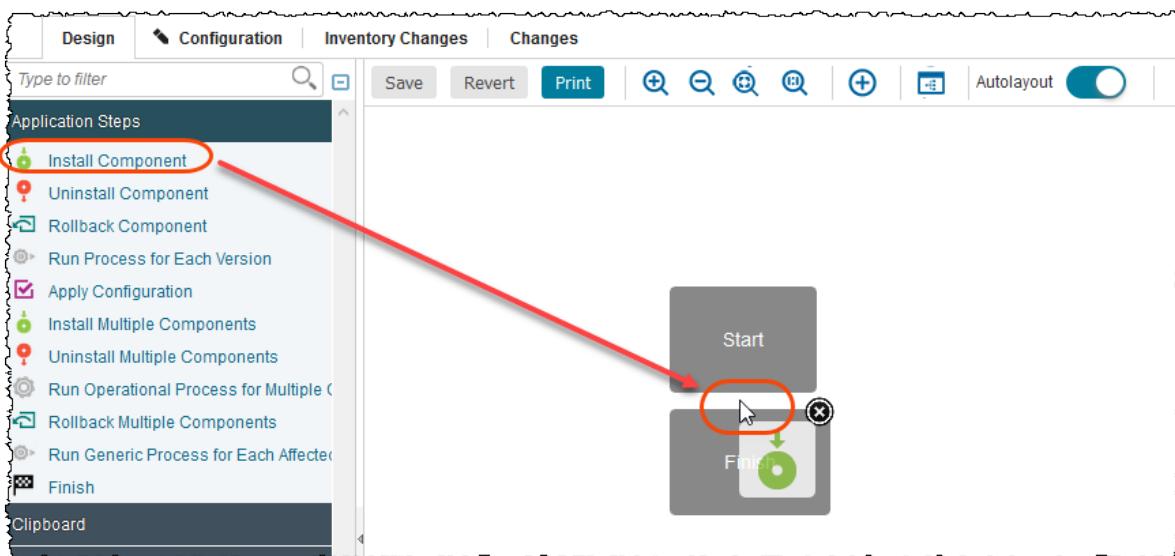
9.4 ► Click the **Processes** tab and click **Create Process**:



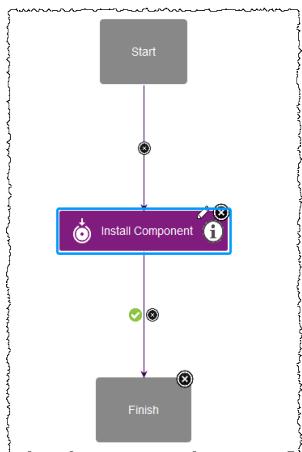
9.5 ► Give a name to your process, like **Deploy J02 to CICS** and click **Save**:



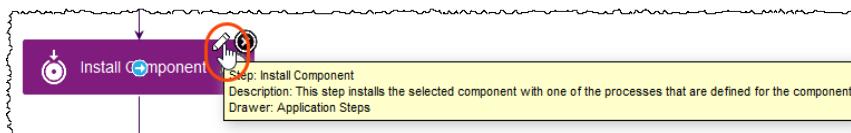
9.6 ► Find **Install Component...** step in the Design Palette, and drag it onto the space between **Start** and **Finish** boxes.



9.7 The result will be as below:

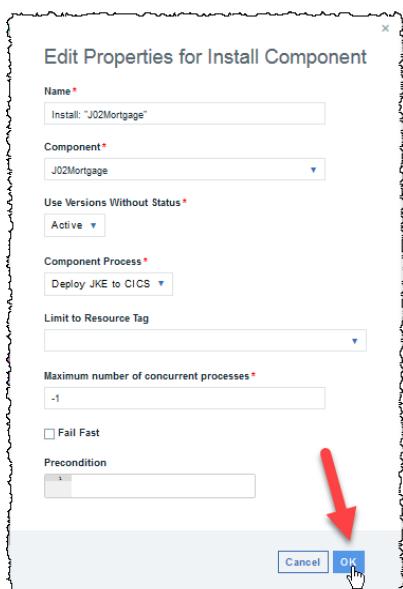


9.8 ➡ Click on the pencil icon on **Install Component** to possibly change the properties

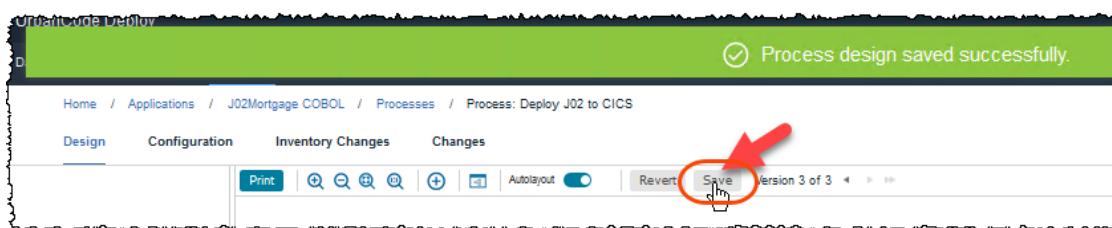


9.9 Notice that the properties of this step are pre filled for you.

➡ Click **OK**:



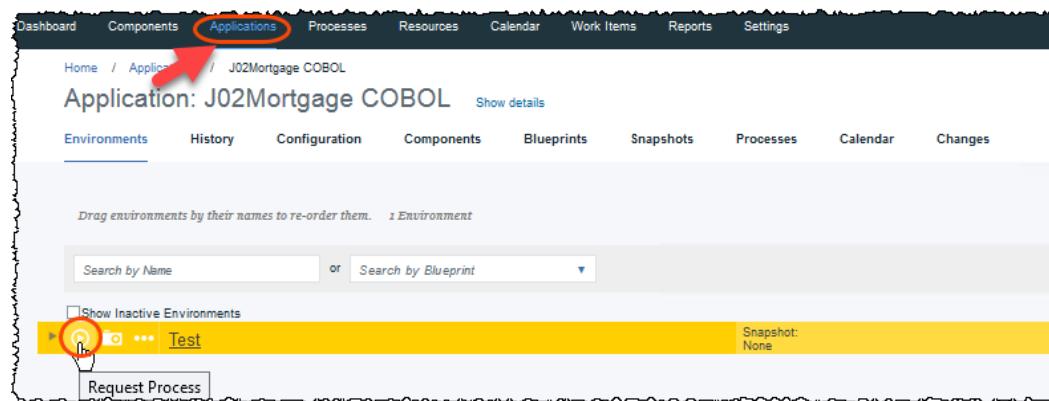
9.10 ➡ Click **Save**



Task 10 – Environment properties configuration

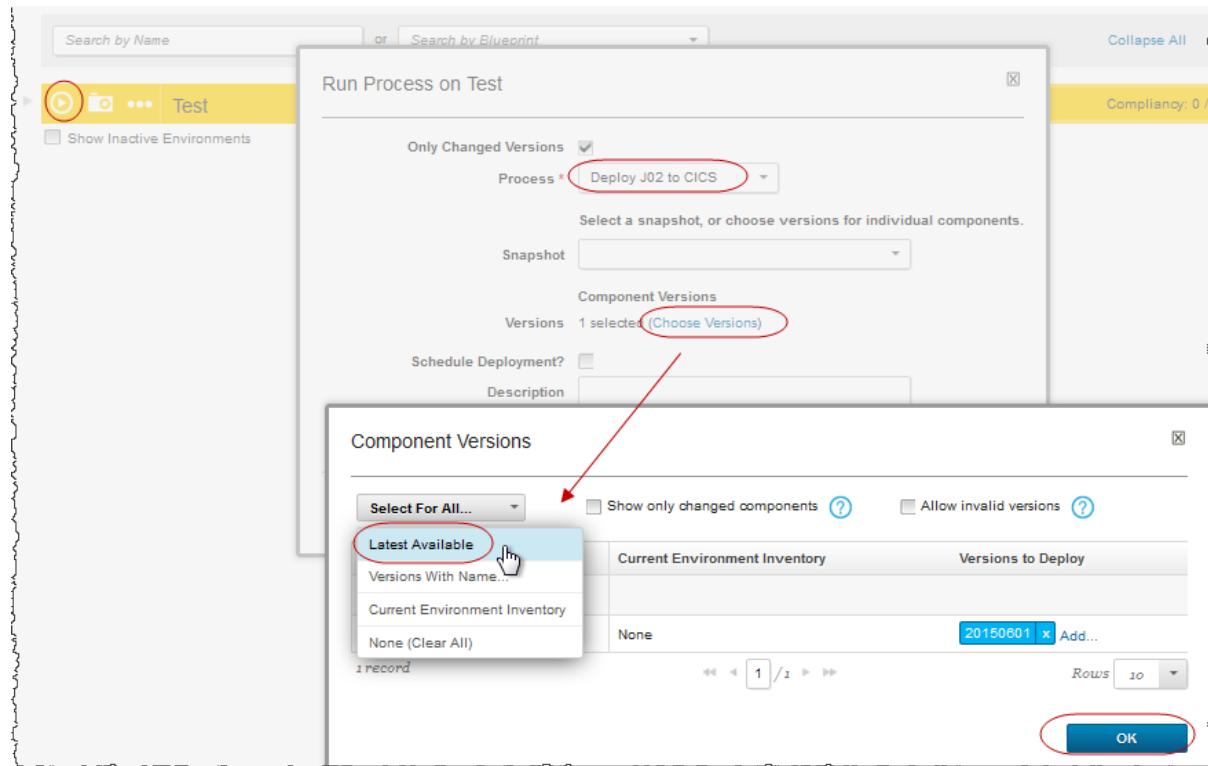
On the task 7 you created the **Test** environment. For this lab we have just one environment, but usually we have few environments. Example the **Test** environment could reflect the Test LPAR and **Prod** environment that would reflect other LPAR. Each environment may have different values used on 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



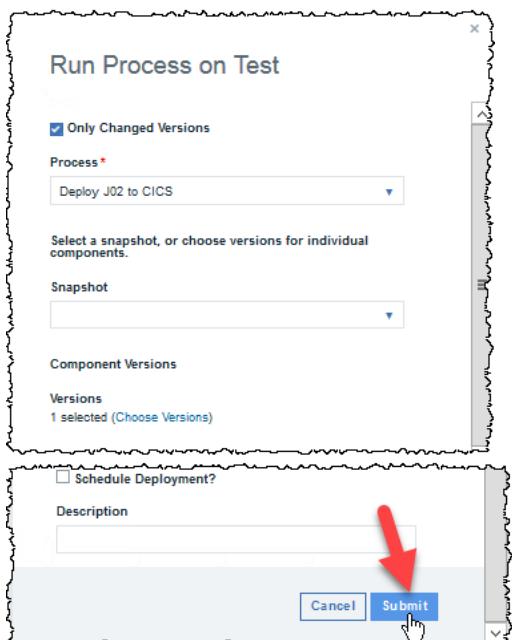
The screenshot shows the Application Management interface. The top navigation bar has tabs for Dashboard, Components, Applications (which is highlighted with a red circle), Processes, Resources, Calendar, Work Items, Reports, and Settings. Below the navigation bar, the breadcrumb path shows Home / Applications / J02Mortgage COBOL. The main title is Application: J02Mortgage COBOL with a 'Show details' link. Below the title is a navigation bar with tabs: Environments, History, Configuration, Components, Blueprints, Snapshots, Processes, Calendar, and Changes. Under the Environments tab, there is a note: "Drag environments by their names to re-order them. 1 Environment". Below this is a search bar with "Search by Name" and "Search by Blueprint" options. A checkbox for "Show Inactive Environments" is unchecked. The environment list shows one entry: "Test" (highlighted with a yellow background and a red circle around the Deploy icon). The Deploy icon is a circular arrow. To the right of the environment name is a "Snapshot: None" label. At the bottom of the list is a "Request Process" button.

- 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 box. The 'Process' dropdown is set to 'Deploy J02 to CICS' (circled in red). Below it is a note: 'Select a snapshot, or choose versions for individual components.' A 'Snapshot' dropdown is present. Under 'Component Versions', it says 'Versions 1 selected (Choose Versions)' (circled in red). A 'Schedule Deployment?' checkbox is unchecked. A 'Description' input field is empty. A 'Compliance' section shows '0' with a 'Collapse All' button. A modal window titled 'Component Versions' is open. It has a dropdown menu with options: 'Select For All...', 'Latest Available' (circled in red and highlighted with a mouse cursor), 'Versions With Name...', 'Current Environment Inventory', and 'None (Clear All)'. There are also checkboxes for 'Show only changed components' and 'Allow invalid versions'. Below the dropdown is a table with columns 'Current Environment Inventory' and 'Versions to Deploy'. It shows 'None' in both columns. A '20150801' button with an 'x' and an 'Add...' button are in the 'Versions to Deploy' column. At the bottom of the modal is an 'OK' button (circled in red).

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

The screenshot shows the 'Application Process Request: J02 Mortgage COBOL' page. At the top, there are tabs: Log, Properties, Manifest, Configuration Changes, and Inventory Changes. Below is a table titled 'Execution' with columns: Step, Progress, Start Time, Duration, and Status. The first row shows a step named '1. Install: "J02Mortgage"' with progress 0 / 1, started at 4:57:55 PM, duration 0:00:49, and status Running. The second row is a summary 'Total Execution' with the same values. At the top right of the table, there are buttons for Pause, Cancel, and Download All Logs, followed by 'Expand All' and 'Collapse All' links, with a red arrow pointing to the 'Expand All' link.

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

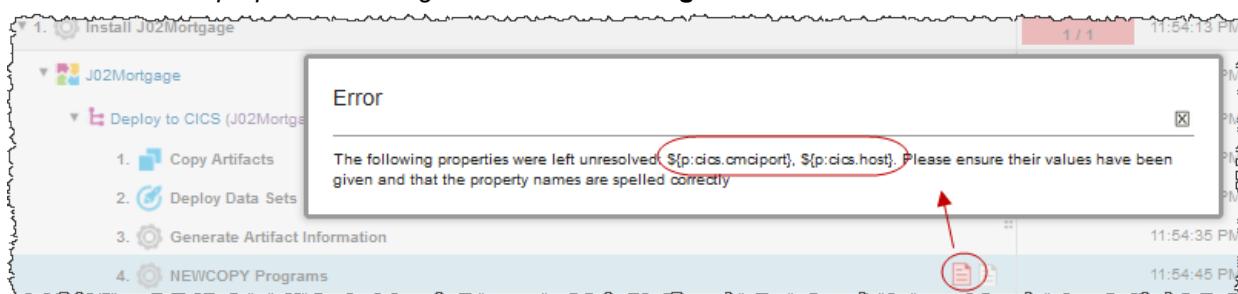
This screenshot shows the same 'Execution' table from the previous screenshot, but with more detail. The first step, '1. Install: "J02Mortgage"', has its details expanded, showing a sub-table with four sub-steps: '1. Deploy JKE to CICS (J02Mortgage 20180709)', '2. Download Artifacts for zOS', '3. Deploy Data Sets', and '4. Generate Program List'. A large yellow callout bubble points to the 'View Child Execution' link in this expanded section. Another yellow callout bubble points to the expand icon (a triangle icon) next to the first step in the main table. The table structure is identical to the one in the previous screenshot, with columns for Step, Progress, Start Time, Duration, and Status.

10.6 Be patient.. this is slow on your environment.. Three steps were successful but the last one failed.

▶ 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)			1:11:20 PM	0:03:12	● Failed
1. ⚒ Download Artifacts for zOS			1:11:20 PM	0:01:15	● Success
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)		1 / 1	1:11:20 PM	0:03:12	● Failed
Error Log					

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 << < 1 > >>

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**.

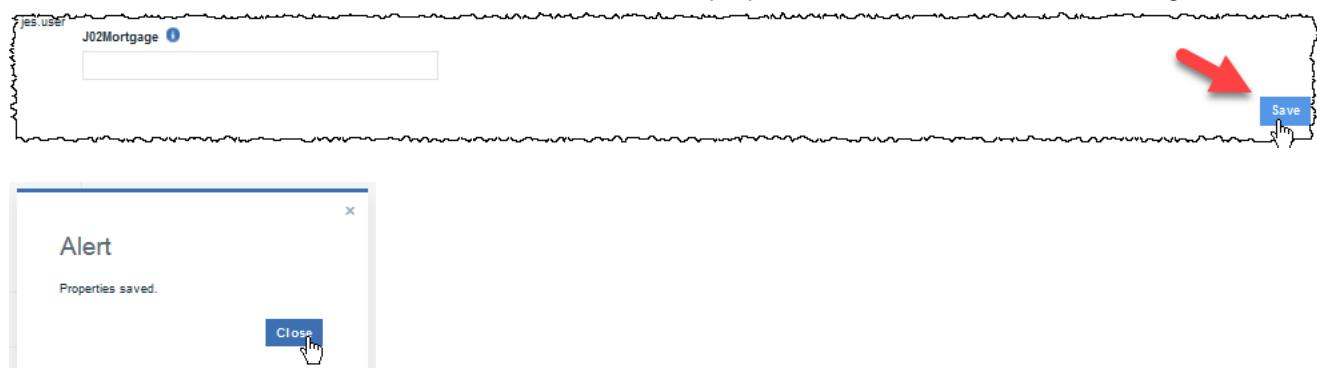
In the real world you will not add the password here and will use other secure way to authenticate., also
In the Edit Property dialog, you can check the secure box and this will hide (secure) the password..

When complete:



Name	Value	Description	Actions
Filter	Filter		
cics.omcport	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.

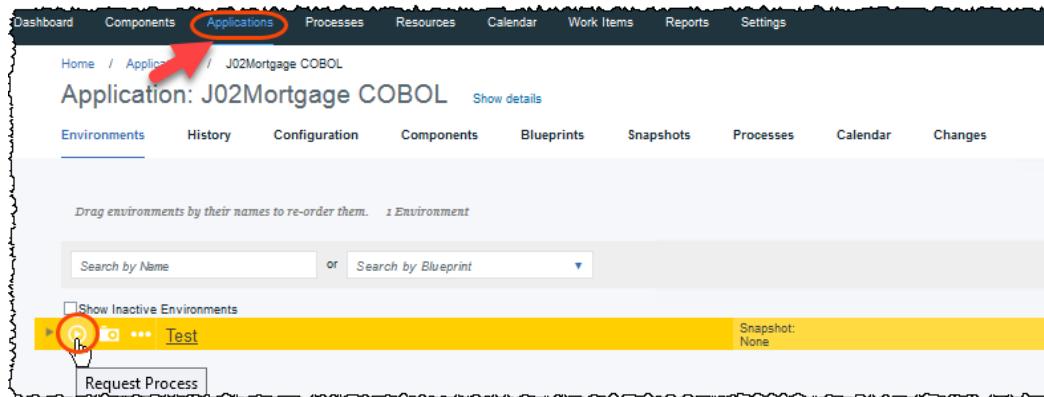
Task 11 – Run an application process

You are now ready to test our deployment process.

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

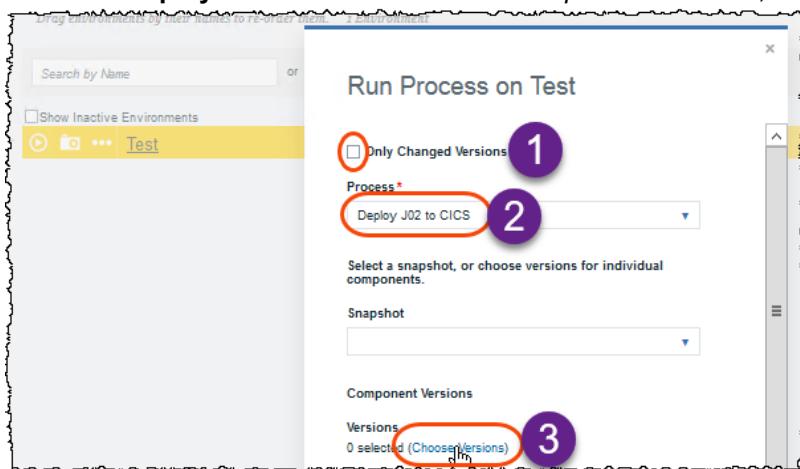
Name	Template	Description	Created
A HC zOS COBOL CICS		Deploy the HC application to CICS - No DB2 Binding	12/14/2017, 4:51 PM
Account Management CICS and Worklight (LINUX)		Used in PDTTOOLS demo	8/18/2015, 11:02 PM
J02 Mortgage zOS and Worklight		J02 CICS + JKE Mobile	12/16/2014, 1:38 PM
J02Mortgage_COBOL			5/8/2020, 8:37 AM
JKE Emulator			12/15/2014, 6:17 PM

11.2  Under Test environment click the Request Process  icon

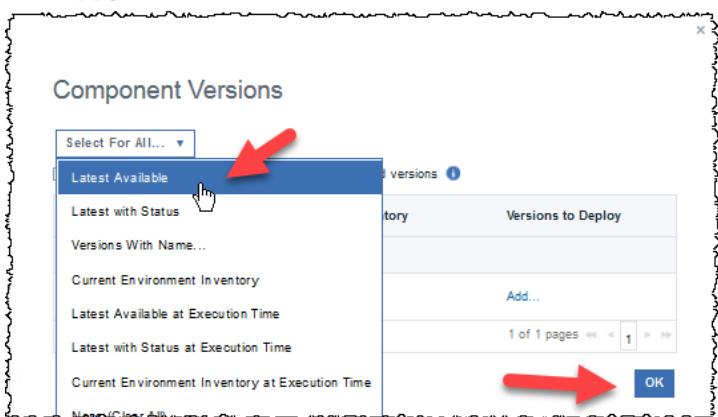


The screenshot shows the UrbanCode Deploy application interface. At the top, there is a navigation bar with links: Dashboard, Components, Applications (which is highlighted with a red oval), Processes, Resources, Calendar, Work Items, Reports, and Settings. Below the navigation bar, the URL is shown as Home / Applications / J02Mortgage COBOL. The main title is Application: J02Mortgage COBOL with a "Show details" link. Below the title, there are tabs: Environments, History, Configuration, Components, Blueprints, Snapshots, Processes, Calendar, and Changes. Under the Environments tab, there is a message: "Drag environments by their names to re-order them. 1 Environment". Below this are search fields for "Search by Name" and "Search by Blueprint". There is also a checkbox for "Show Inactive Environments". A yellow bar at the bottom contains a "Request Process" button, which is highlighted with a yellow circle and a red arrow pointing to it from the left.

- 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**

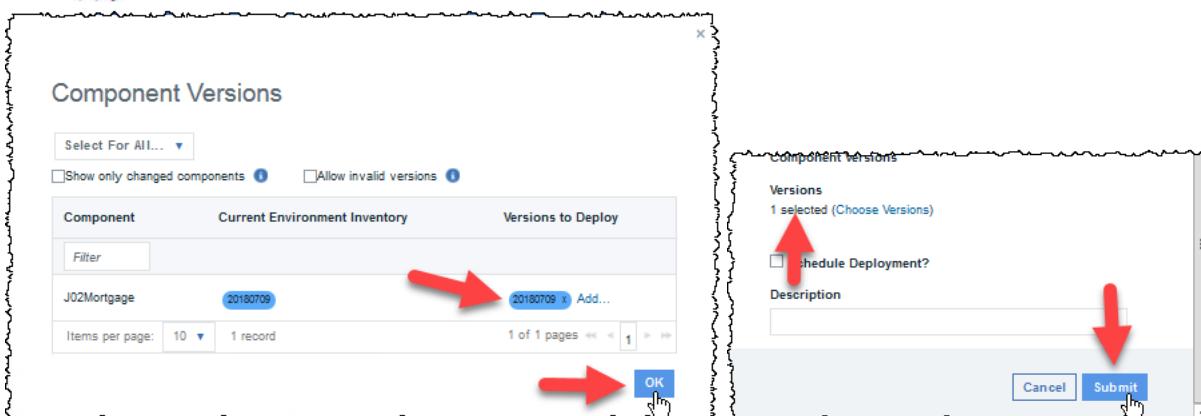


- 11.4 ► Click **Select For All > Latest available** click **OK**



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 deployment.

Application Process Request: J02 Mortgage COBOL (show details)

Log Properties Manifest Configuration Changes Inventory Changes

Execution

Pause Cancel Download All Logs Expand All Collapse All

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				Not Started
4. New copy resources				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 6:08:25 PM Progress 1 / 1 Status Success Duration 0:02:23 End 6:10:49 PM

Repeat Request Download All Logs Expand All Collapse All

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.

Step	Progress	Start Time	Duration	Status
▼ 1. Install: "J02Mortgage"	1 / 1	4:48:45 PM	0:04:11	Success
▼ J02Mortgage	1 / 1	4:48:45 PM	0:04:11	Success
▼ Deploy JKE to CICS (J02Mortgage 20180709)	1 / 1	4:48:45 PM	0:04:11	Success
1. Download Artifacts for zOS	1 / 1	4:48:45 PM	0:01:19	Success
2. Generate Program List	1 / 1	4:48:05 PM	0:00:27	Success
3. Deploy Data Sets	1 / 1	4:48:32 PM	0:01:32	Success
4. New copy resources	1 / 1	4:50:04 PM	0:00:52	Success
Total Exec.	1 / 1	4:48:45 PM	0:04:11	Success

Output Log 1

```

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 "J02MPMT" succeeded.
41 2020/05/08 20:51:19.714 GMT BUZCP0029I 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 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"
l cicsts53_
M 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 . . .

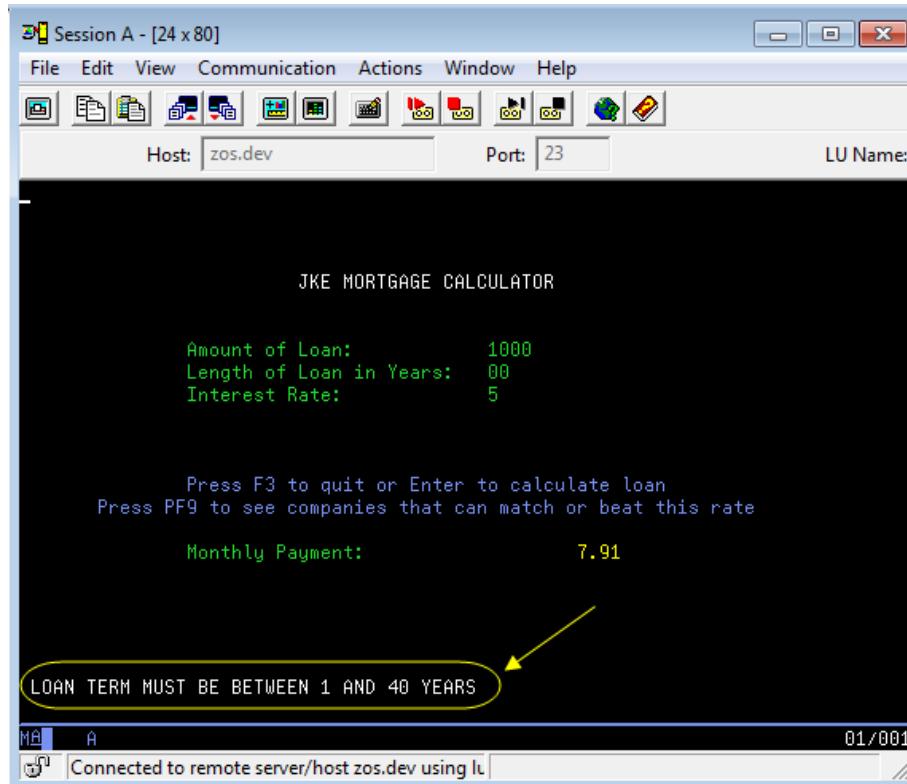
DFHCE3520 Please type your userid.
F3=Exit
M A                                         11/033

```

12.4 ➔ Type the transaction **J02P**.



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



Congratulations! You completed the lab.

LAB 3A – (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.

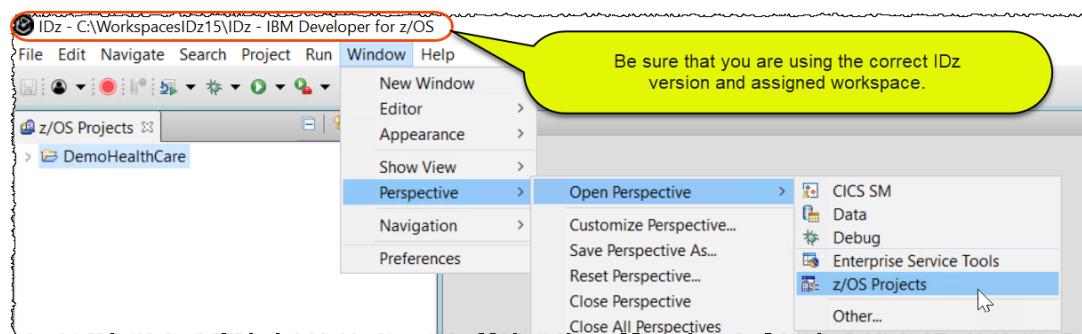
1.1.1 Start *IBM Developer for z Systems version 15* if it is not already started

- ▶ 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**



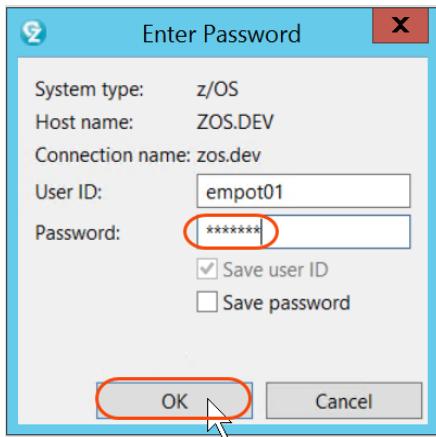
1.1.3 You will use userid **empot01**.

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

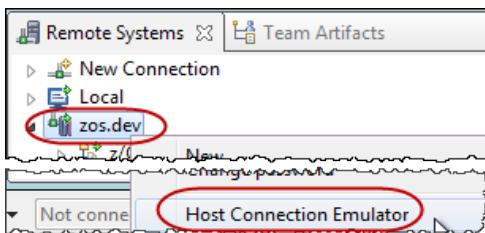
- ▶ Using *Remote Systems* view, right click on **zos.dev** and select **Connect**

(You also may need to disconnect first)

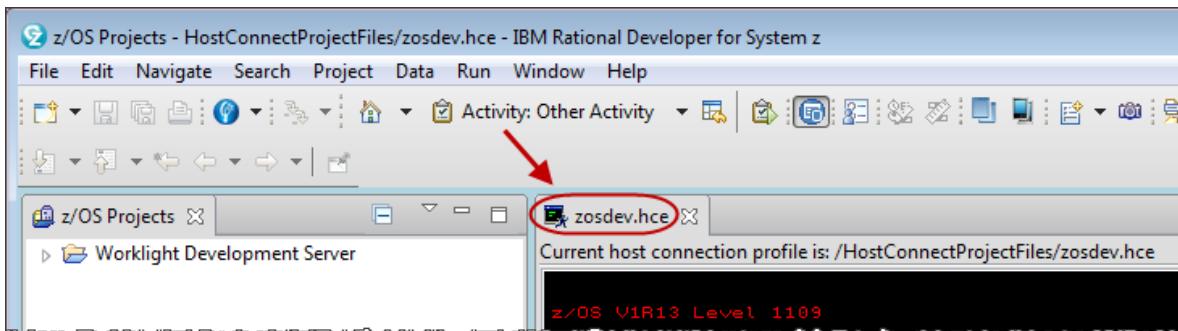
- 1.1.4 ► Type **empot01**as userid and **empot01** as password.
 The userid and password can be any case; don't worry about having it in UPPER case.
 Click **OK** to connect to z/OS.



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



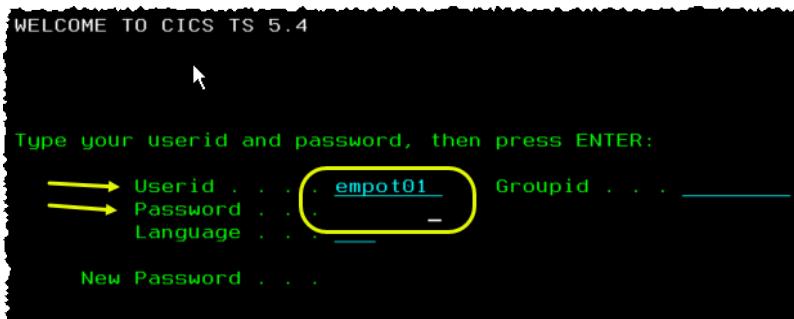
- 1.1.6 ► Since you will need more space, **double-click** on the **zos.dev.hce** title



- 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**.



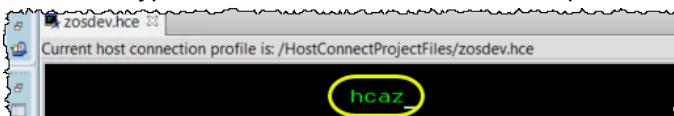
1.1.9 The sign-on message is displayed



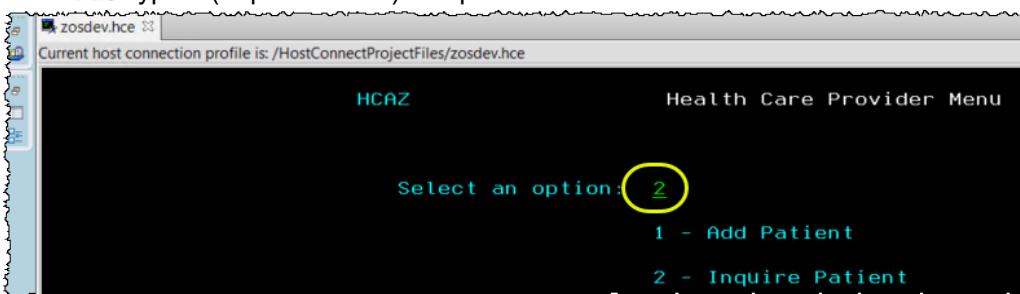
1.2 Run CICS transaction HCAZ

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**.

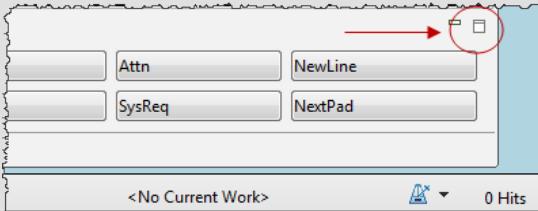


1.2.3 ► Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**

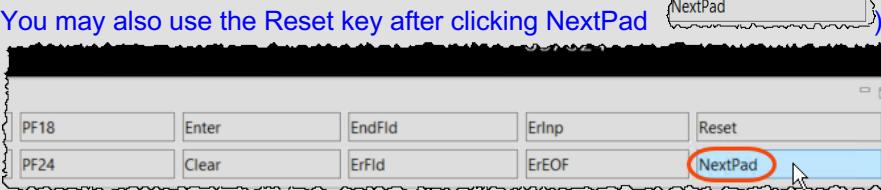


IF you need to use functions like Clear or Reset

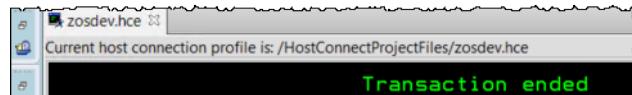
If you need to use the **clear** function use the key **Esc**.
Also you may look in the right lower corner, select this icon . This will display possible keys, including the clear button.



You may also use the Reset key after clicking NextPad

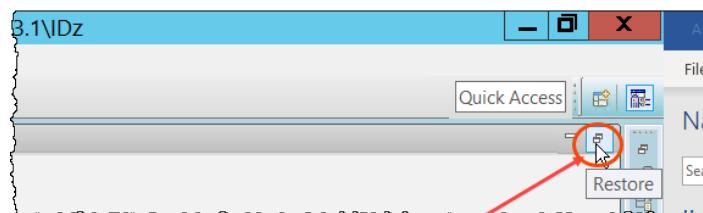


1.2.4 Press **F3** to end the application.



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.

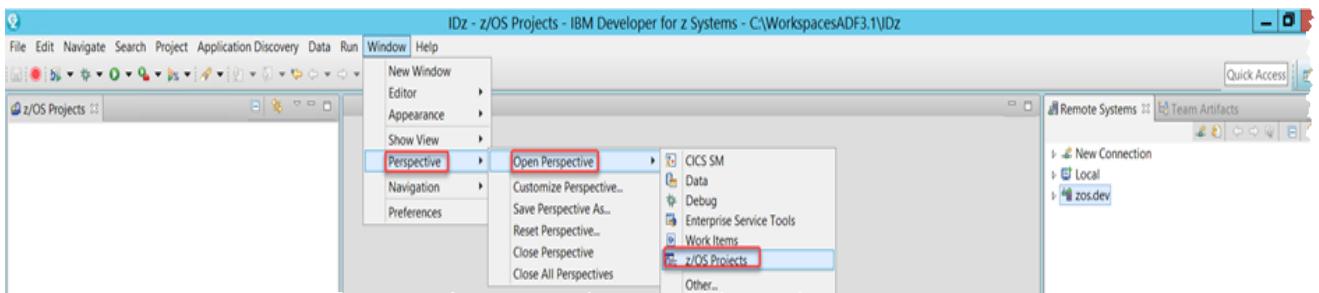
Section 2 – Import a z/OS project

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

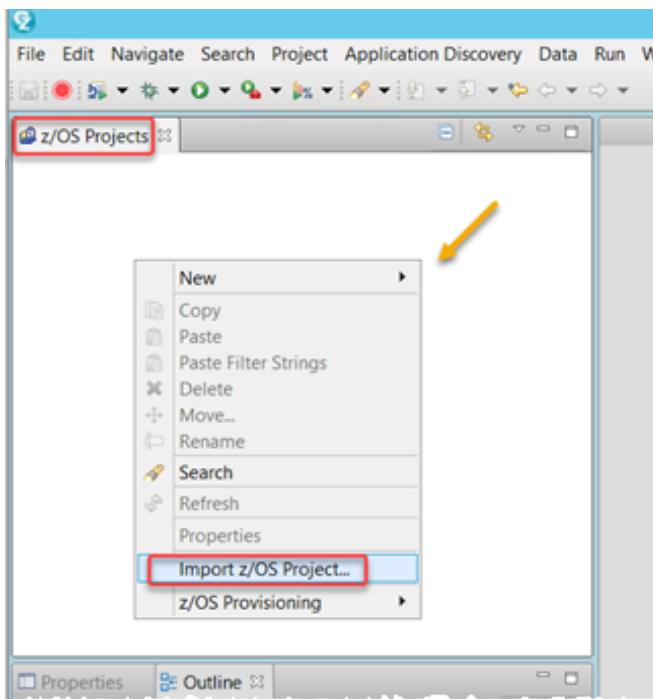
2.1 Importing the LAB6B z/OS Project

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).

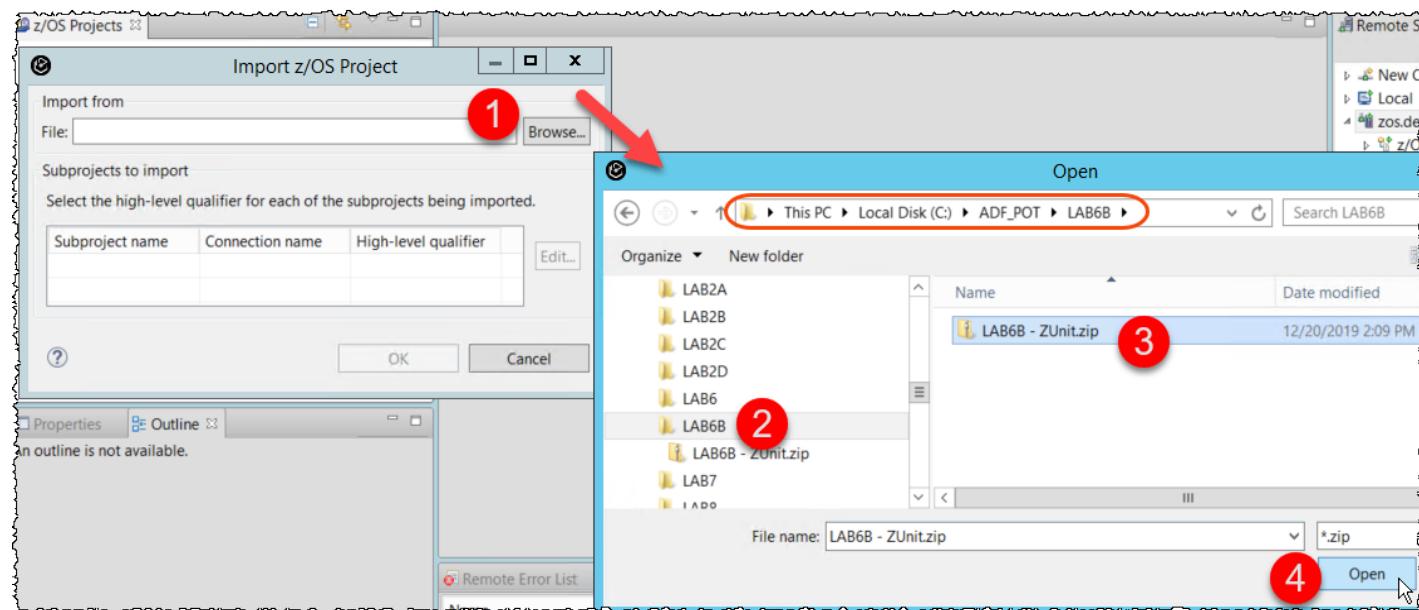
- 2.1.1 ► If not already in the *z/OS Projects* perspective, open the ***z/OS Projects*** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



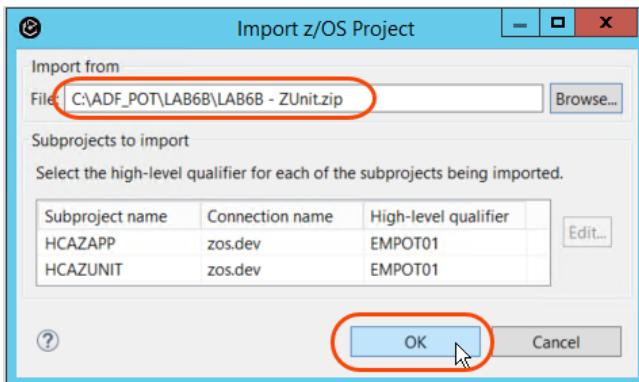
- 2.1.2 ► Using the ***z/OS Projects*** view on top left, position the cursor anywhere in the view. Right mouse click and select the **Import z/OS Project** action.



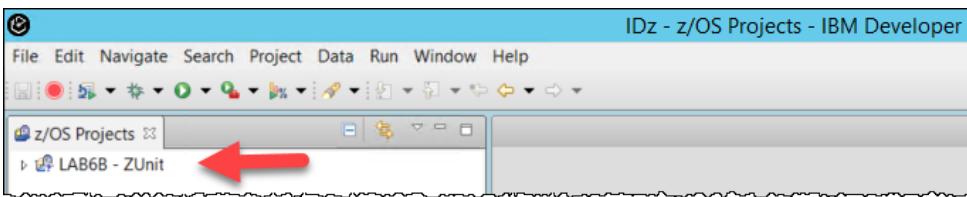
2.1.3 ► In the Import z/OS Project dialog, click the **Browse** button, navigate to C:\ADF_POT\LAB6B and select **LAB6B - ZUnit.zip**. Click **Open**.



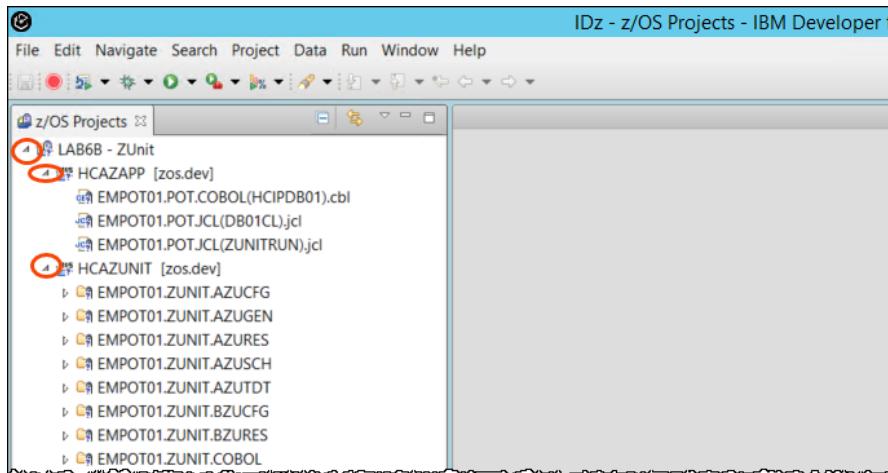
2.1.4 ► In the dialog that opens, click **OK**.



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.

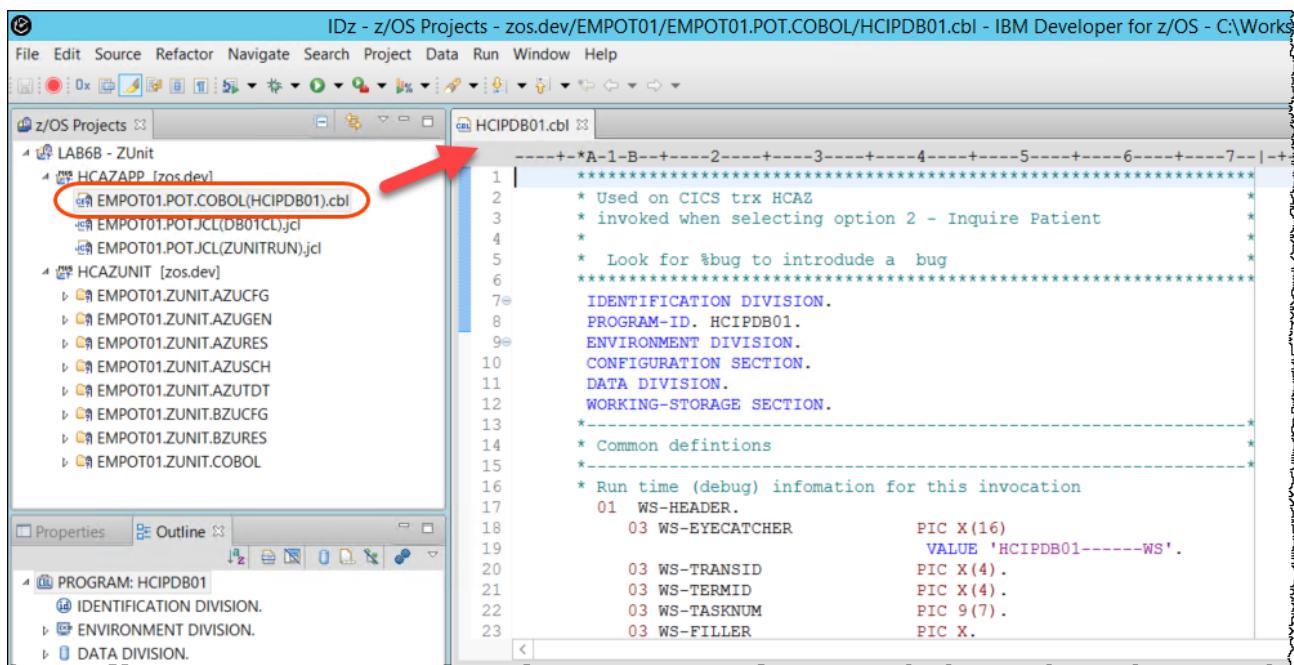
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

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..

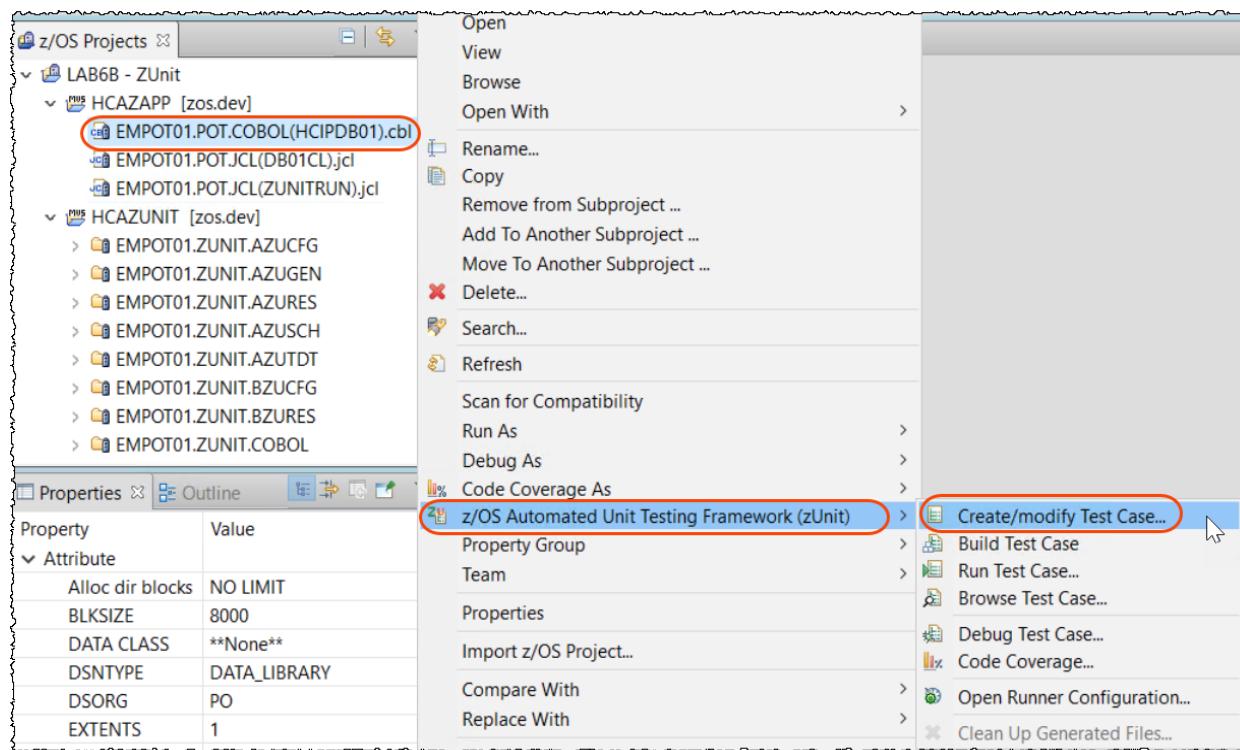
```

z/OS Projects <> HCIPDB01.cbl <> Properties <> Outline <> Remote Error List <> z/OS File System Mapping <> Property Group Manager <>
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,

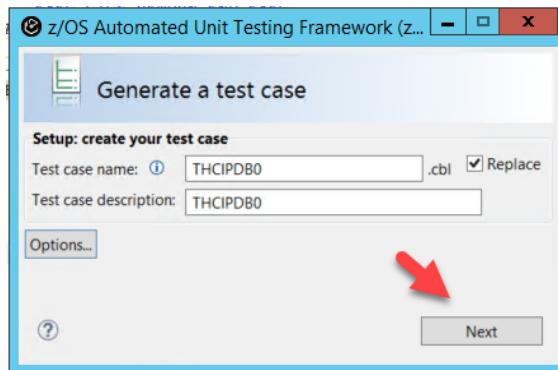
```

3.2 Recording the COBOL program that sends the message

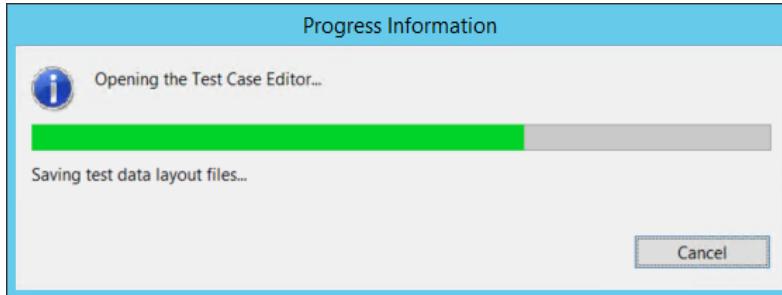
3.2.1 ► To start the recording, right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)-> Create/modify Test Case..**



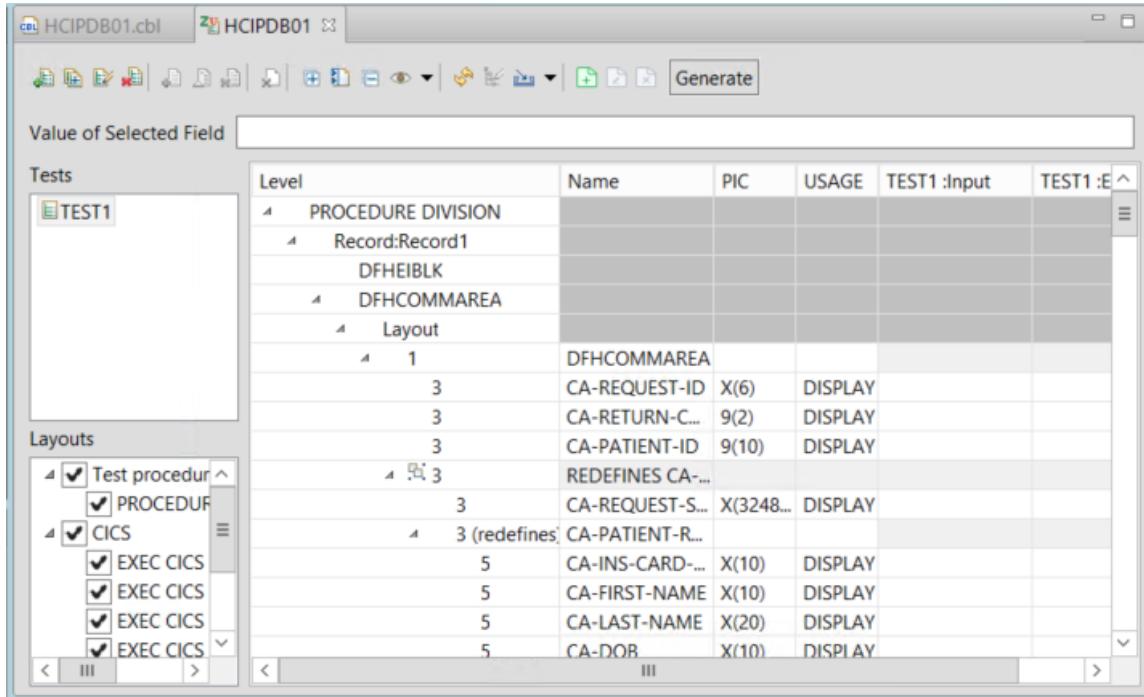
3.2.2 ► This opens a dialog where you can name your test case.
Accept the auto-generated name and click **Next**.



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

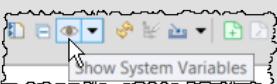


3.2.4 This will open the **Test Case Editor**, as shown below. TEST1 may or may not be on your screen.
If TEST1 is there you will delete on next step.



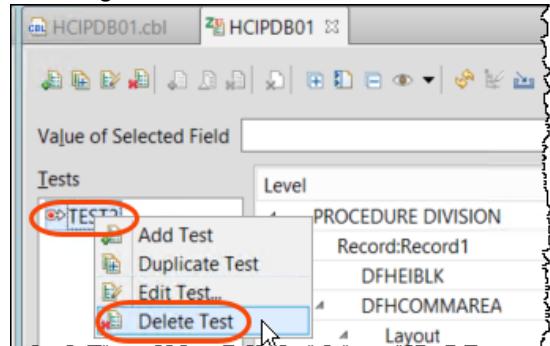
Understanding the test case editor

- 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**



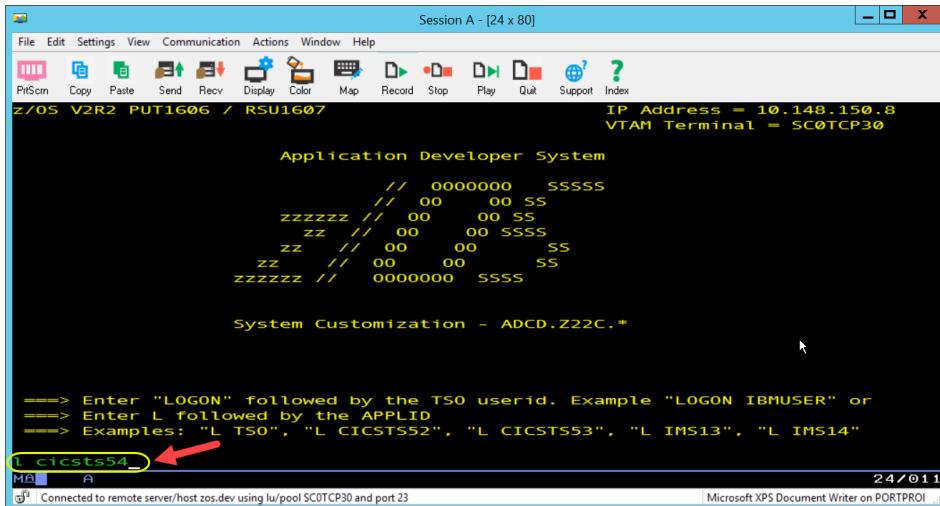
Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

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.



- 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

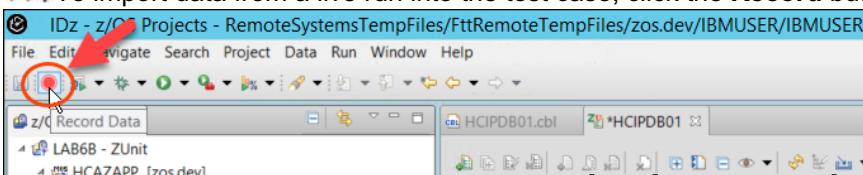
If you need to use the *clear* or *reset* function right mouse click and select as below



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

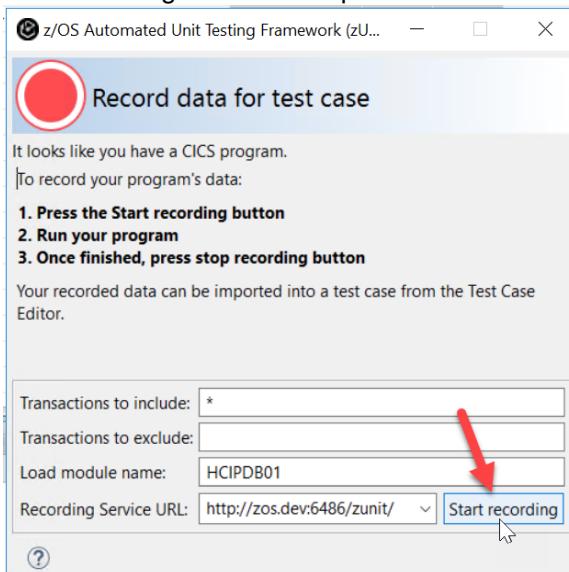
- 3.2.10 ►| Minimize the terminal emulator and go back to IDz dialog.

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

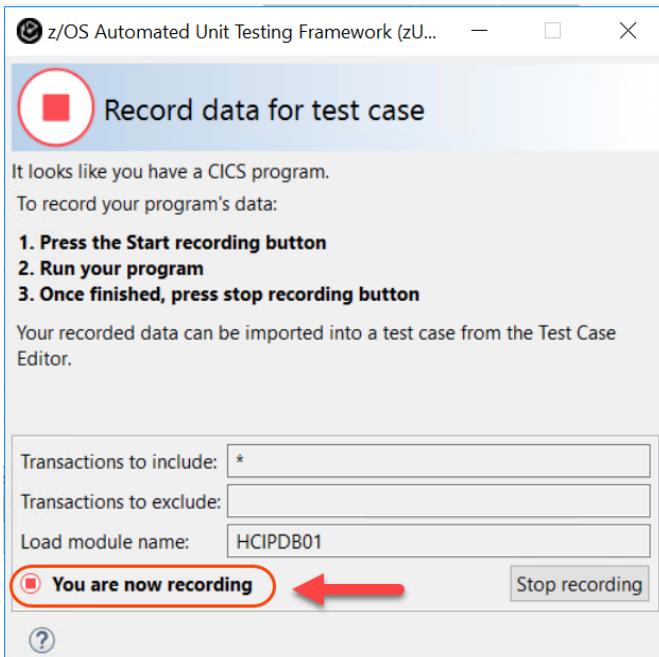


- 3.2.11 ►| In the dialog that comes up, click on **Start recording**.

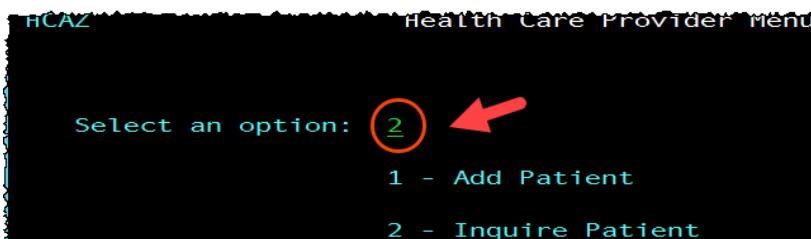
The Recording Service URL points to the CICS region where the live run is recorded.



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



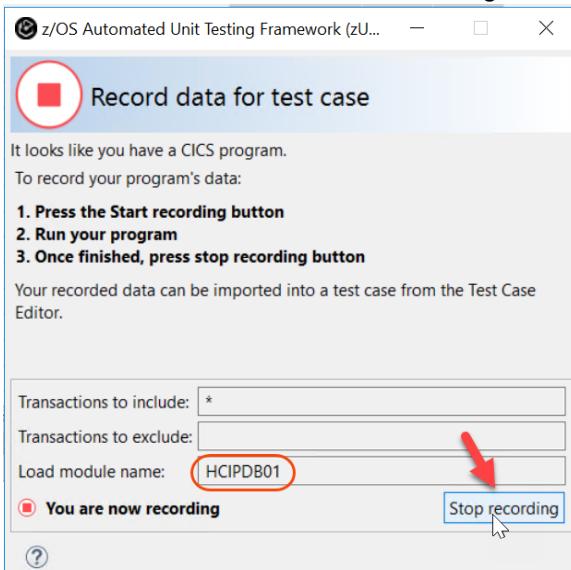
3.2.13 ► Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



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

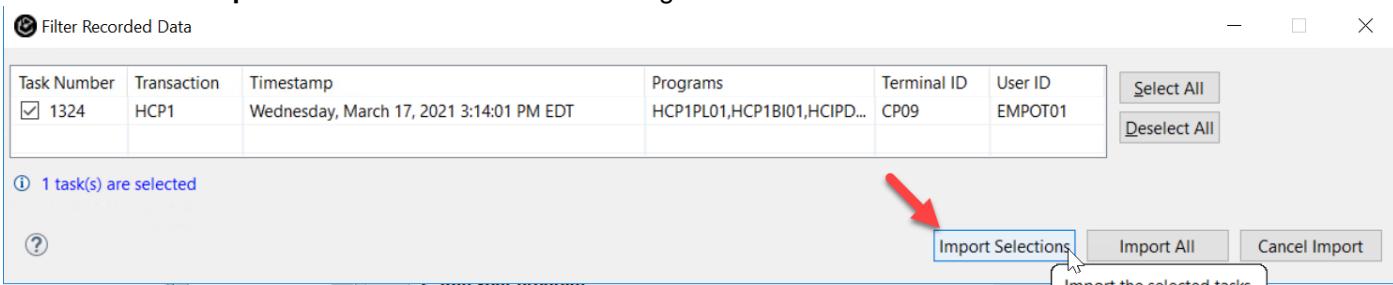


3.2.15 ►| Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



The dialog Filter Recorded Data is displayed.

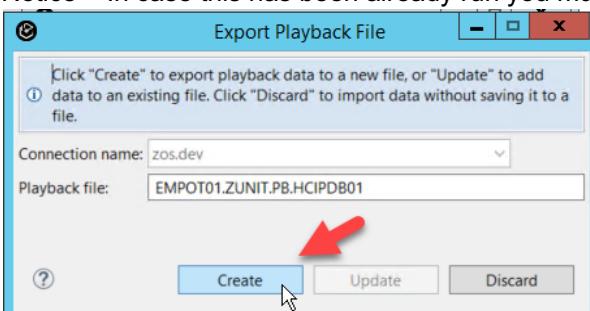
3.2.16 ►| Click **Import Selections** to dismiss the dialog.



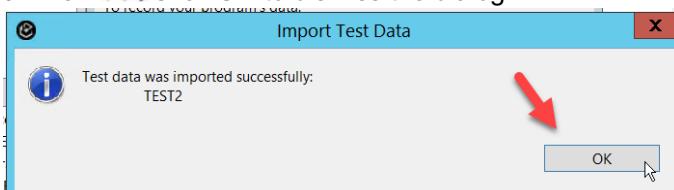
3.2.17 ►| Change the *Playback file* to **EMPOT01.ZUNIT.PB.HCIPDB01** (instead of **EMPOT05**)

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

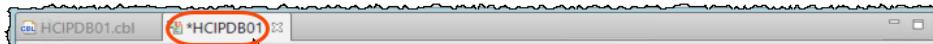
Notice – In case this has been already run you must select **Update** instead.



3.2.18 ►| Click **OK** to dismiss the dialog



3.2.19 ► Double click on the **Hcipdb01** title to enlarge the view.

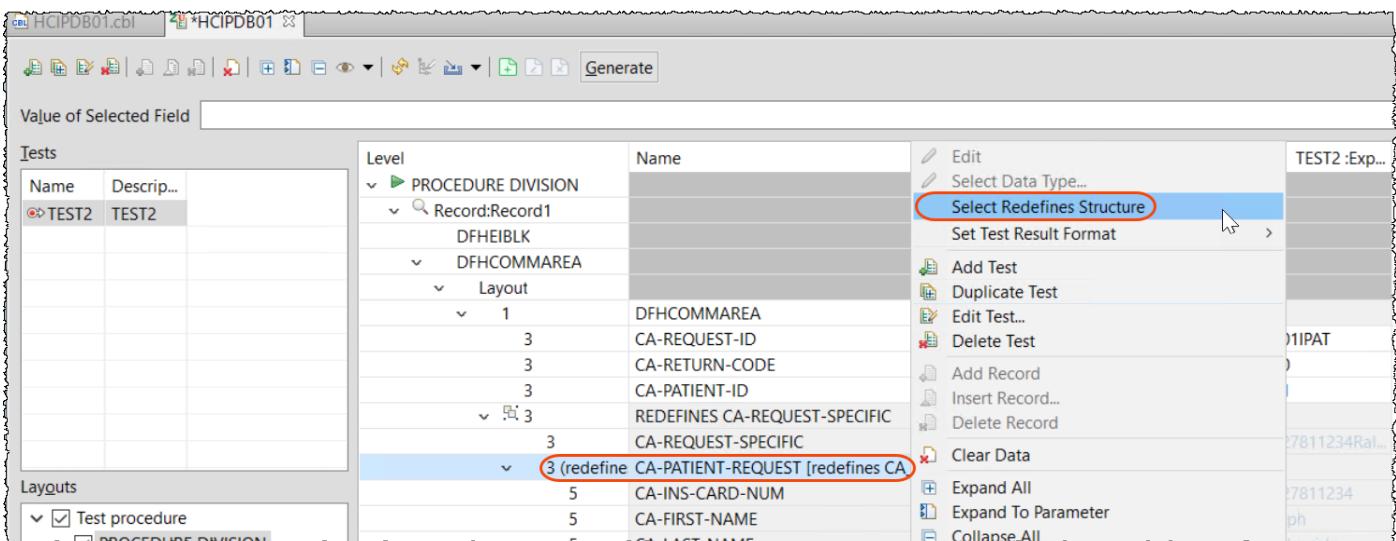


3.2.20 You now see a new test created in the editor and populated with the values from the live run. Also you may notice that this program has many redefines.

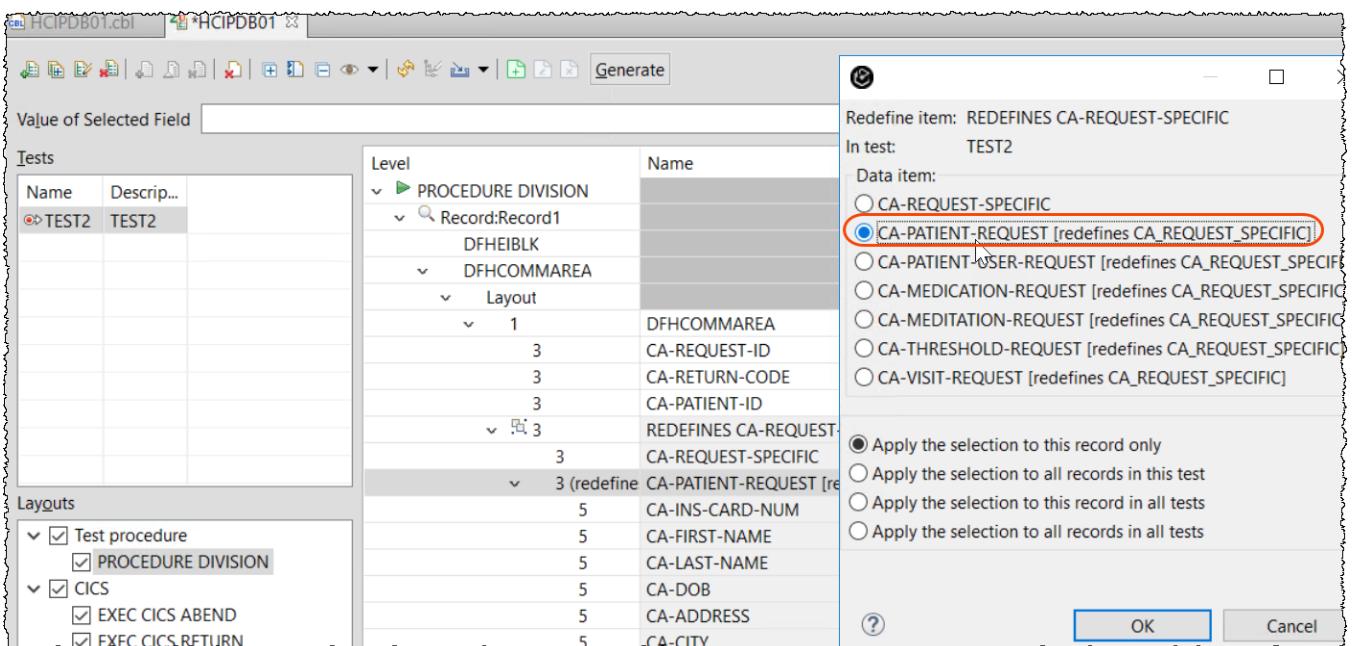
You need to identify which is the area being used on this program execution. In our example is the “CA-PATIENT-REQUEST” area being redefined.

► 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	901PAT
3	CA-RETURN-CODE	9(2)	DISPLAY	0	0
3	CA-PATIENT-ID	9(10)	DISPLAY	1	1
3	REDEFINES CA-REQUEST-SPECIFIC				
3	CA-REQUEST-SPECIFIC	X(3248...)	DISPLAY		9627811234Ral...
5	CA-INS-CARD-NUM	X(10)	DISPLAY		9627811234
5	CA-FIRST-NAME	X(10)	DISPLAY		Ralph
5	CA-LAST-NAME	X(20)	DISPLAY		DAlmeida ...
5	CA-DOB	X(10)	DISPLAY		1980-07-11
5	CA-ADDRESS	X(20)	DISPLAY		34 Main Stre...
5	CA-CITY	X(20)	DISPLAY		Toronto ...
5	CA-POSTCODE	X(10)	DISPLAY		MSH 1T1
5	CA-PHONE-MOBILE	X(20)	DISPLAY		077-123-998...
5	CA-EMAIL-ADDRESS	X(50)	DISPLAY		RalphD@ibm.c...
5	CA-USERID	X(10)	DISPLAY		ralphd

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

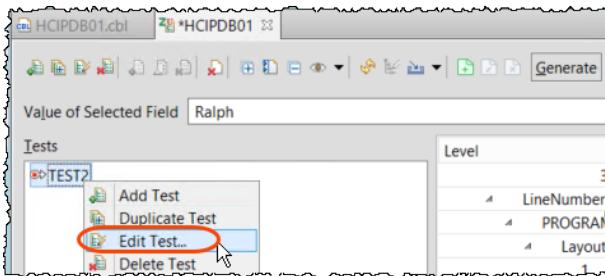
3.2.24 You now see a new test created in the editor and populated with the values from the live run.

► 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

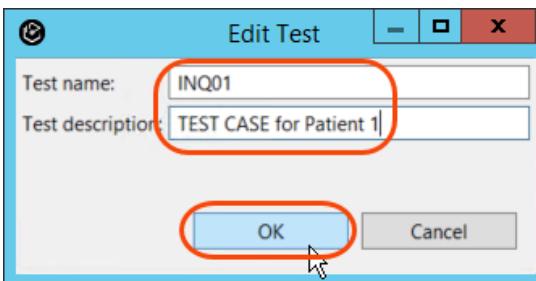
Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
1	HCAZERRS	X(8)	DISPLAY		
COMMAREA					
Layout					
1	CA-ERROR-MSG				
3	FILLER	X(9)	DISPLAY		
3	CA-DATA	X(90)	DISPLAY		
EXEC SQL SELECT INTO [PATI]	line=117				
Record:Record1					
LineNumber=117					
INTO					
5	CA-FIRST-NAME	X(10)	DISPLAY	Ralph	3
5	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	MSH 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	
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.

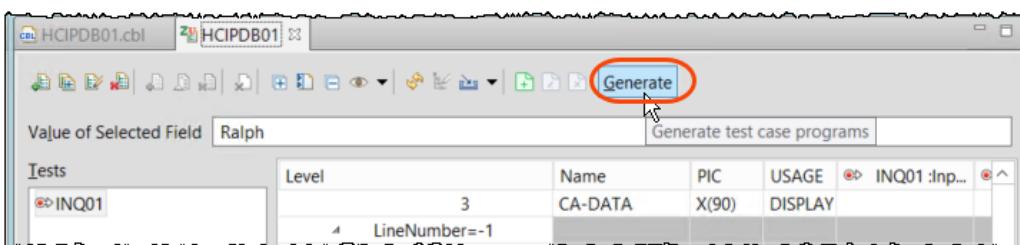


Section 4. Generate, build and run the unit test.

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

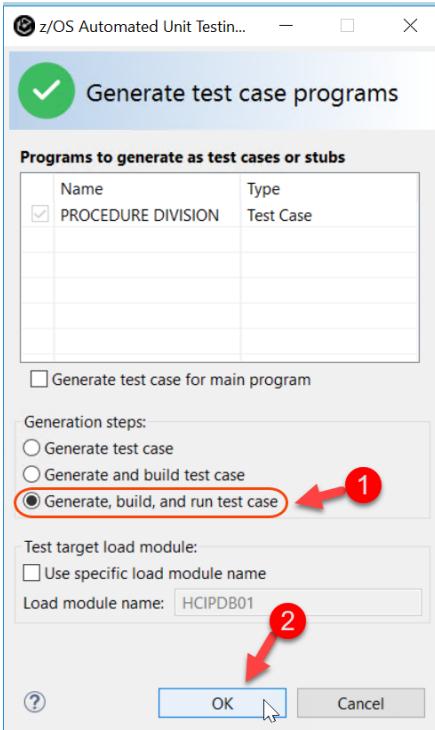
4.1 Generating, building and running the test case program.

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.



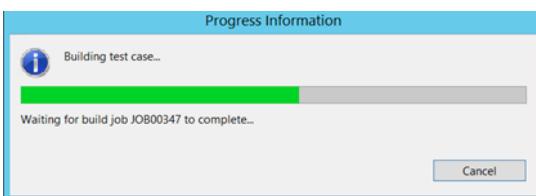
On the **Generate test case programs** dialog,

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

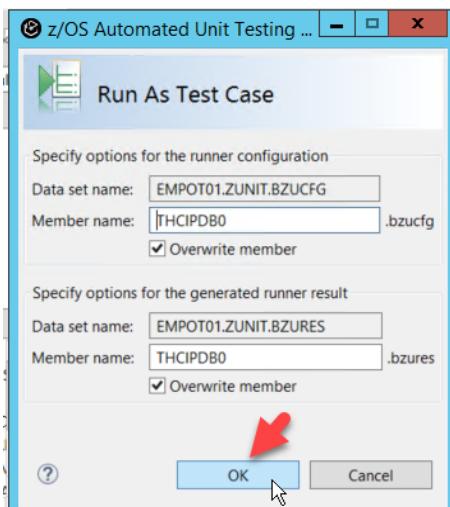


A **Progress information** dialog on the building will open.

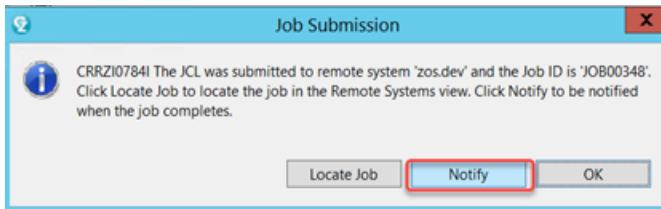
This generation will take a while and a job is submitted to z/OS for execution. The COBOL test cases programs were compiled/link edited and are ready to run.



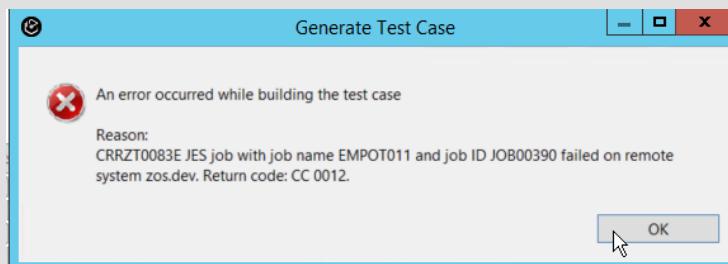
4.1.3 ►| Once the building is complete, a **Run As Test Case** dialog will open, click **OK** to continue.



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

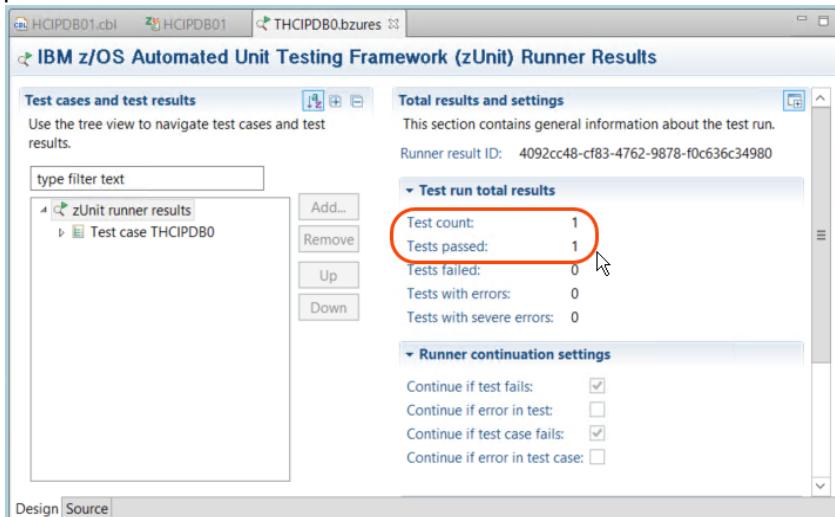
#2. Had a “strange Test fail” ?

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.

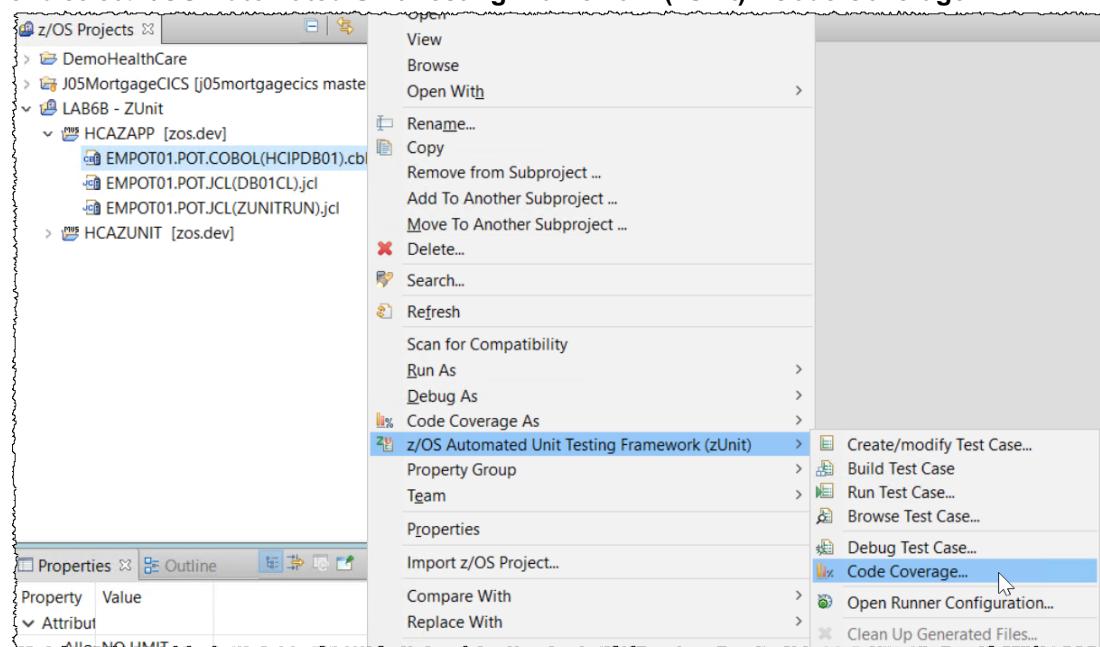
- 4.1.6 ►| Use **Ctrl + Shift + F4** to close all opened editors.

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.

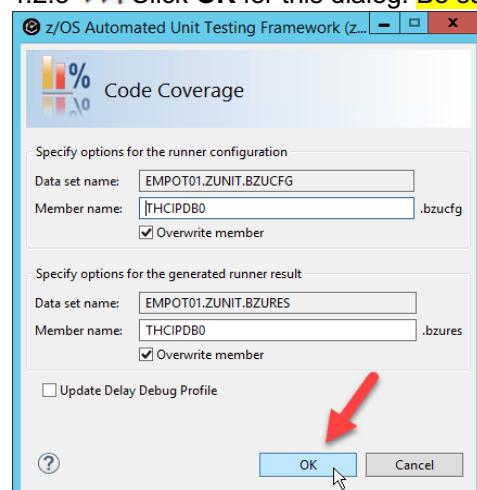
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

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



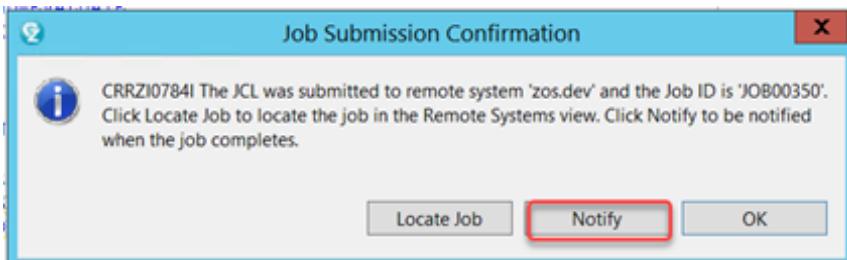
- | If you get a message that files are updated click **Yes** to continue

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

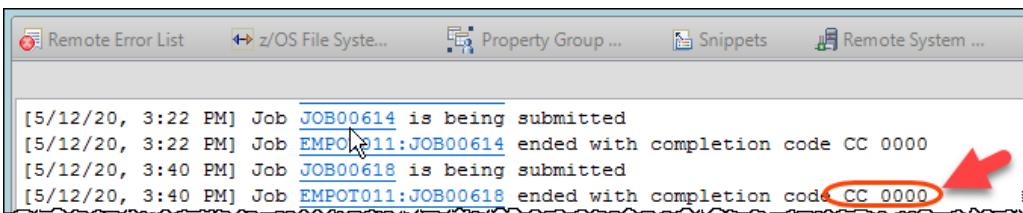


4.2.4 A Job Is submitted for batch execution..

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

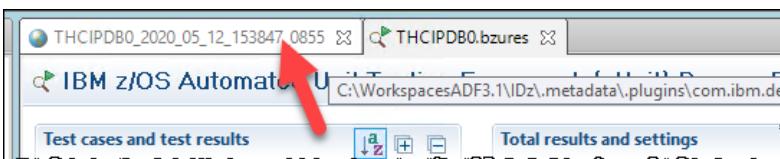


- Make sure the job completes with completion code of **0**.



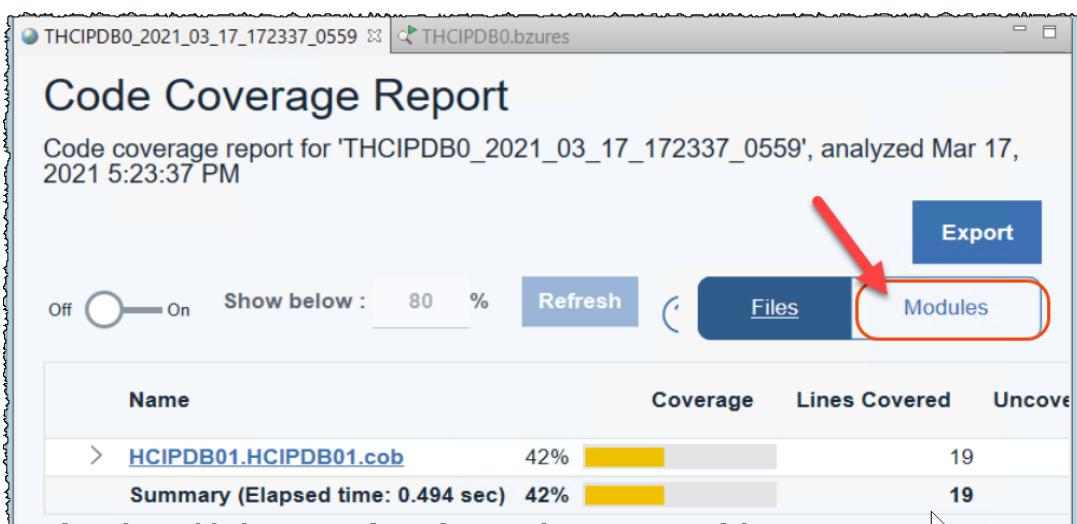
4.2.5 Once again the zUnit Results are displayed and the Test passed.

- 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**:

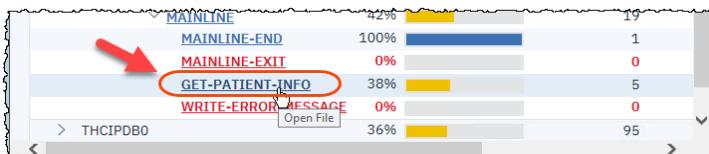


4.2.7 ► Expand the nodes as below to see the COBOL paragraphs.

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 window for 'HCIPDB0.HCIPDB01.cob'. The code is as follows:

```

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. 00
260          ... 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

```

A yellow speech bubble labeled 'lines covered' points to the green-colored lines 247 through 255. Two red circles highlight specific lines: one on line 259 and another on line 262. A red box surrounds the text 'Lines 259-266 not covered. 00'.

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 to 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.

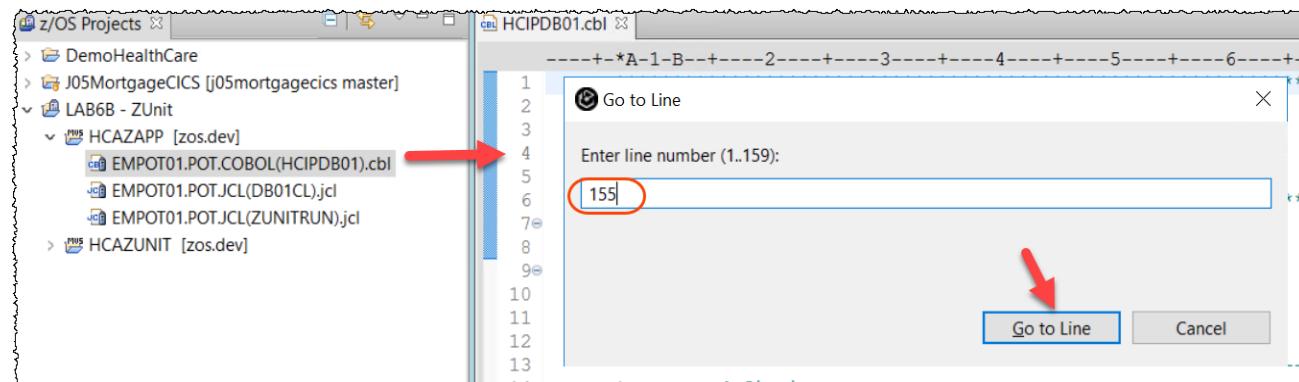
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

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.

```

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      *

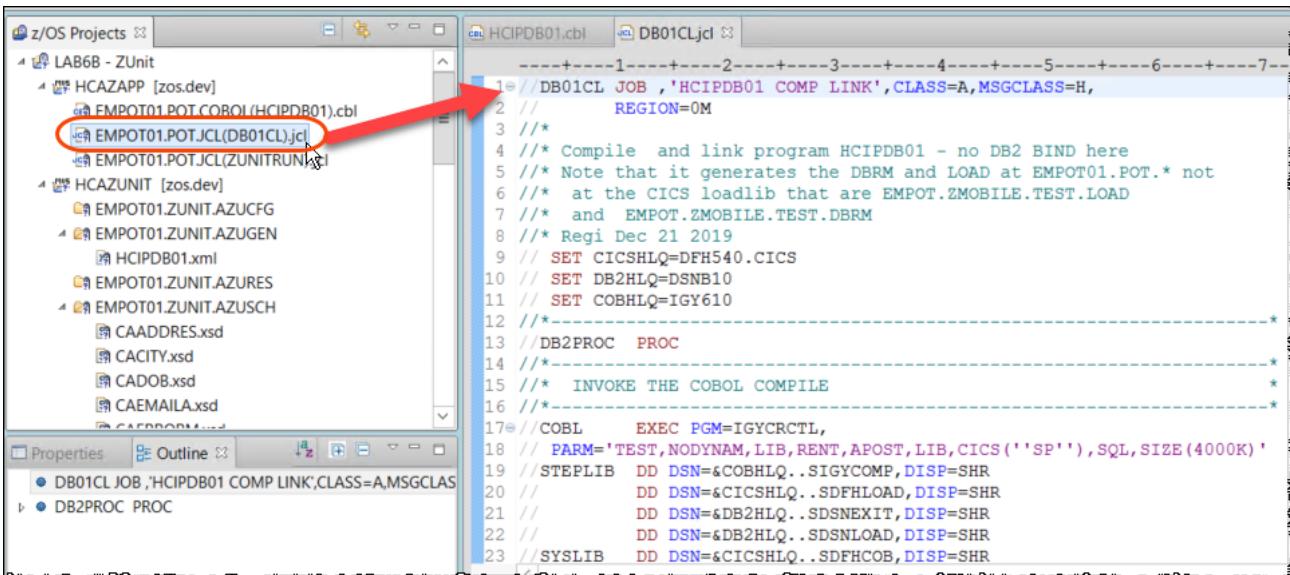
```

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.



```

z/OS Projects : LAB6B - ZUnit
  HCAZAPP [zos.dev]
    EMPOT01.POT.COBOL(HCIPDB01).cbl
    EMPOT01.POTJCL(DB01CL).jcl (highlighted)
    EMPOT01.POTJCL(ZUNITRUN)X
  HCAZUNIT [zos.dev]
    EMPOT01.ZUNIT.AZUCFG
    EMPOT01.ZUNIT.AZUGEN
    HCIPDB01.xml
    EMPOT01.ZUNIT.AZURES
    EMPOT01.ZUNIT.AZUSCH
      CAADDRES.xsd
      CACITY.xsd
      CADOB.xsd
      CAEMAILA.xsd
      CAZUNIT.xsd
Properties : Outline : 
  DB01CL JOB , 'HCIPDB01 COMP LINK',CLASS=A,MSGCLAS
  DB2PROC PROC

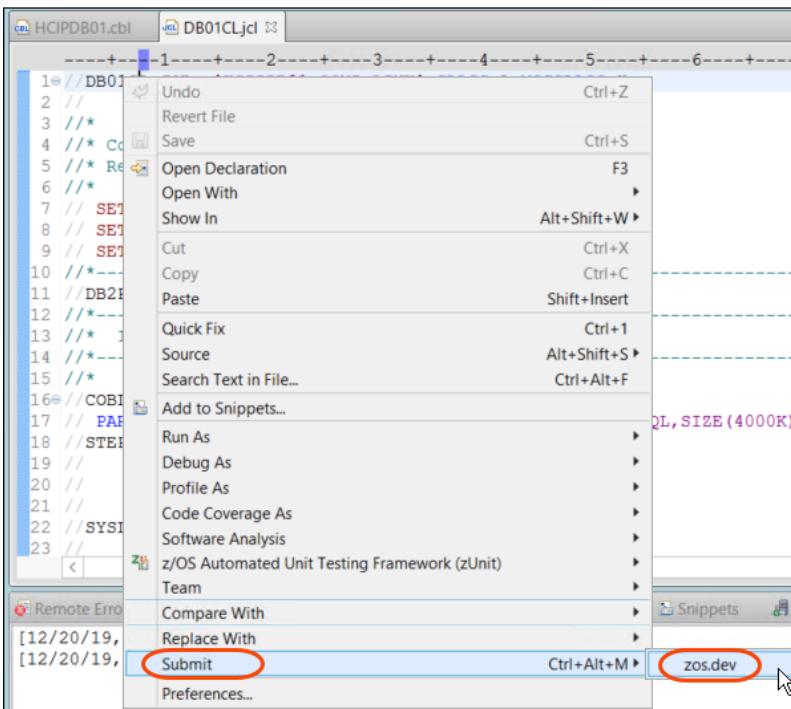
```

```

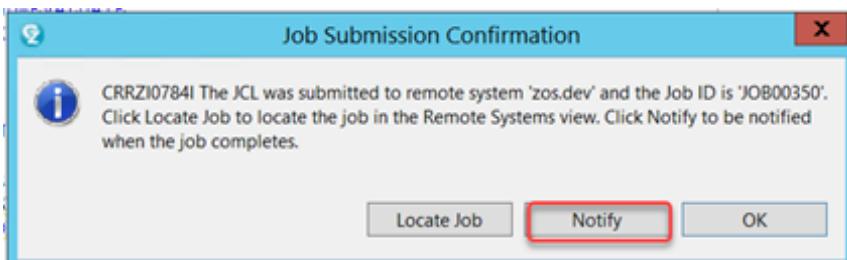
1 //DB01CL JOB , 'HCIPDB01 COMP LINK',CLASS=A,MSGCLAS
2 // REGION=0M
3 /**
4 //** Compile and link program HCIPDB01 - no DB2 BIND here
5 //** Note that it generates the DBRM and LOAD at EMPOT01.POT.* not
6 //** at the CICS loadlib that are EMPOT.ZMOBILE.TEST.DBRM
7 //** and EMPOT.ZMOBILE.TEST.DBRL
8 //** Regi Dec 21 2019
9 // SET CICSHLQ=DFH540.CICS
10 // SET DB2HLQ=DSNB10
11 // SET COBHLQ=IGY610
12 /**
13 //DB2PROC PROC
14 /**
15 //** INVOKE THE COBOL COMPILE
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

```

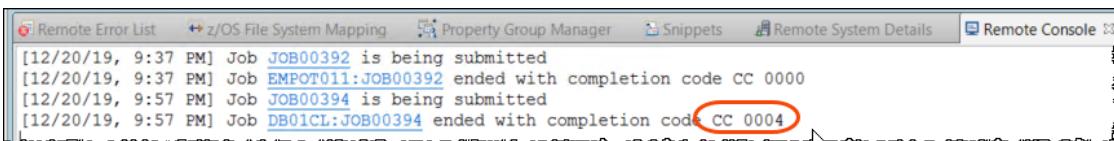
- 5.2.2 ► Right click and select **submit > zos.dev** to submit this job for execution



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

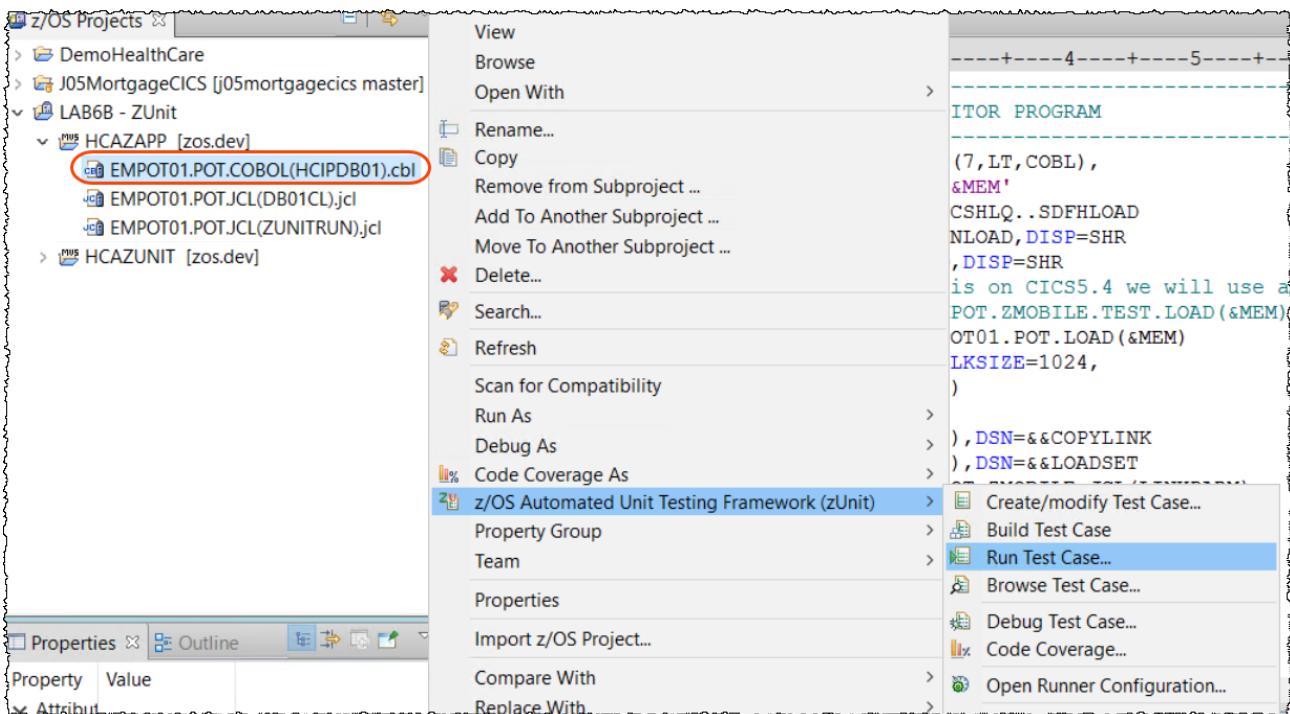


Make sure the job completes with completion code of 4.

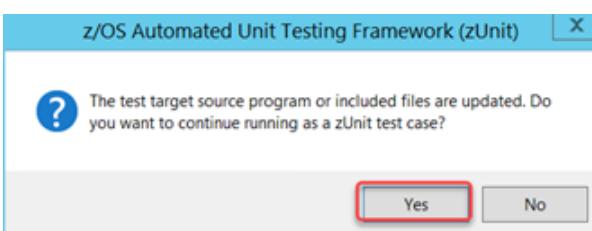


5.3 Rerunning the test case

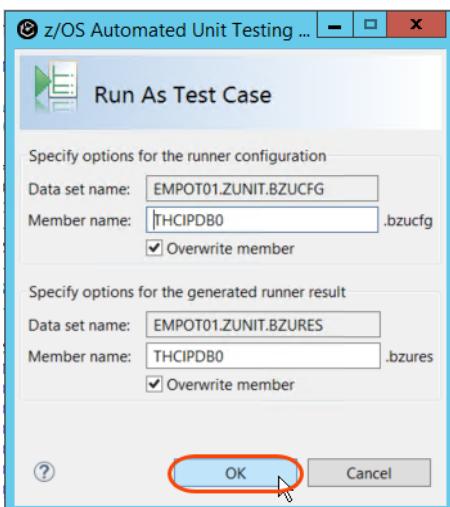
5.3.1 ► On the z/OS Projects view, select **EMPOT01.POT.COBOL(HCIPDB01).cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..** action.



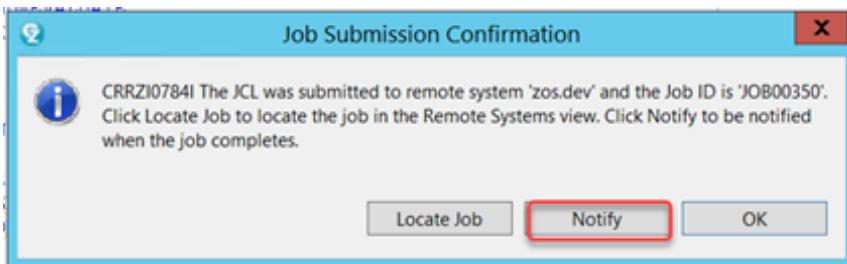
5.3.2 ► Click **Yes** to the following dialog to continue running the test case.



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



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

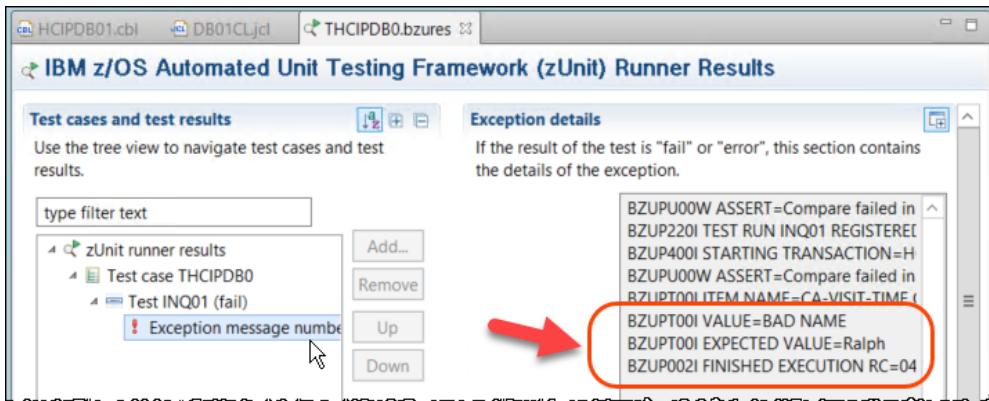


5.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

► Click **Test case 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

5.3.6 ► 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



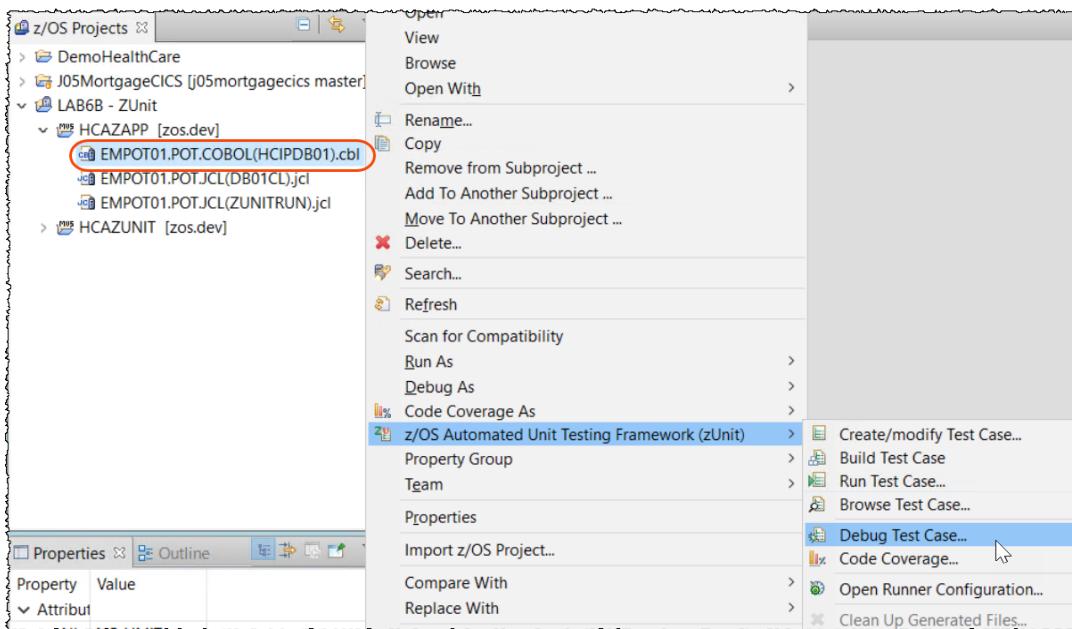
5.4 Running zUnit test case with debugging

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.

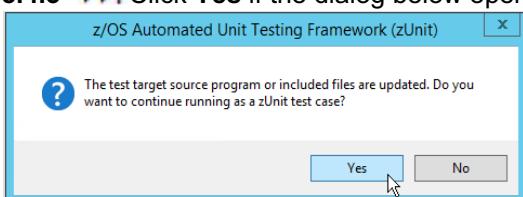
Notice that you are debugging a batch job without need to have CICS and DB2 active, what is very handy.

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

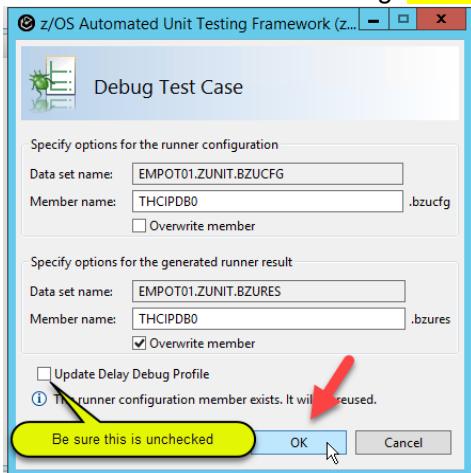
5.4.2 ► Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > Debug Test Case...**



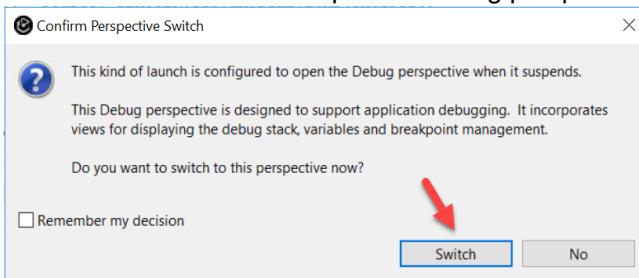
5.4.3 ►| Click Yes if the dialog below opens



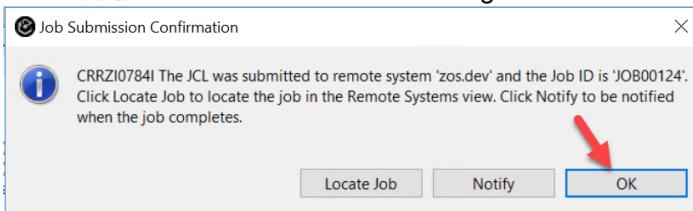
5.4.4 ► Click **OK** for this dialog. Be sure that Update Delay Debug Profile is un-checked.



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

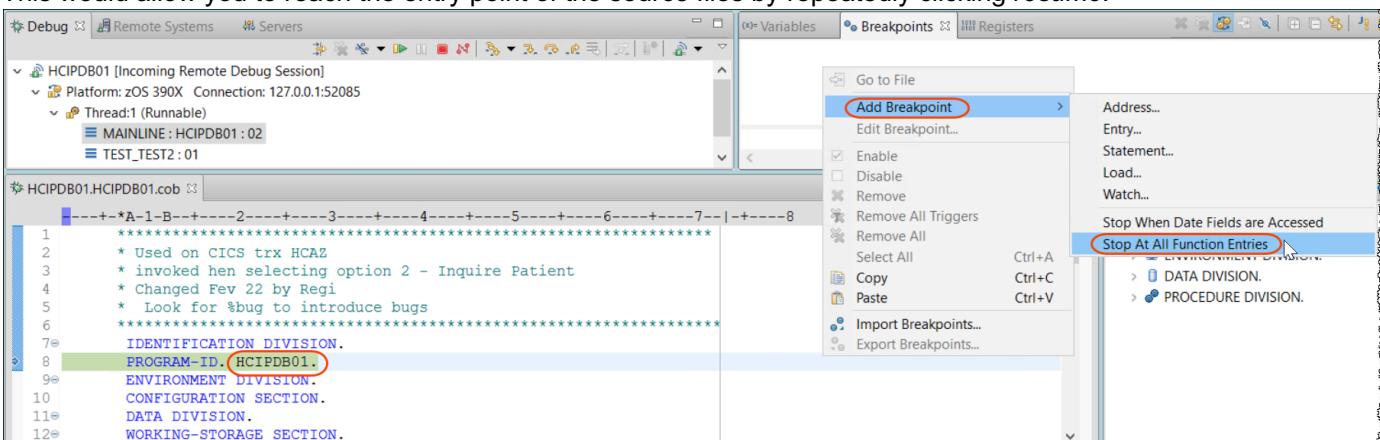


5.4.6 ► Click **OK** to dismiss this dialog

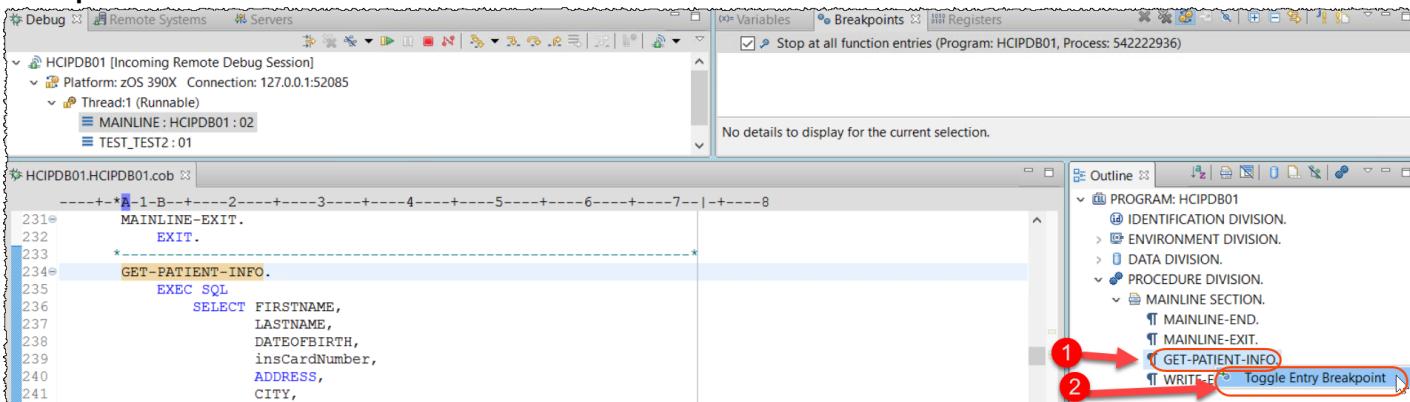


5.4.7 Notice that the first program being debugged is the program that you introduced the bug. (**Hcipdb01**) The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

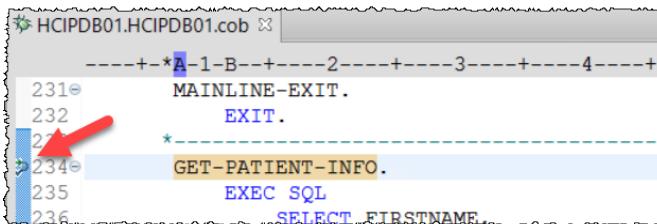
► Using the Breakpoints view, right click and select **Add Breakpoints > Stop At All Functions Entries**. This would allow you to reach the entry point of the source files by repeatedly clicking resume.



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



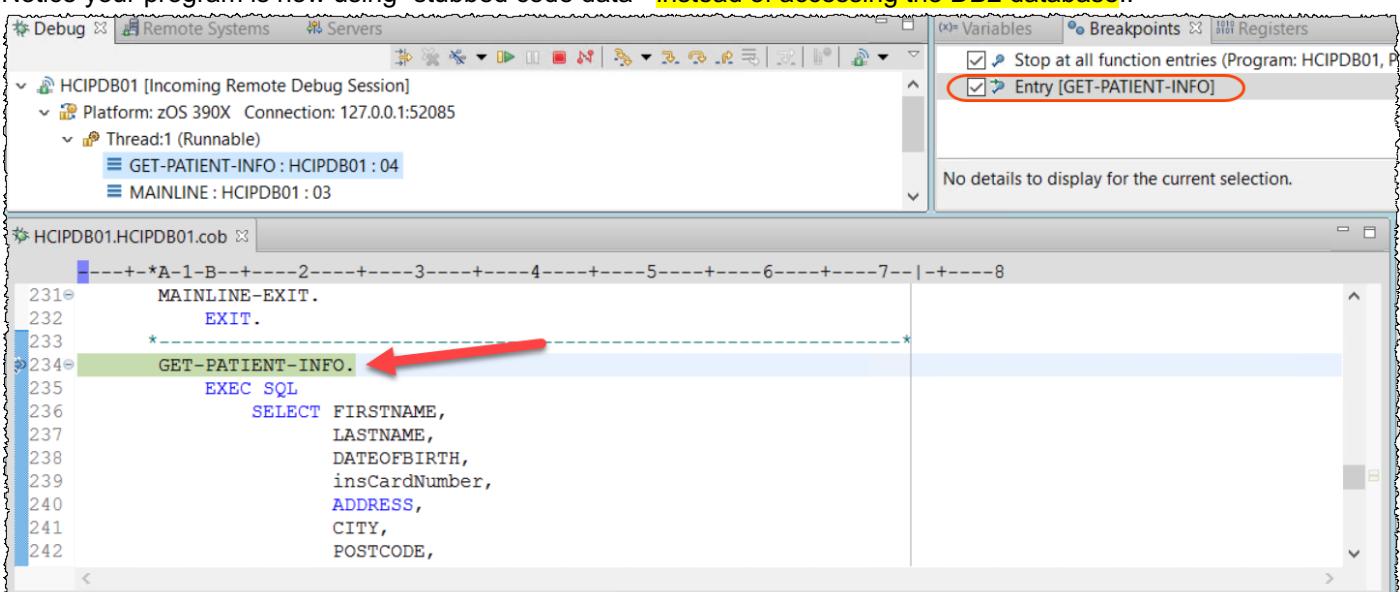
5.4.9 A breakpoint is created on the **EXEC SQL SELECT** statement. When the program runs it will stop there.



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).

```

256      FROM PATIENT
257      WHERE PATIENTID = :DB2-PATIENT-ID
258      END-EXEC.
259      Evaluate SQLCODE
260      When 0
261          MOVE '00' TO CA-RET
262      When 100
263          MOVE '01' TO CA-RET
264      When -913
265          MOVE '01' TO CA-RET
266      When Other
267          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**".

```

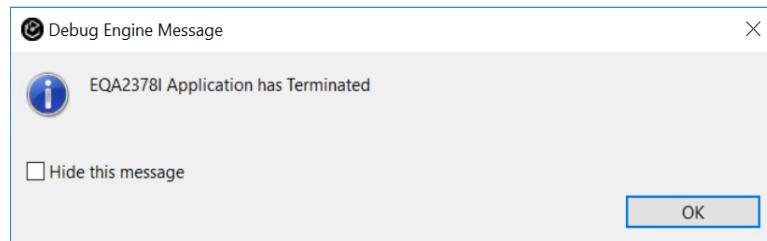
267      END-Evaluate.
268      *%bug -- the line below will introduce a BUG
269      *
270      MOVE "BAD NAME" to CA-FIRST-NAME
271      *
272      EXIT.
273      *
274      *COPY HCERRSPD.
275      *=====
276      * Procedure to write error m
277      *   message will include Dat
278      *   Medication Id and SQLCOD

```

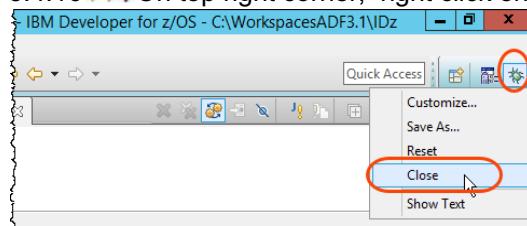
5.4.14 ►| Click on icon or press **F8** to resume.. Or if you prefer keep clicking on *Step Into*.



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

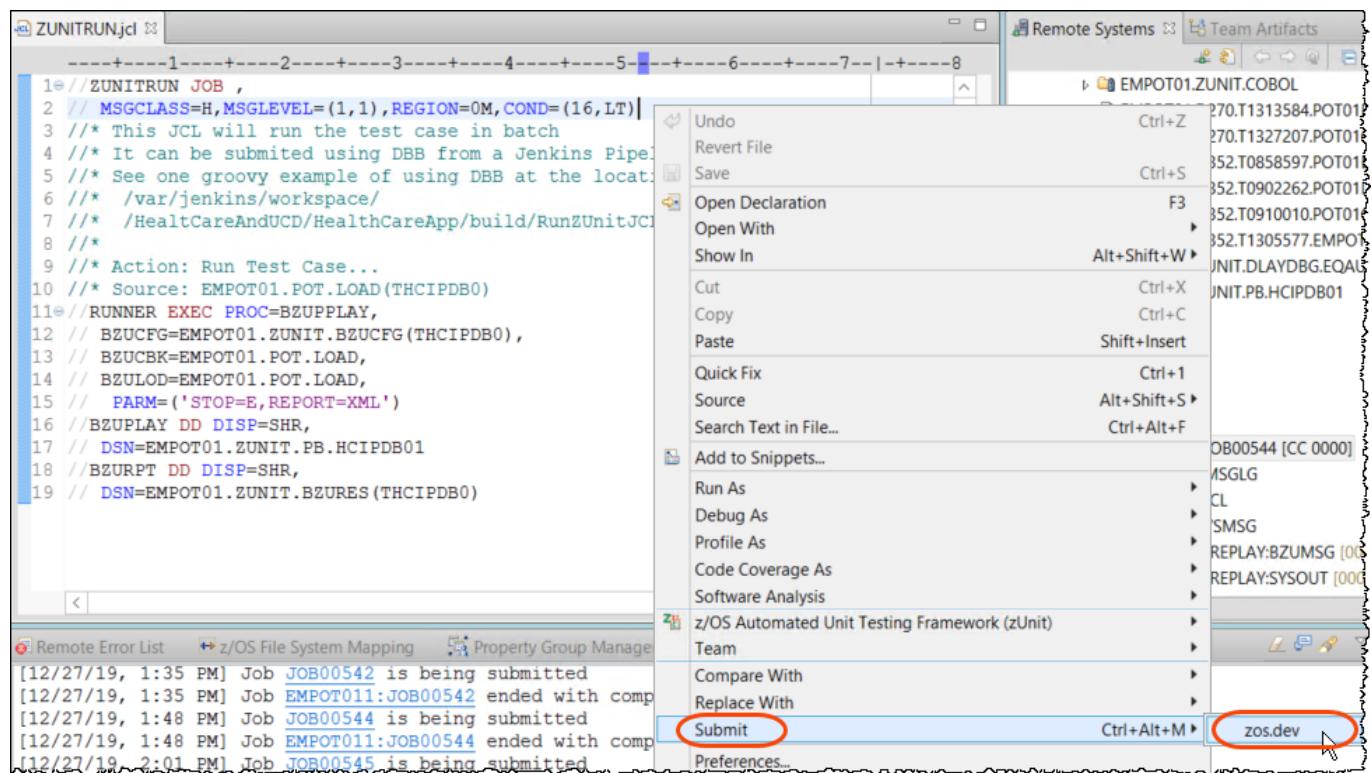
6.1.1 ►| Close any active windows by pressing **Ctrl+Shift+F4**.

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.

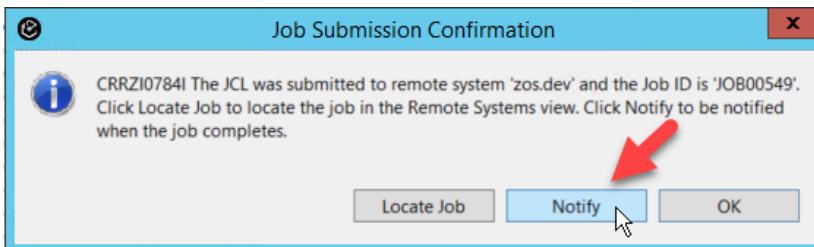
The screenshot shows the IBM Rational Application Developer interface. On the left, the 'z/OS Projects' view displays a tree structure of projects and files. A red arrow points from the 'z/OS Projects' view to the 'ZUNITRUN.jcl' file in the main editor window. The 'ZUNITRUN.jcl' file contains JCL code for running a test case. The code includes comments explaining the purpose of the job, the submission method (DBB from Jenkins Pipeline), and the specific parameters for the test case (STOP=E, REPORT=XML). It also specifies the use of BZUPLAY and BZURPT DD statements with SHR and DSN options.

```
1 //ZUNITRUN JOB ,  
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)  
3 /* This JCL will run the test case in batch  
4 /* It can be submitted using DBB from a Jenkins Pipeline  
5 /* See one groovy example of using DBB at the location below  
6 /* /var/jenkins/workspace/  
7 /* /HealtCareAndUCD/HealthCareApp/build/RunZUnitJCL.groovy  
8 /*  
9 /* Action: Run Test Case...  
10 /* Source: EMPOT01.POT.LOAD(THCIPDB0)  
11 //RUNNER EXEC PROC=BZUPPLAY,  
12 // BZUCFG=EMPOT01.ZUNIT.BZUCFG(THCIPDB0),  
13 // BZUCBK=EMPOT01.POT.LOAD,  
14 // BZULOD=EMPOT01.POT.LOAD,  
15 // PARM='STOP=E,REPORT=XML'  
16 //BZUPLAY DD DISP=SHR,  
17 // DSN=EMPOT01.ZUNIT.PB.HCIPDB01  
18 //BZURPT DD DISP=SHR,  
19 // DSN=EMPOT01.ZUNIT.BZURES(THCIPDB0)
```

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

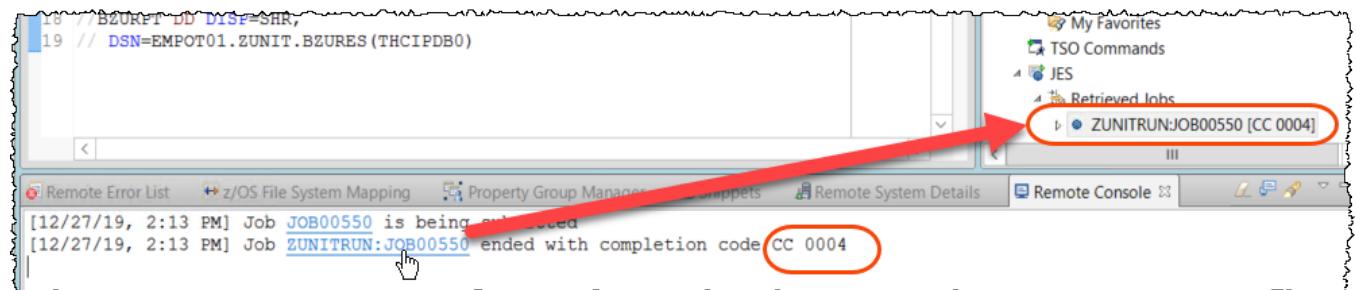


6.1.4 ► Click **Notify** on the Job Submission Confirmation dialog.



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

► Click on **ZUNITRUN:JOB00xxx**



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

```

1 TEST_INQ01 STARTED...
2 CALL HCIPDB01
3 DB2_INPT ...
4 DB2_OUTP ...
5 CICS_0E08_HCIPDB01 CHECK VALUES...
6 EXEC CICS RETURN X'0000' L=00227
7 AREA ALLOCATED FOR RECORD COUNT:00000048
8 CICS_0E08_HCIPDB01 SUCCESSFUL.
9 ****
10 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
15 ****
16 TEST_INQ01 SUCCESSFUL.
17

```

**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

```

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 */
28def confdir = "/var/dbb/1.0.7/conf"
29
30/* The data set contains the ZUnit JCL */
31/* For example: IBMUSER.ZUNIT.JCL */
32def jclDataset = "IBMUSER.ZUNIT.JCL"

```

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

Congratulations! You have completed the Lab 3A.

LAB 3B – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 program using Local assets (60 minutes)

Updated November 17, 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

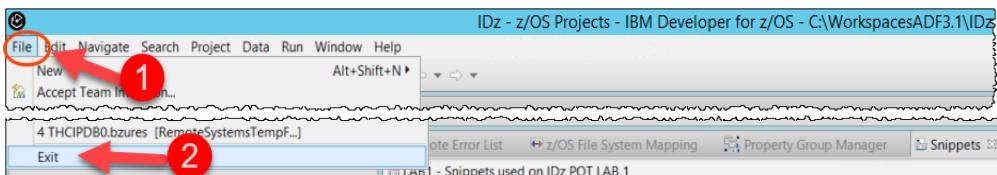
You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be interacting with.

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*

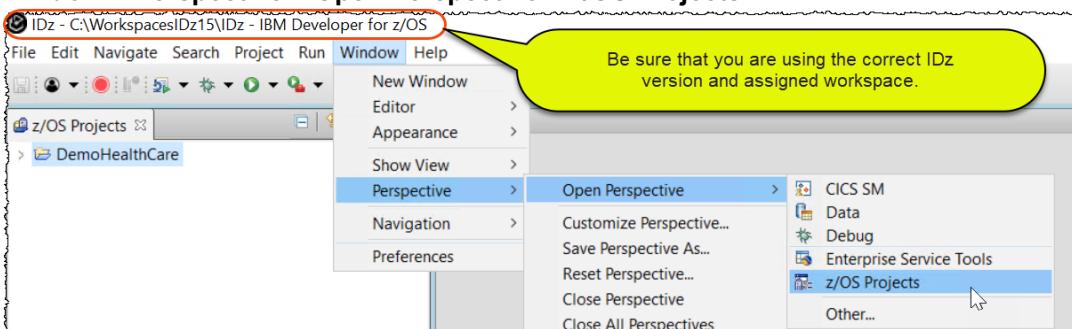
► Using the desktop double click on **IDz V15** icon.

► 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**.

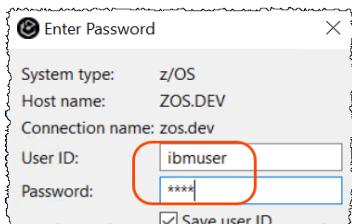
If you are already connected as ibmuser, jump to step 1.1.5 Otherwise disconnect from z/OS and reconnect

► 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.



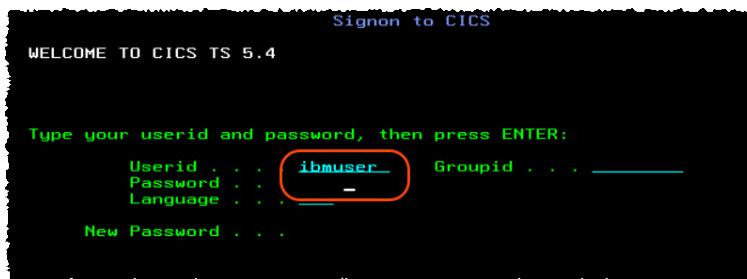
1.1.5 Wait until connection is complete (on the bottom and left until the green bar disappears)

►| 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**.

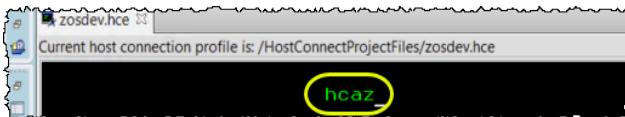


1.1.9 The sign-on message is displayed

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.

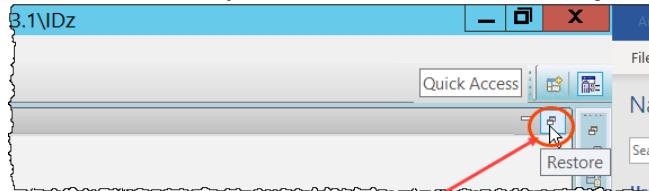


You may also use the Reset key after clicking NextPad)

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.

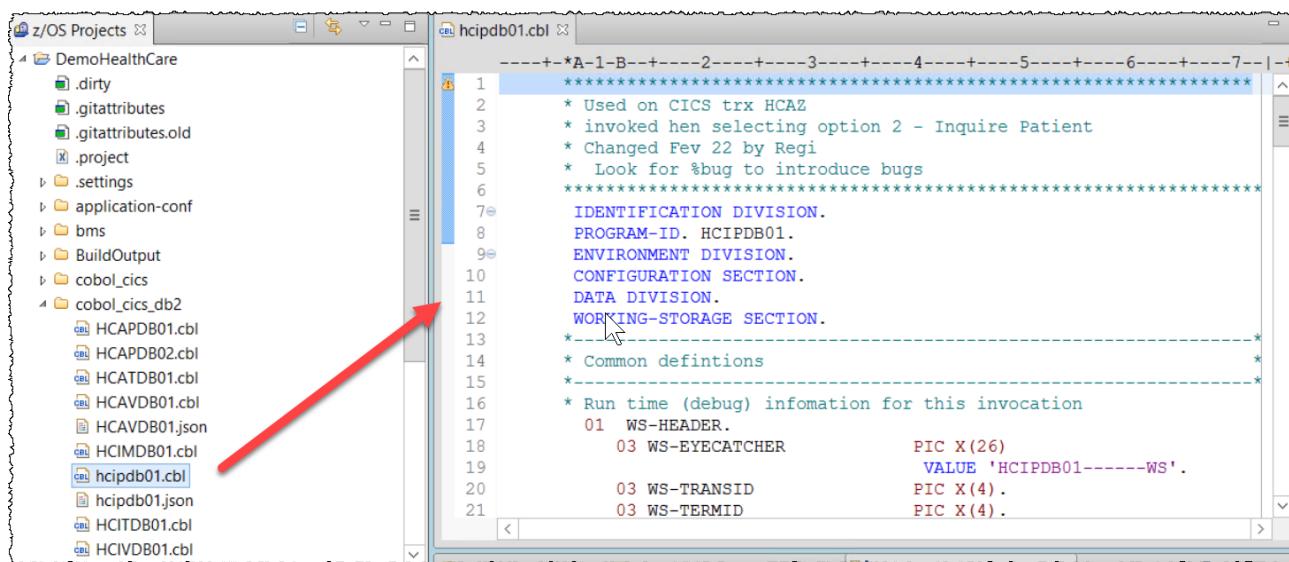
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

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 **hcindb01cbl** 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. On the left, the 'z/OS Projects' view displays a project structure for 'DemoHealthCare' with various source files like .dirty, .gitattributes, .gitattributes.old, .project, .settings, application-conf, bms, BuildOutput, cobol_cics, and cobol_cics_db2. The 'cobol_cics_db2' folder contains several COBOL programs: HCAPDB01.cbl, HCAPDB02.cbl, HCATDB01.cbl, HCAVDB01.cbl, and HCACDB01.cbl. The 'Properties' and 'Outline' tabs are selected in the bottom-left toolbar.

The main editor window on the right displays the COBOL code for 'HCAPDB01.cbl'. A red box highlights the following code block:

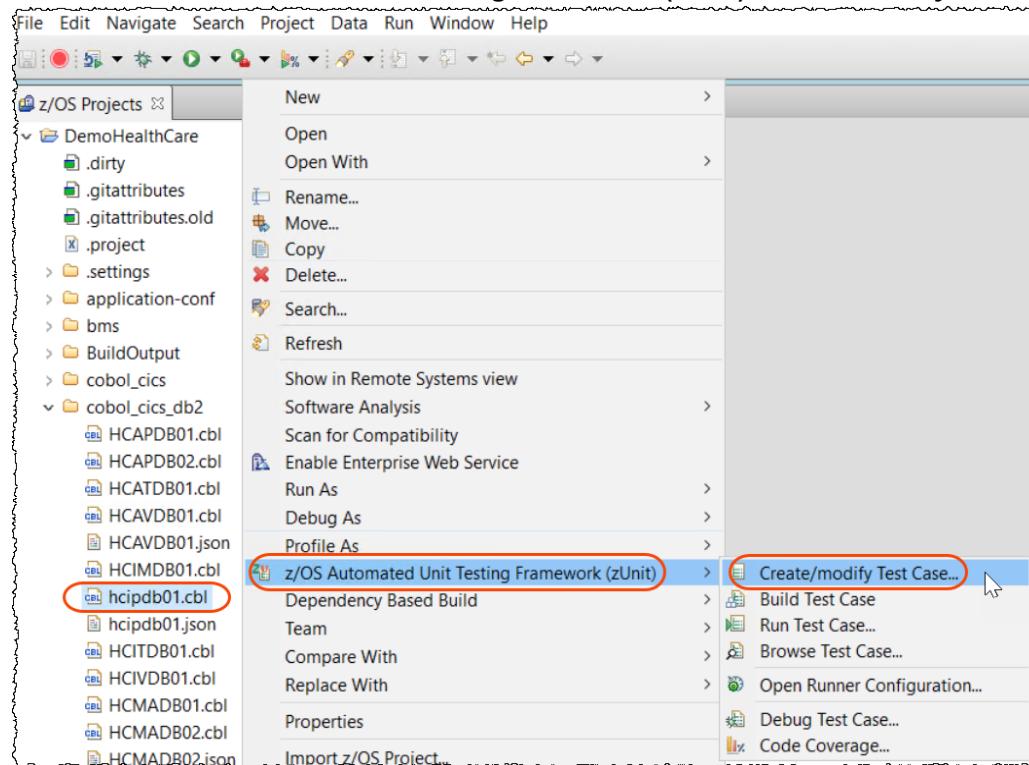
```
MAINLINE-END.  
EXEC CICS RETURN END-EXEC.  
MAINLINE-EXIT.  
EXIT.  
*  
GET-PATIENT-INFO.  
EXEC SQL  
SELECT FIRSTNAME,  
LASTNAME,  
DATEOFBIRTH,  
insCardNumber,  
ADDRESS,  
CITY,  
POSTCODE,  
PHONE_MOBILE,  
EMAILADDRESS,  
USERNAME  
INTO :CA-FIRST-NAME,  
:CA-LAST-NAME,  
:CA-DOB,
```

A red arrow points from the 'Properties' tab in the toolbar to the 'Name' column of the 'Property Group Mappings' table at the bottom of the screen.

Name	Description
LAB1_Remote_COBOL	Property Group
AZUPGCOB_CICS_J05P	Property group
AZUPGCOB_CICS_HCAZ_EMPOT01	Property group
OLD_AZUPGCOB_CICS_HCAZ_EMPOT01	Property group

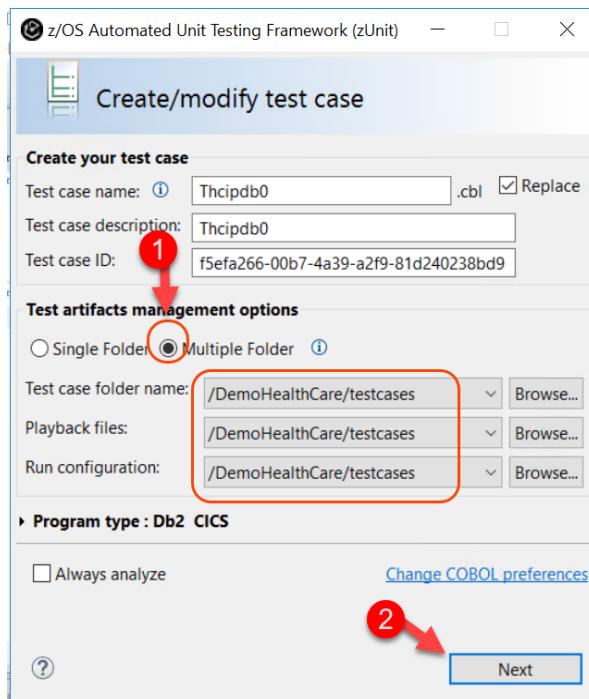
2.2 Recording the COBOL program that sends the message

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..**



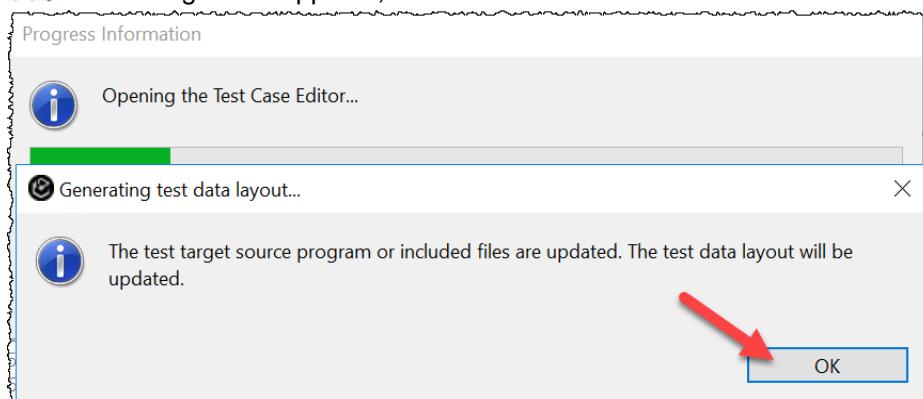
2.2.2 This opens a dialog where you can name your test case.

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



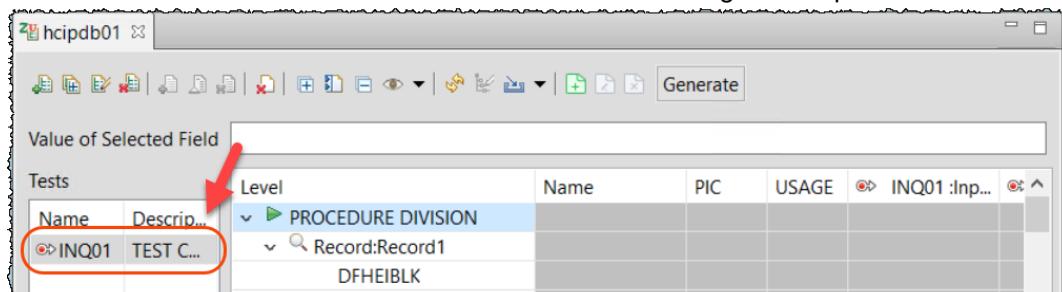
2.2.3 This operation will generate the test data layout on the local workspace.

- ▶ 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.



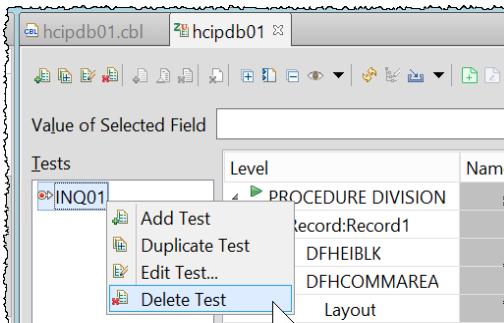
Understanding the test case editor



- 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**



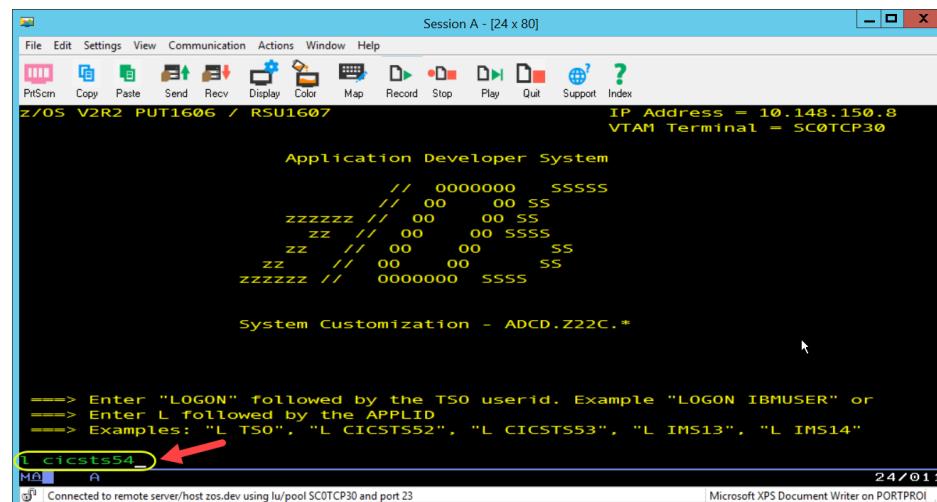
Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

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.



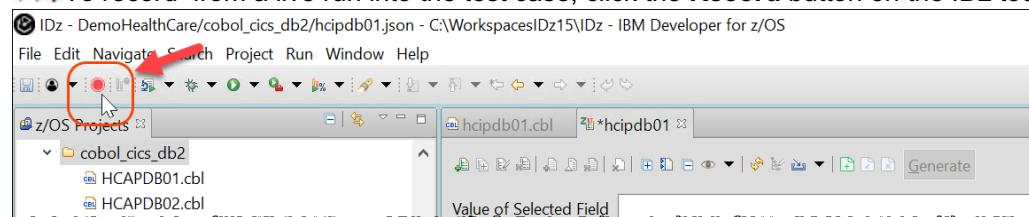
2.2.9 ►| Once you see the sign-on is complete message, enter **hcaz** and press the **Enter** key.



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

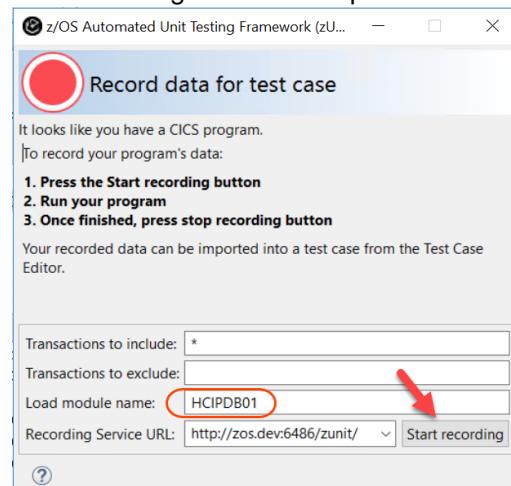
2.2.10 ►| Minimize the terminal emulator and go back to IDz dialog.

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

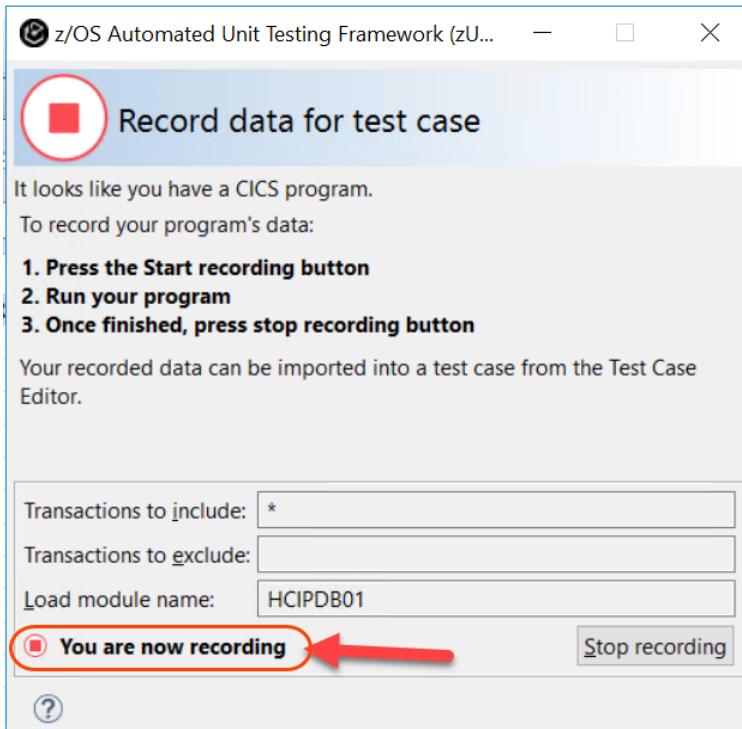


2.2.11 ►| In the dialog that comes up, click on **Start recording**.

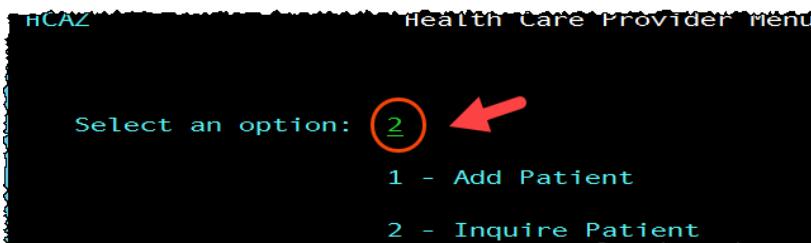
The Recording Service URL points to the CICS region where the live run is recorded.



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.

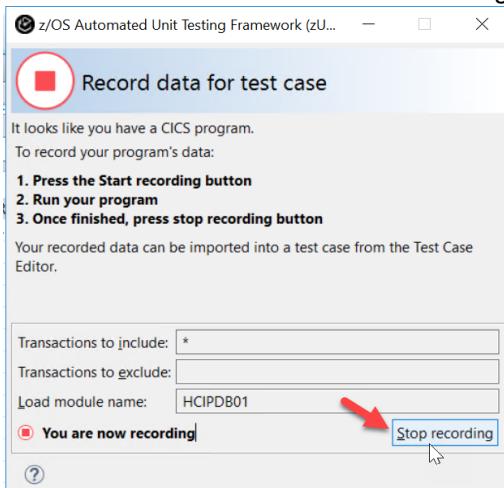


2.2.13 ► Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**. In case your transaction is not running just type **HCAZ** again.



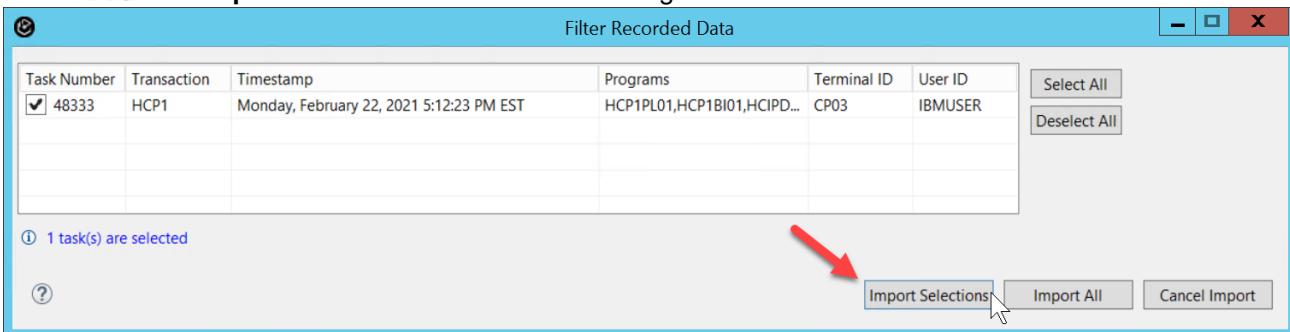
2.2.14 ► Type **1** and press **enter**

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



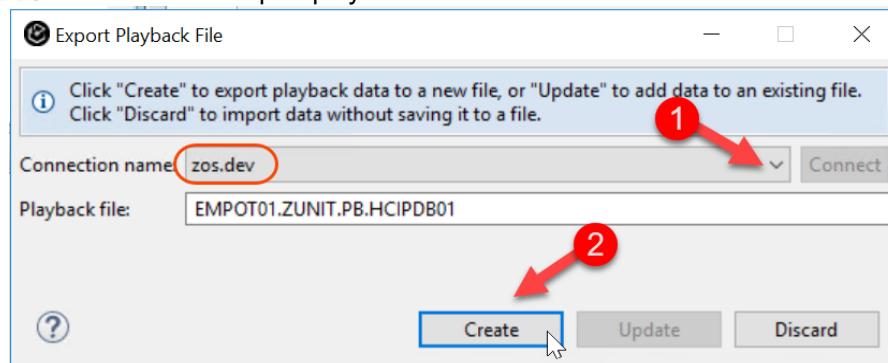
The dialog *Filter Recorded Data* is displayed.

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

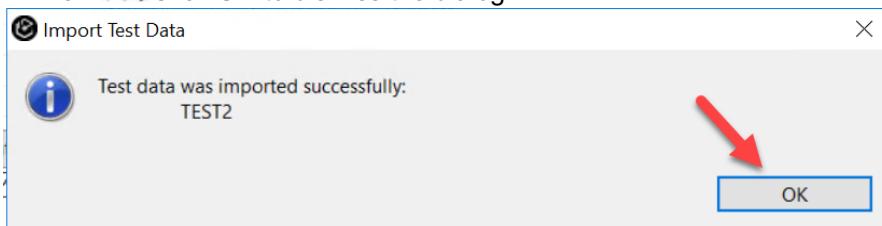


2.2.17 ►| Using drop down select the **zos.dev** as *Connection name*

►| 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

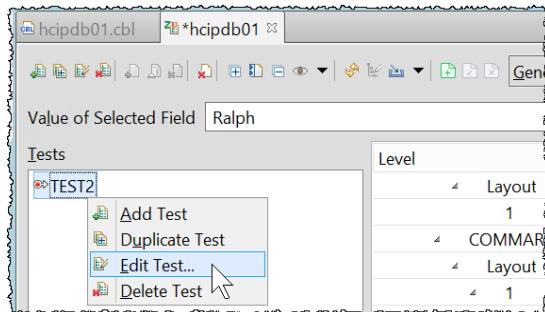


2.2.20 You now see a new test created in the editor and populated with the values from the live run.
 ► Scroll down the editor and notice the data displayed on the screen that was captured..

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"

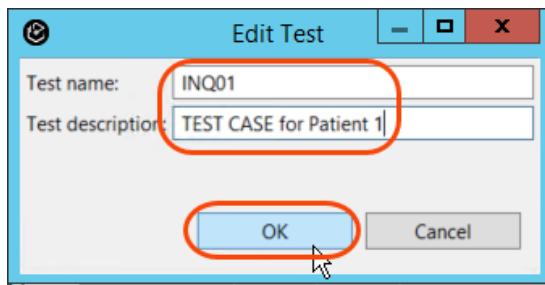
Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
1	HCAZERRS	X(8)	DISPLAY		
1	COMMAREA				
1	Layout				
1	CA-ERROR-MSG				
3	FILLER	X(9)	DISPLAY		
3	CA-DATA	X(90)	DISPLAY		
4	EXEC SQL SELECT INTO [PATI]	line=117			
4	Record:Record1				
4	LineNumber=117				
4	INTO				
5	CA-FIRST-NAME	X(10)	DISPLAY	Ralph	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.

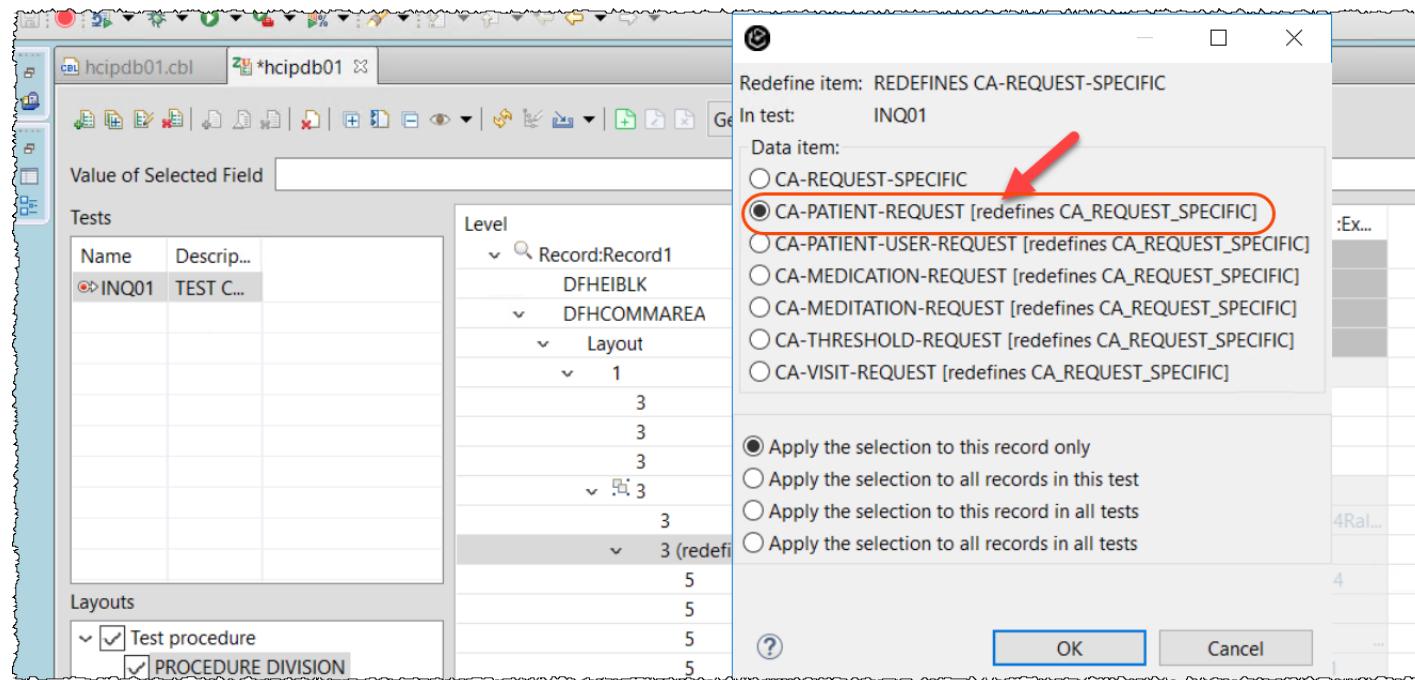
► Use a name like **INQ01** and a description like “**TEST CASE for Patient 1**”: . Click **OK**



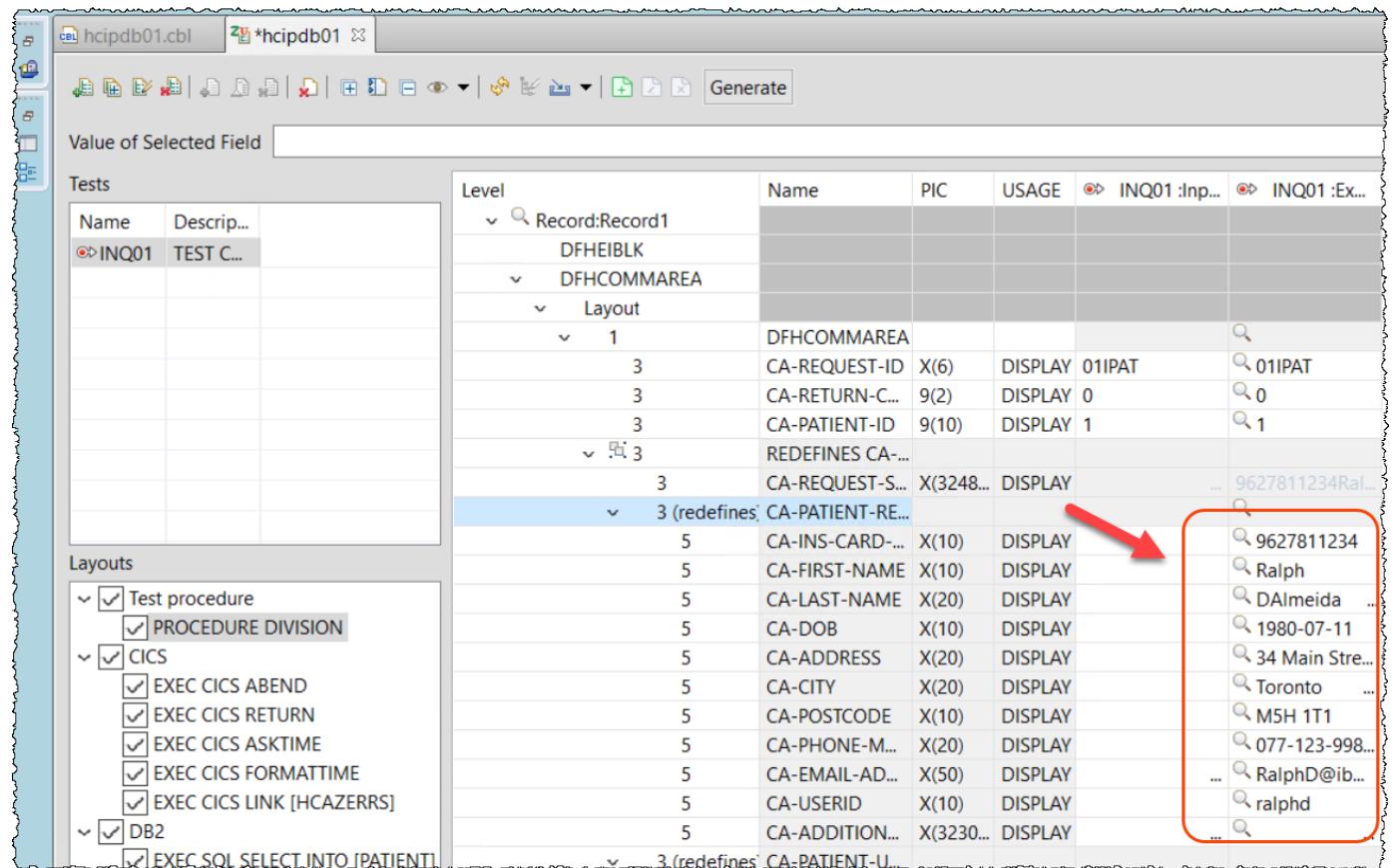
2.2.25 You may notice that this program has many redefines. You may need to identify which is the area being used on this program execution. In our example is the “inquiry patient” area being redefined.

► 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

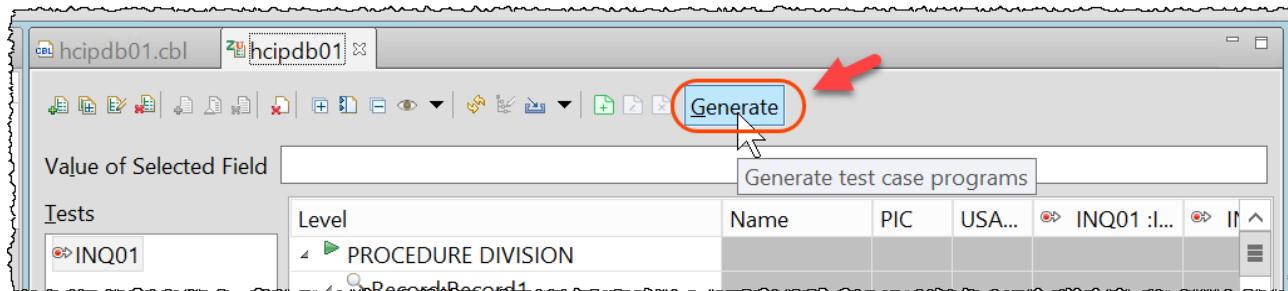


Section 3. Generate, build, and run the unit test.

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

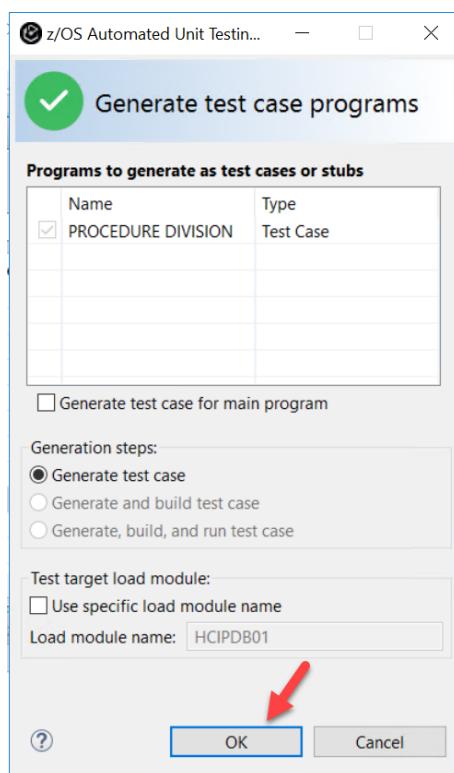
3.1 Generating the test case program.

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.

The screenshot shows the RAD interface with the following details:

- z/OS Projects View:** Shows a tree structure of project files. A red circle highlights the file **Thcipdb0.cbl** under the **testcases** folder.
- Outline View:** Displays the generated COBOL code for **Thcipdb0.cbl**. The code includes:


```

1      -----+-----+-----+-----+-----+-----+-----+-----+
1      *| PROCESS NODLL, NODYNAM, TEST (NOSEP), NOCICS, NOSQL, PGMN (LU)
2      *+
3      *| Thcipdb0
4      *| PRODUCT: IBM DEVELOPER FOR Z/OS
5      *| COMPONENT: IBM Z/OS AUTOMATED UNIT TESTING FRAMEWORK (ZUNIT)
6      *|   FOR ENTERPRISE COBOL AND PL/I
7      *| PROGRAM: ENTERPRISE COBOL ZUNIT TEST CASE FOR DYNAMIC RUNNER
8      *| DATE GENERATED: 03/12/2021 11:45
9      *| ID: f5efa266-00b7-4a39-a2f9-81d240238bd9
10     *+
11    *+-----+
12    *| TEST_INQ01
13    *|   THIS PROGRAM IS FOR TEST INQ01
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.
      
```
- Properties View:** Shows a list of generated programs:
 - PROGRAM: TEST_INQ01
 - PROGRAM: BZU_TEST
 - PROGRAM: BZU_INIT
 - PROGRAM: BZU_TERM
 - PROGRAM: EVALOPT
 - PROGRAM: GTMEMRC
 - PROGRAM: AZU_GENERIC_CICS
 - PROGRAM: AZU_GENERIC_DB2
 - PROGRAM: CICS_0E08_HCIPDB01
- Outline View:** Shows the generated COBOL code for **Thcipdb0.cbl**.
- Bottom Tab Bar:** Includes tabs for Remote Error List, z/OS File System Mapping, Property Group Manager, Snippets, and Remote System.

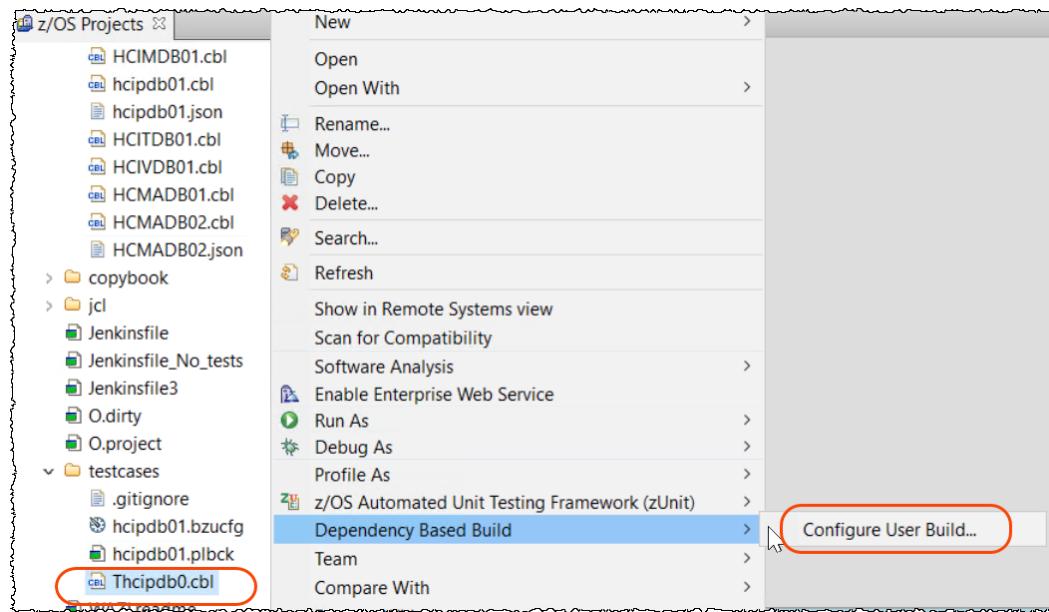
3.2 Building the generated test case programs using IBM DBB .

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.

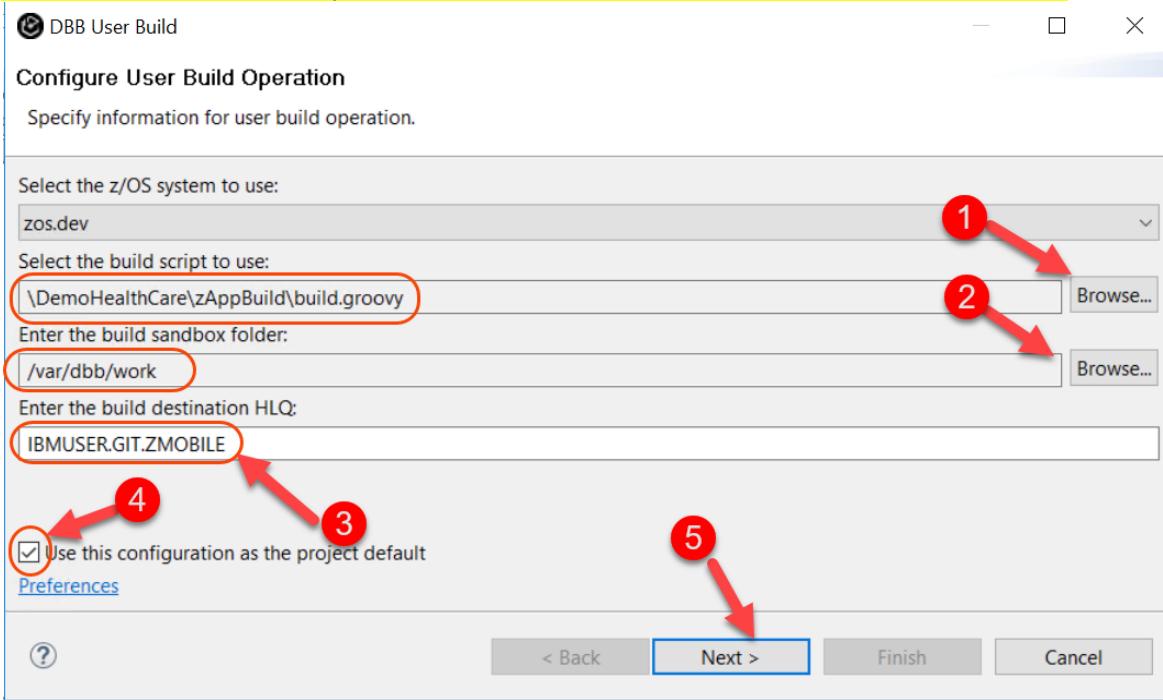
3.2.1 ► Use **Ctrl + Shift + F4** to close all opened editors.

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.xxxxx**



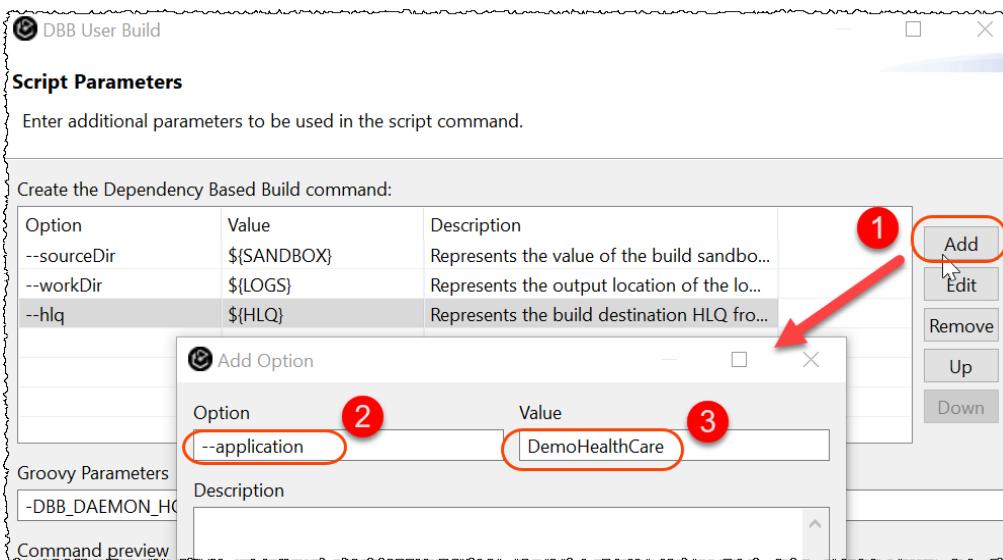
3.2.4 ► Click **Next** 3 times

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.

Option	Value	Description
--sourceDir	`\${SANDBOX}	Represents the value of the build sandbox from the first screen of the zAppBuild script command.
--workDir	`\${LOGS}	Represents the output location of the log folder from the first screen of the zAppBuild script command.
--hlq	`\${HLQ}	Represents the build destination HLQ from the first screen of the zAppBuild script command.
--application	DemoHealthCare	

Add Option

Option	Value
--debug	

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 of the zAppBuild script command.
--workDir	`\${LOGS}	Represents the output location of the log folder from the first screen of the zAppBuild script command.
--hlq	`\${HLQ}	Represents the build destination HLQ from the first screen of the zAppBuild script command.
--application	DemoHealthCare	
--debug		

Groovy Parameters

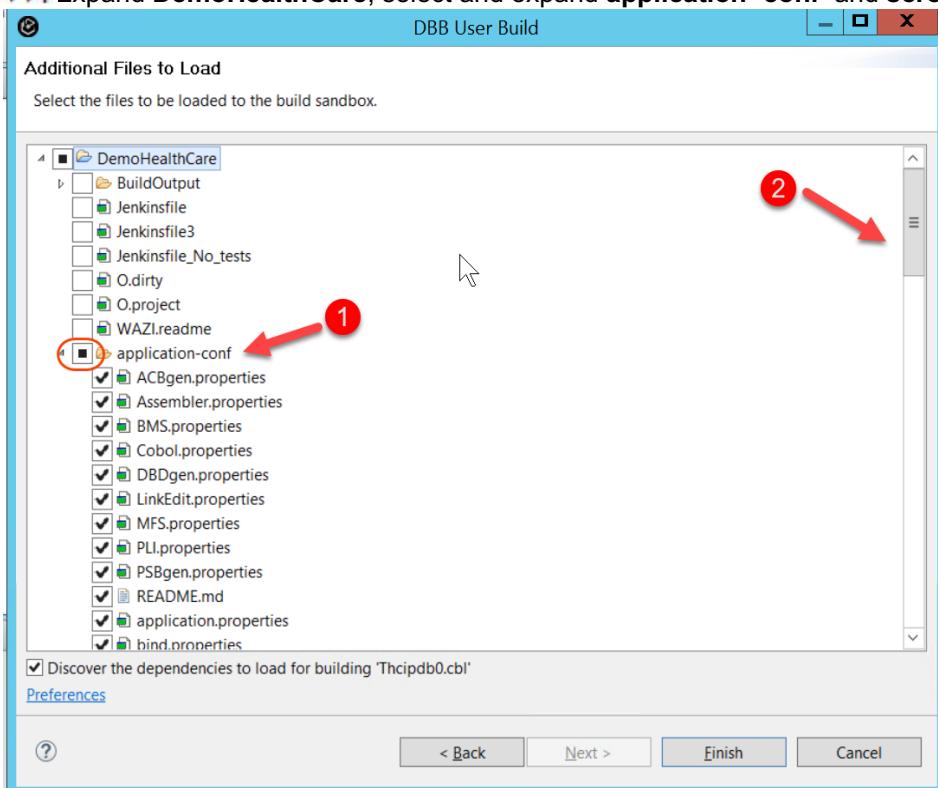
Command preview

```
$DBB_HOME/bin/groovyz /var/dbb/work/DemoHealthCare/zAppBuild/build.groovy --sourceDir /var/dbb/work --workDir /var/dbb/logs --hlq ${HLQ} --application DemoHealthCare --debug
```

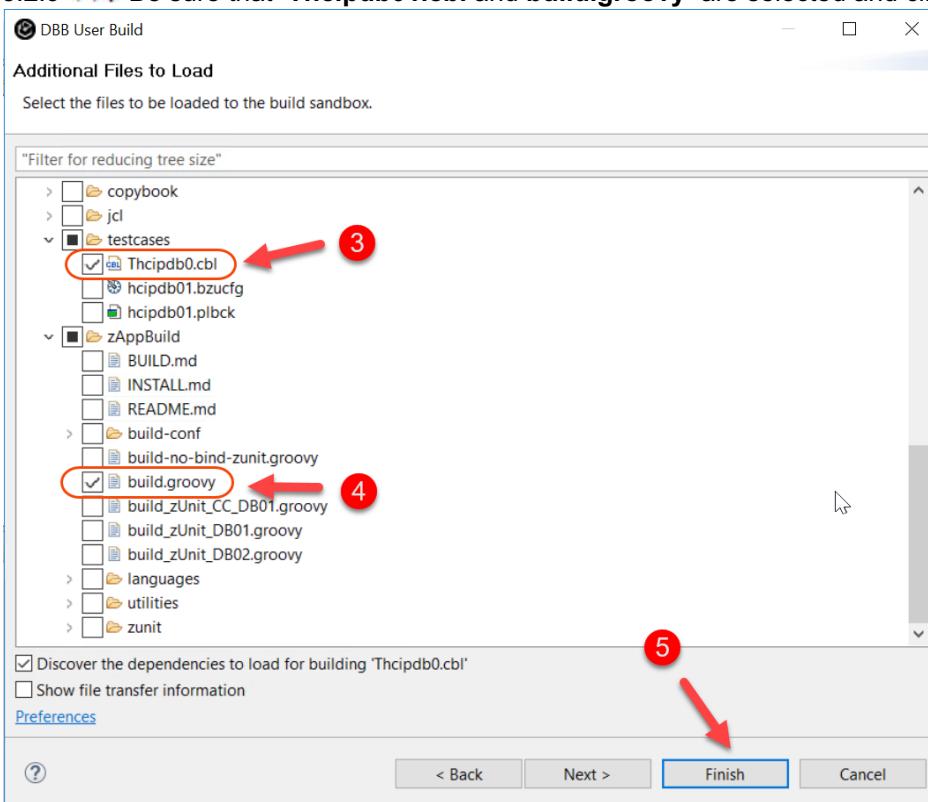
< Back Next > Finish Cancel

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

▶ Expand DemoHealthCare, select and expand **application -conf** and scroll down

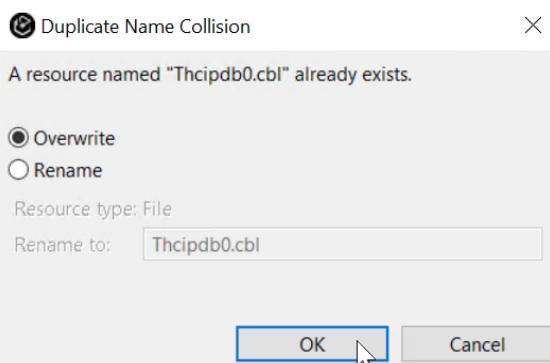


3.2.9 ▶ Be sure that **Thcipdb01.cbl** and **build.groovy** are selected and click **Finish**

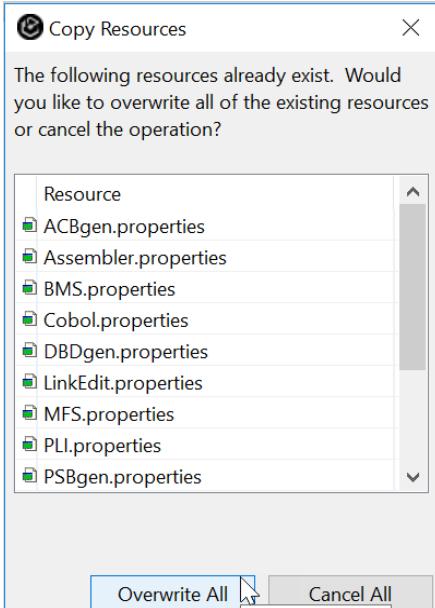


3.2.9 Those assets were once already moved to the z/OS UNIX files and will overwrite the old copies.

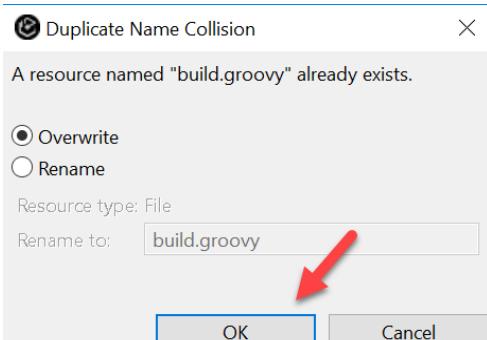
▶ Click **OK**



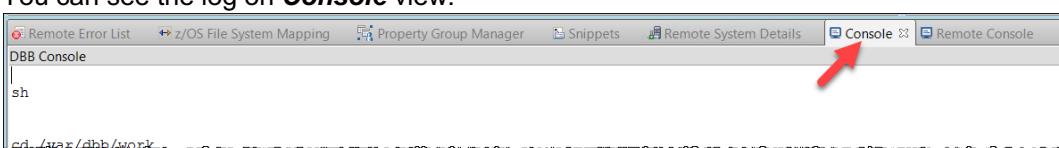
▶ Click **Overwrite All**



▶ 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)

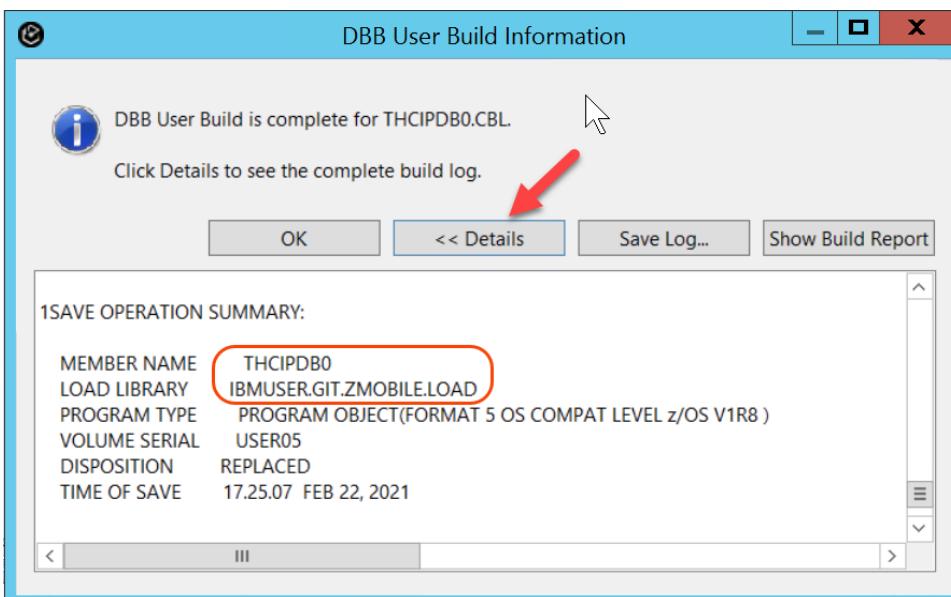


DBB Console
 Cannot run program "sh" (in directory "/var/dbb/work"): EDC5112I Resource temporarily unavailable
 \$DBB_HOME/bin/groovyz -DBB_DAEMON_HOST 127.0.0.1 -DBB_DAEMON_PORT 8080 /var/dbb/work/DemoHealthCare/zAppBuild/build.groovy --sc

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.

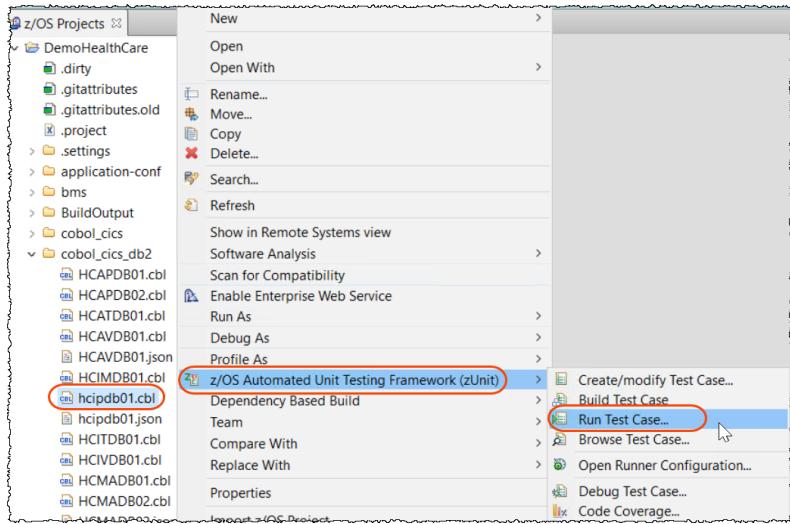


► The **Console view** also show the results as below:

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.

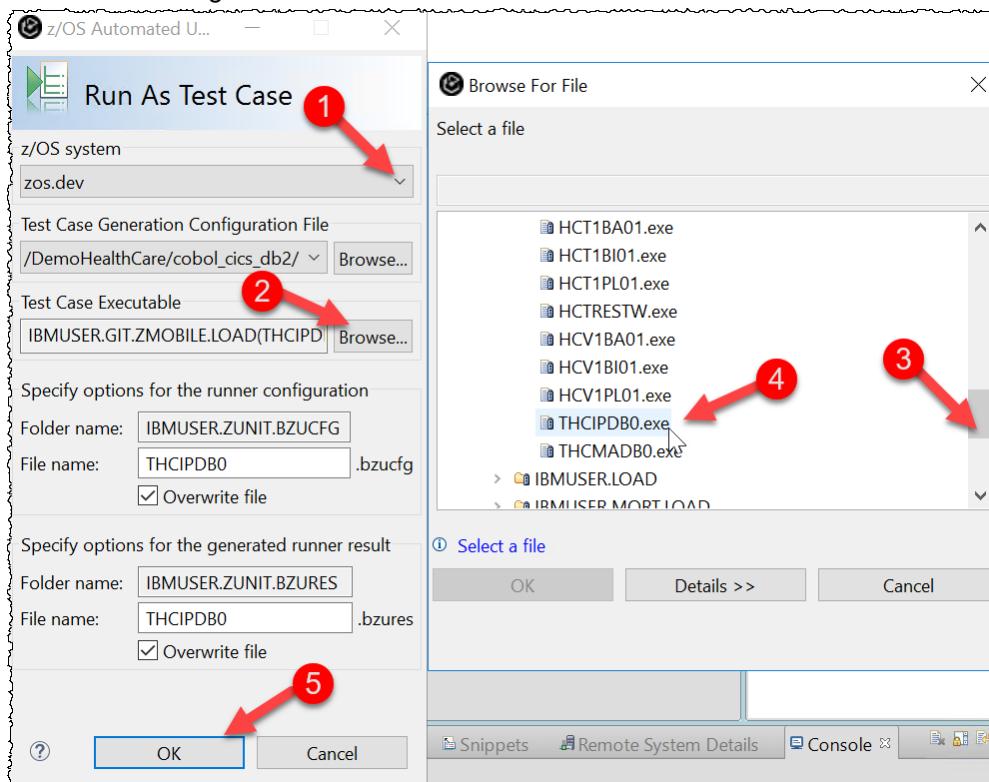
3.3.1 ► Using z/OS Projects view, scroll up, right click on **hcipdb01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > Run Test Case...**



3.3.2 ► Select **zos.dev** and use the **Browse** button to select the PDS and load module where the test case was created.

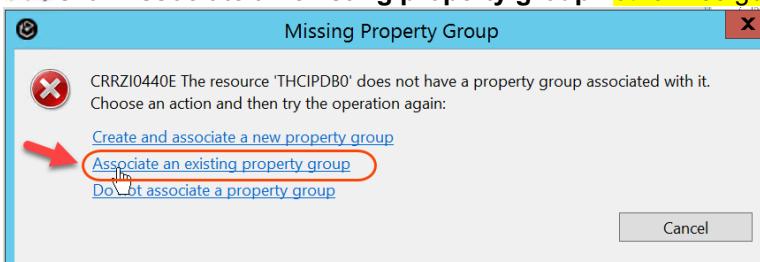
You will find the PDS **IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)** under MVS Files and **My Data Sets ..**

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

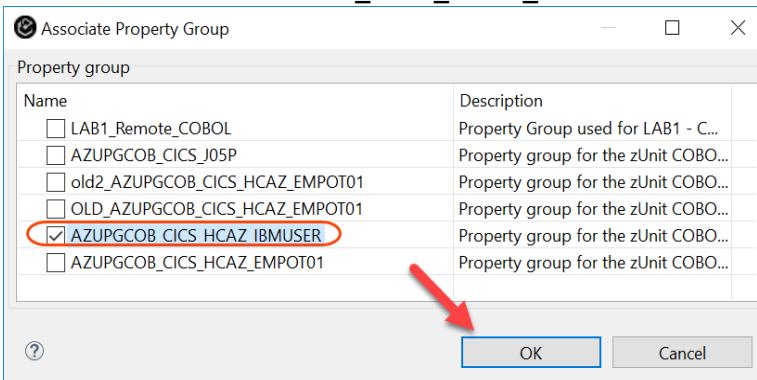


3.3.3 If the *Missing Property Group* dialog will prompt.

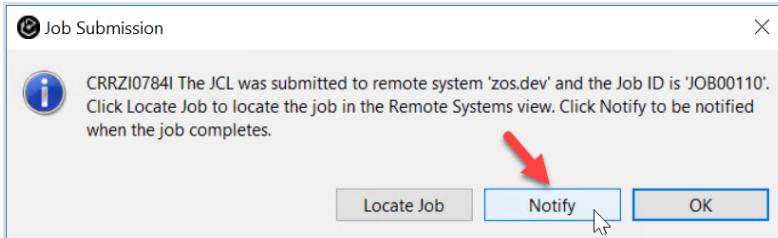
▶ Click **Associate an existing property group** otherwise go to 3.3.5



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.

The screenshot shows the Rational Application Developer interface. On the left is a code editor window titled "JOB00092.jcl" containing JCL code. On the right is the "Remote Systems" view, which includes a "Team Artifacts" tab and a "JES" node under "My Jobs". A red arrow points from the "My Jobs" section of the tree view to the code editor window, indicating where to click to show the JCL.

```

JOB00092.jcl
-----
1 //IBMUSER1 JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
3 /* Action: Run Test Case...
4 //** Source: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)
5 //      SET FELJOB=ZUNITGO
6 //RUNNER EXEC PROC=BZUPPLAY,
7 // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),
8 // BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,
9 // BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,
10 //    PRM='STOP=E,REPORT=XML'
11 //REPLAY.BZUPLAY DD DISP=SHR,
12 // DSN=IBMUSER.ZUNIT.PB1.HCIPDB01
13 //REPLAY.BZURPT DD DISP=SHR,
14 // DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)
15 //STEPLIB DD
16 //      DD DSN=IBMUSER.LOAD,DISP=SHR
17 //      DD DSN=IBMUSER.GIT.ZMOBILE.LOAD,DISP=SHR
18

```

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

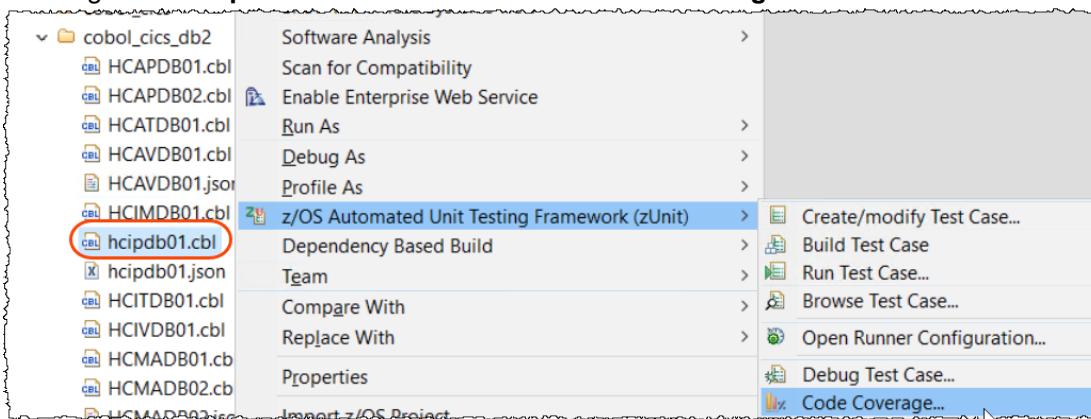
3.5 Running zUnit test case Code Coverage

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.

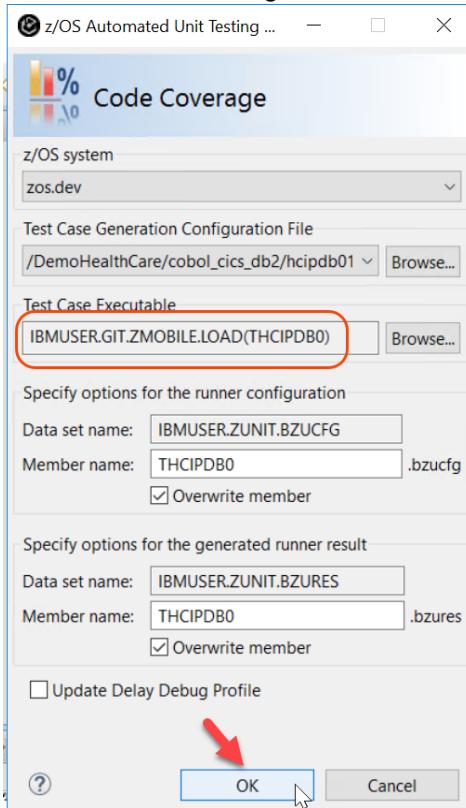
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, which is very handy..

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.



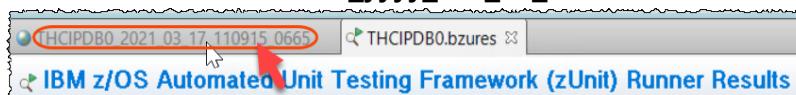
3.5.3 ► Click **Notify** on the Job Submission Confirmation dialog.

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 EMPOT011: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 EMPOT011:JOB00618 ended with completion code CC 0000
```

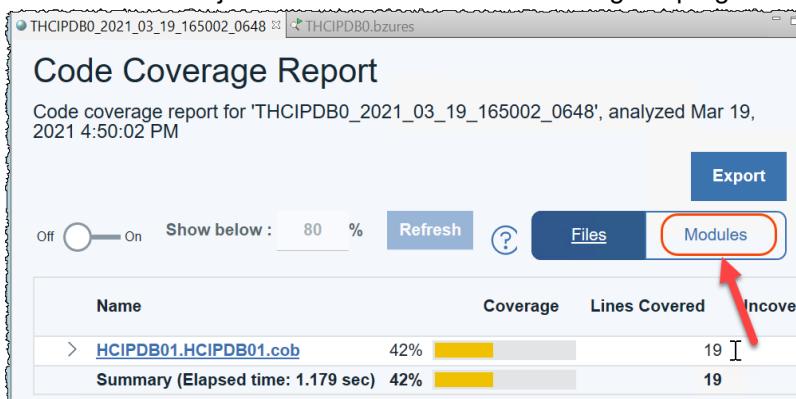
3.5.5 The code coverage report shows all COBOL programs executed for this specific test case.

▶ Click on title **THCIPDB0_yyyy_mm_dd_xxxxxxx** to see the Code coverage report



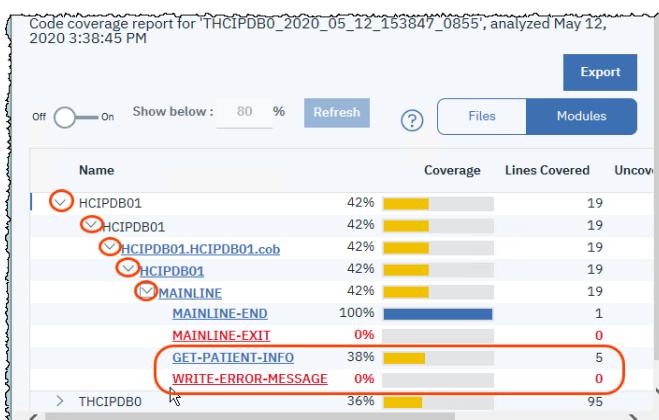
3.5.6 This code coverage report shows the coverage of the COBOL programs generated for zUnit to perform the test as well our HCIPDB01 COBOL program..

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

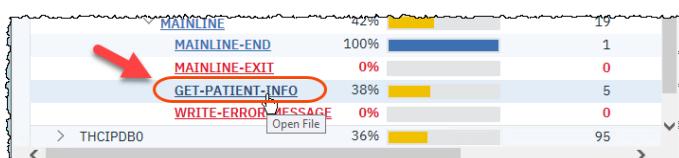


3.5.7 ▶ Expand the nodes as below to see the COBOL paragraphs.

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).

```

JOB00096.jcl   THCPDB0_20...   THCPDB0.bzures   CBL HCIPDB01.HC...   >
-----+---+---+---+---+---+---+---+---+---+---+---+---+
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          Lines 267-274 not covered. 00
268          MOVE '01' TO CA-RETURN-CODE
269          When -913
270          MOVE '01' TO CA-RETURN-CODE
271          When Other
272          MOVE '90' TO CA-RETURN-CODE
273          PERFORM WRITE-ERROR-MESSAGE
274          EXEC CICS RETURN END-EXEC
275          End-Evaluate
276

```

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 to 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/Hcipcc.jcl](#)

► Double click on [jcl/Hcipcc.jcl](#) and verify this JCL. You could just submit this JCL to run the test case and the Code Coverage.

```

z/OS Projects   THCPDB0_20...   THCPDB0.bzures   CBL HCIPDB01.HC...   jcl HCIPCC.jcl   >
-----+---+---+---+---+---+---+---+---+---+---+---+---+
1 //< Hcipcc JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
3 //> Action: Code Coverage...
4 //> Source: IBMUSER.GIT.ZMOBILE.LOAD(THCPDB0)
5 //> RUNNER EXEC PROC=BZUPPLAY,
6 // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCPDB0),
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,THCPDB0,cclevel=LINE,testid=THCPDB0"
14 /*
15 //BZUPLAY DD DISP=SHR,
16 // DSN=IBMUSER.ZUNIT.PB1.HCIPDB01
17 //BZURPT DD DISP=SHR,
18 // DSN=IBMUSER.ZUNIT.BZURES(THCPDB0)

```

Section 4. Introduce a bug in the program and rerun the unit test.

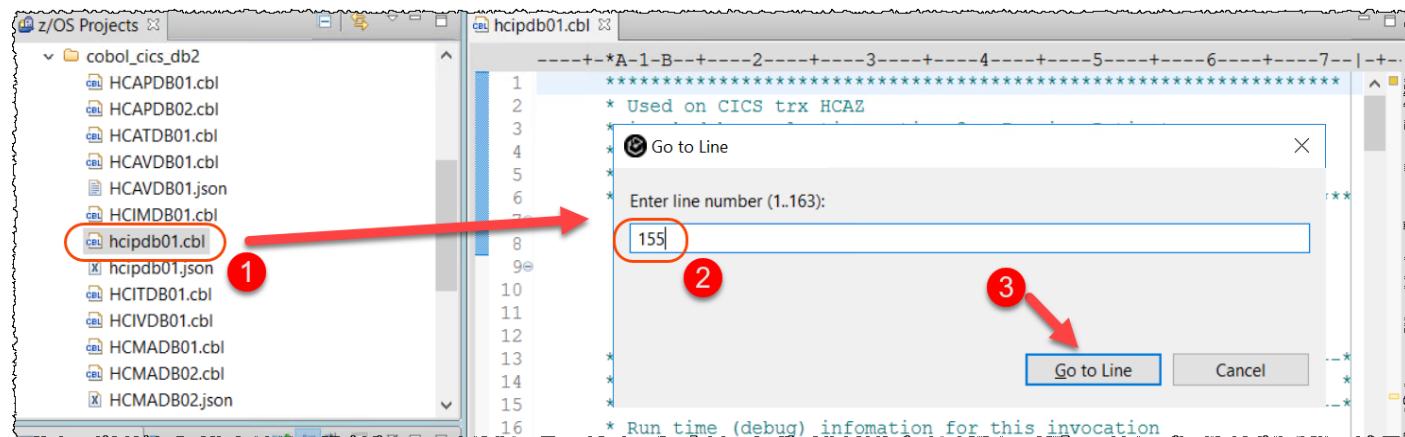
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

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.

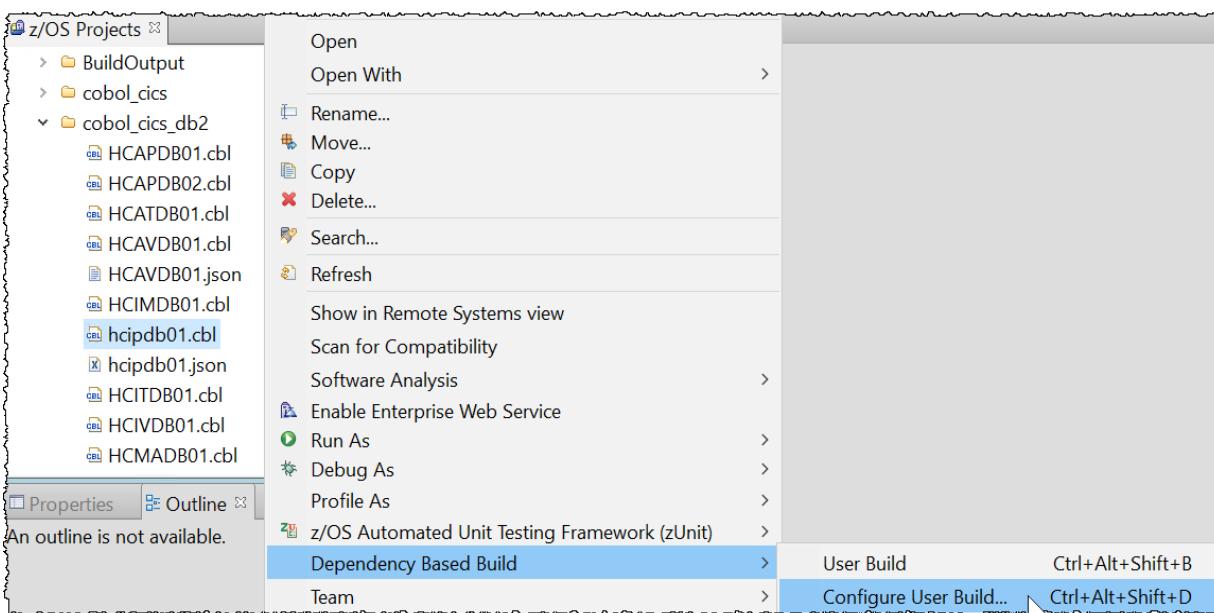
```

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.

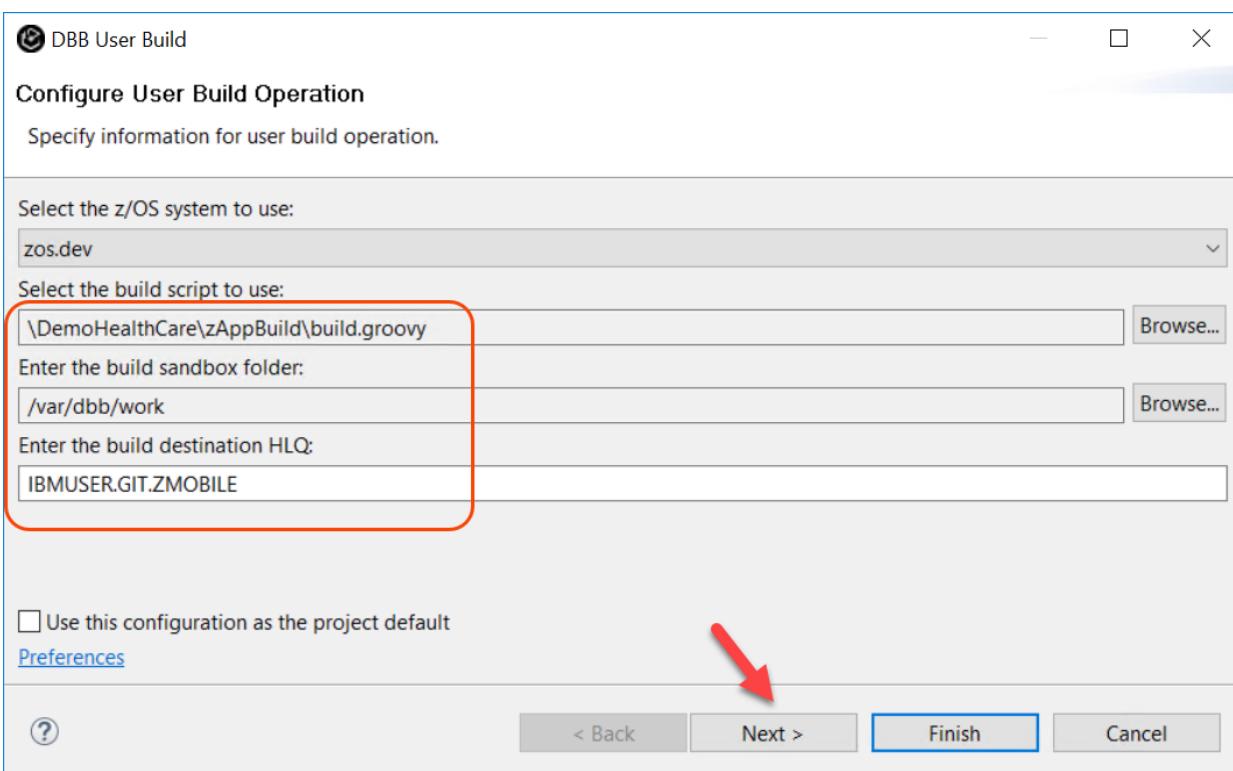
```

4.2 Rebuilding the changed program without deploying to CICS using DBB

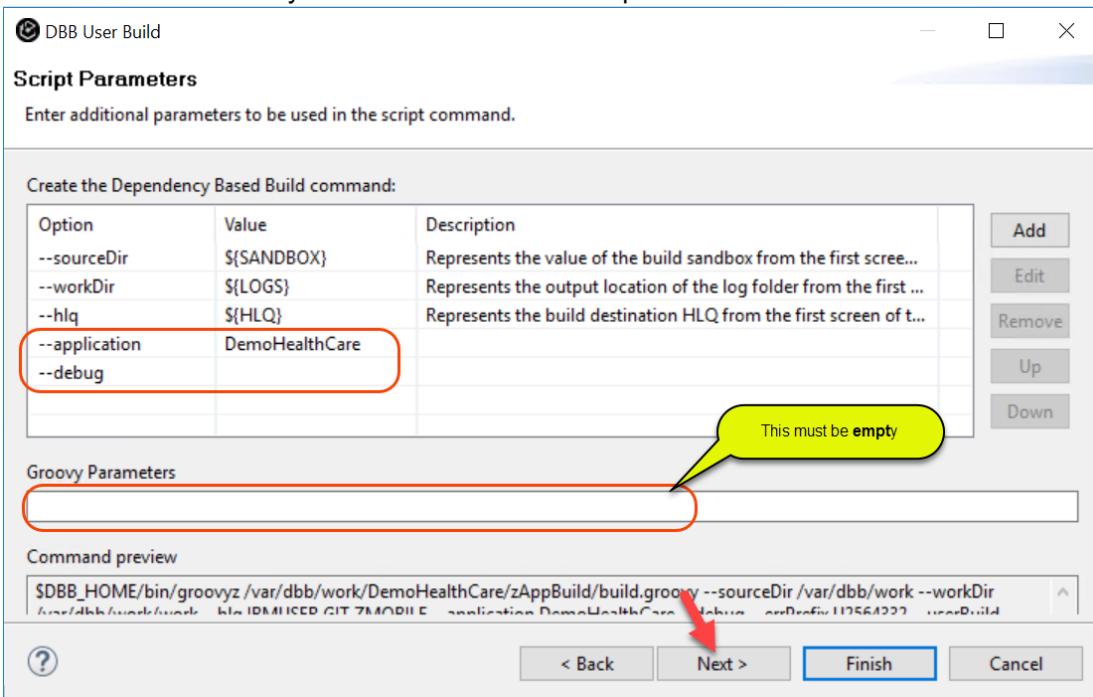
- 4.2.1 On the z/OS Projects view, right click on **hcidb01.cbl** and select **Dependency Based Build > Configure User Build...**



- 4.2.2 If the values are not populated, use the Browse button and specify the values below and click **Next 3 times**



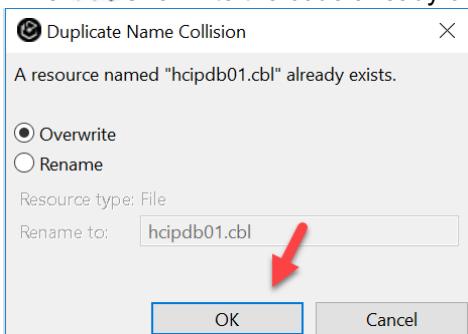
4.2.3 ► Be sure that you have the values below specified and click **Next**



4.2.4 ► Expand **DemoHealthCare**, Un-select **zAppBuild** and click **Finish**.

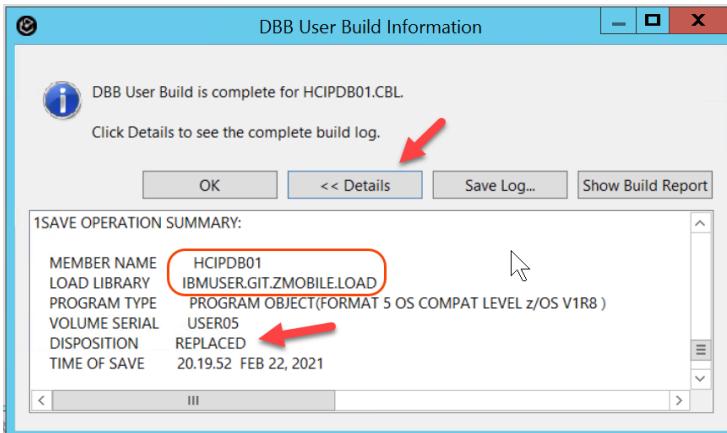
You have already moved all groovy scripts to z/OS before when compiled the Test case program.

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

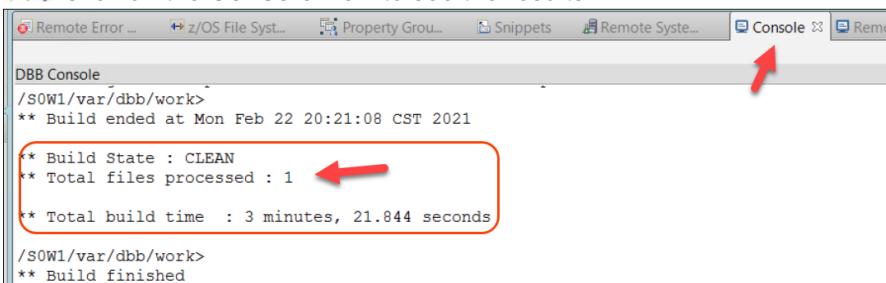


4.2.6 This operation may take up to 2 minutes When finished the dialog below will be displayed.

▶ Click on **Details** and **OK** after you verified that the load module was replaced.



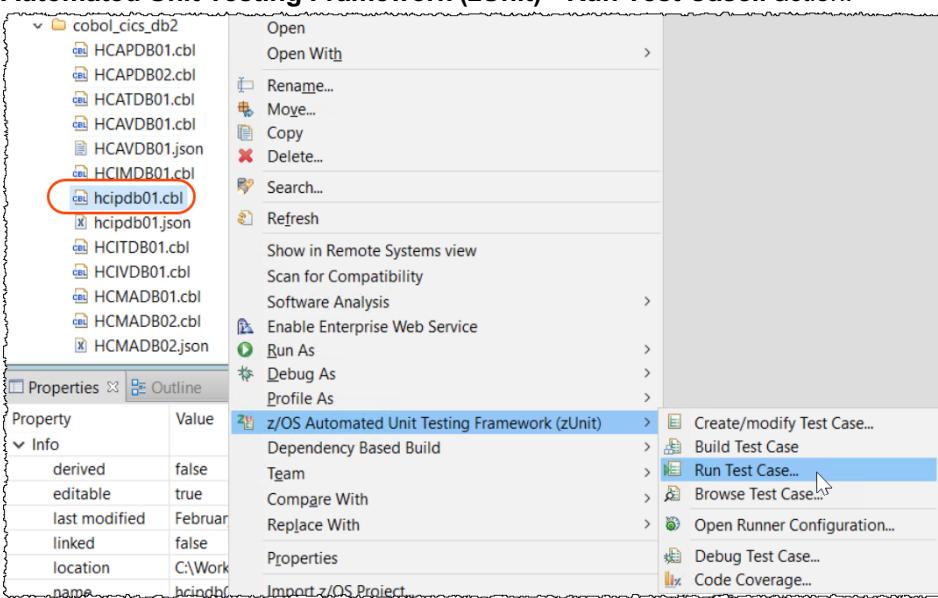
▶ Click on the **Console** view to see the results:



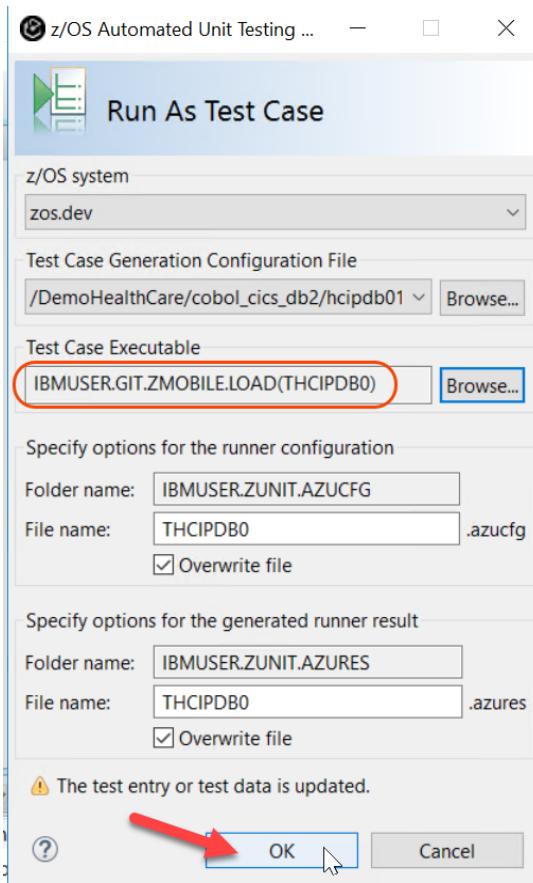
4.3 Running the test case again

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

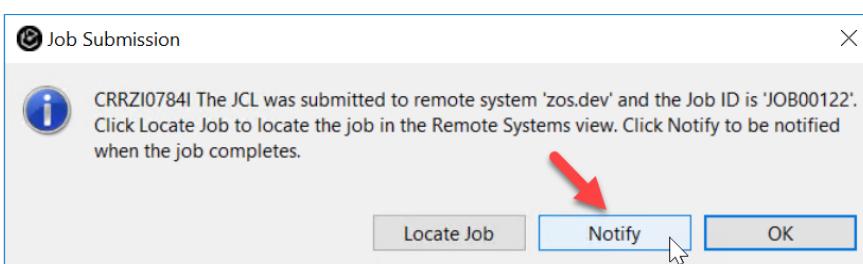
4.3.1 ▶ On the z/OS Projects view, select **hcipdb01.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..** action.



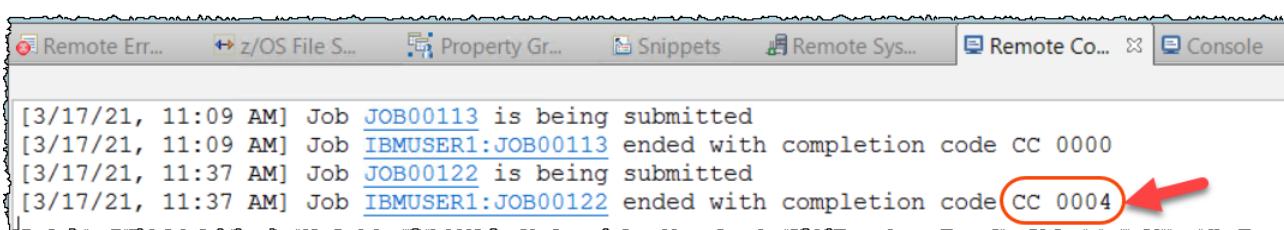
4.3.2 ► Be sure that the test case load module **IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)** is selected and click **OK** to submit the execution via JCL.



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.

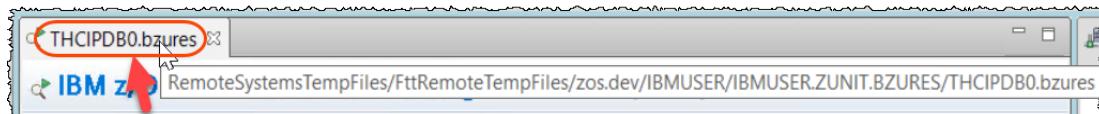


4.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

► Click **Test case THCPDB0**. 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:	THCPDB0
Test case name:	THCPDB0
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 **THCPDB0.bzures** to maximize the windows and better check the results



4.3.7 ► Expand **Test case THCPDB0, 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.

Message:

```
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 EXECUTIO\x RC=04
```

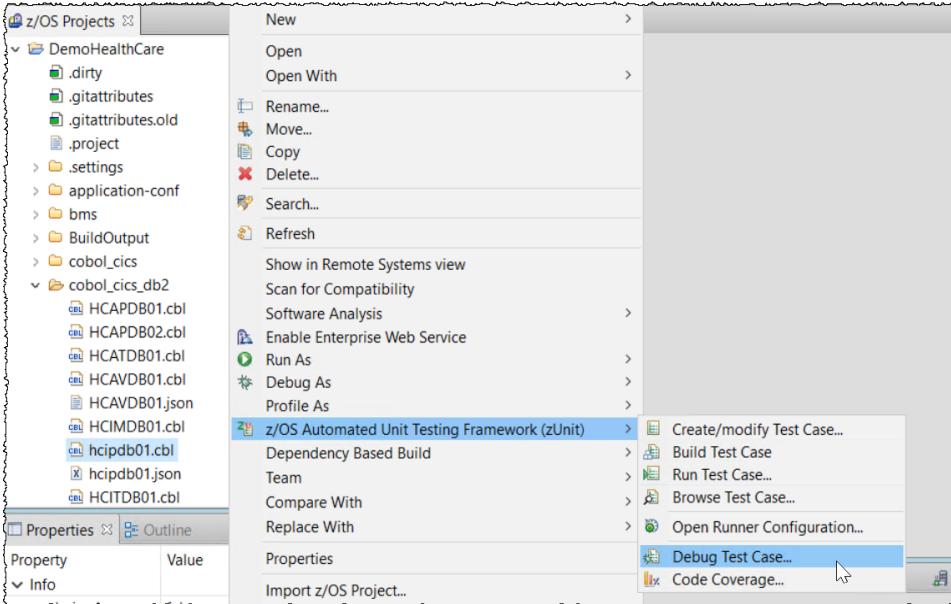
Component code: BZU
Message number: 3000
Message severity: 4

4.4 Running zUnit test case with debugging

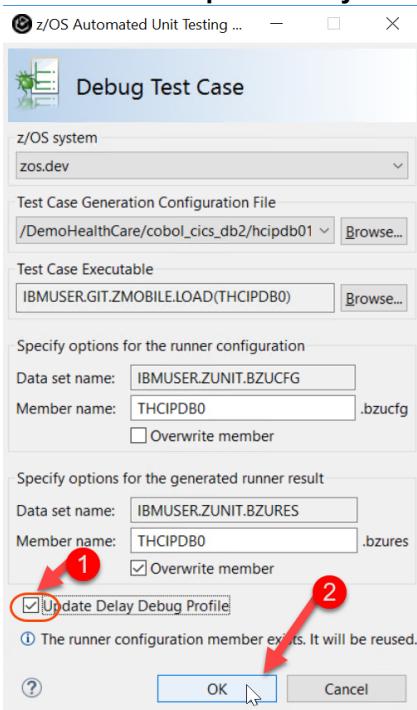
The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example. **Notice that you are debugging a batch job without need to have CICS and DB2 active, which is very handy.**

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

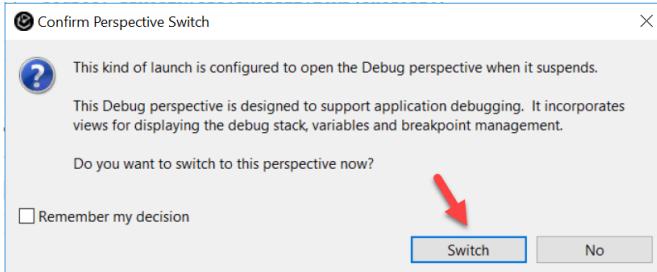
4.4.2 ► Right click on **hcipdb01.cbl** and select **zUnit → Debug Test Case ...**



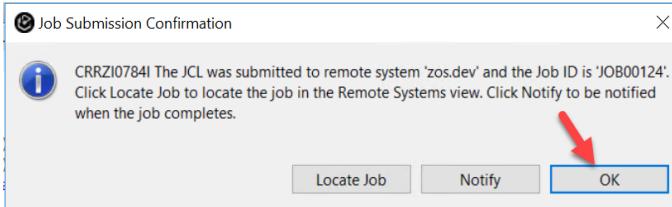
4.4.3 ► Click **Update Delay Debug Profile** and click **OK**



4.4.4 ► Click **Switch** to open the debug perspective.



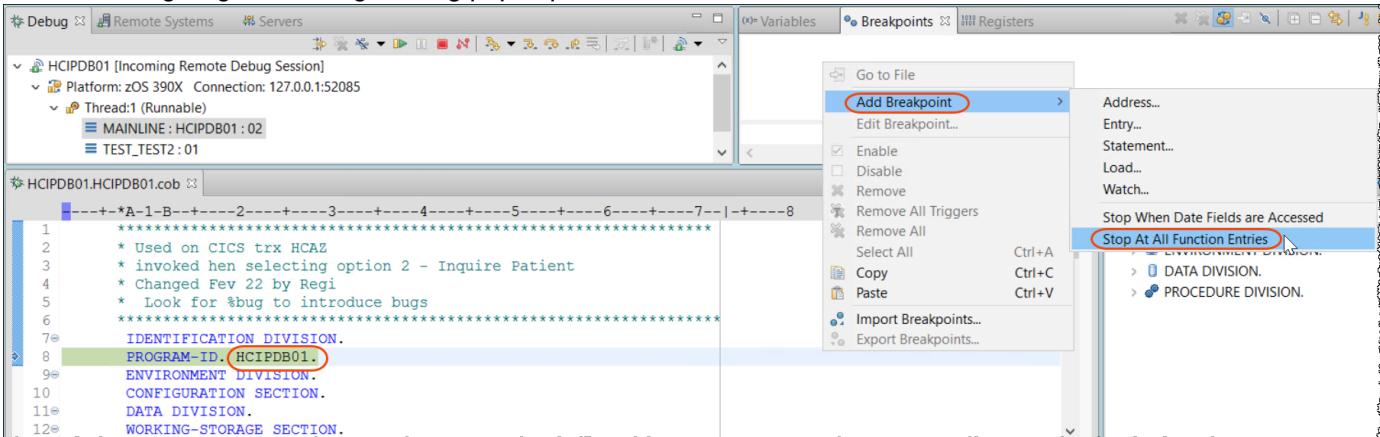
4.4.5 ► Click **OK** to dismiss this dialog



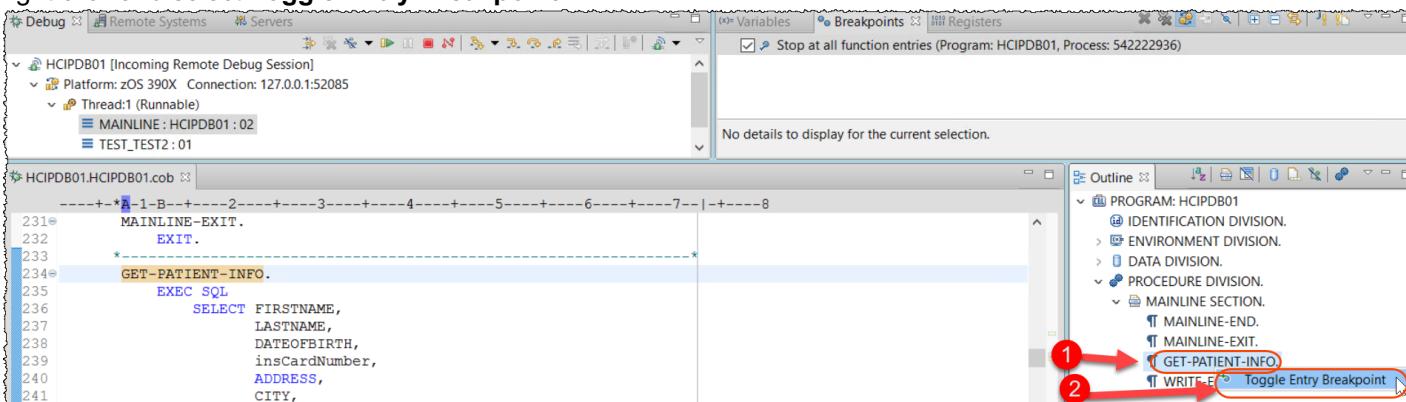
4.4.6 Notice that the first program being debugged is the program that you introduced the bug. (**HCIPDB01**)
The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

► 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*.

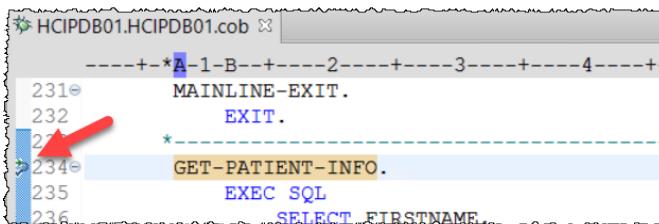
► If a **Debug Engine Message** dialog pops up click **OK**



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.



```

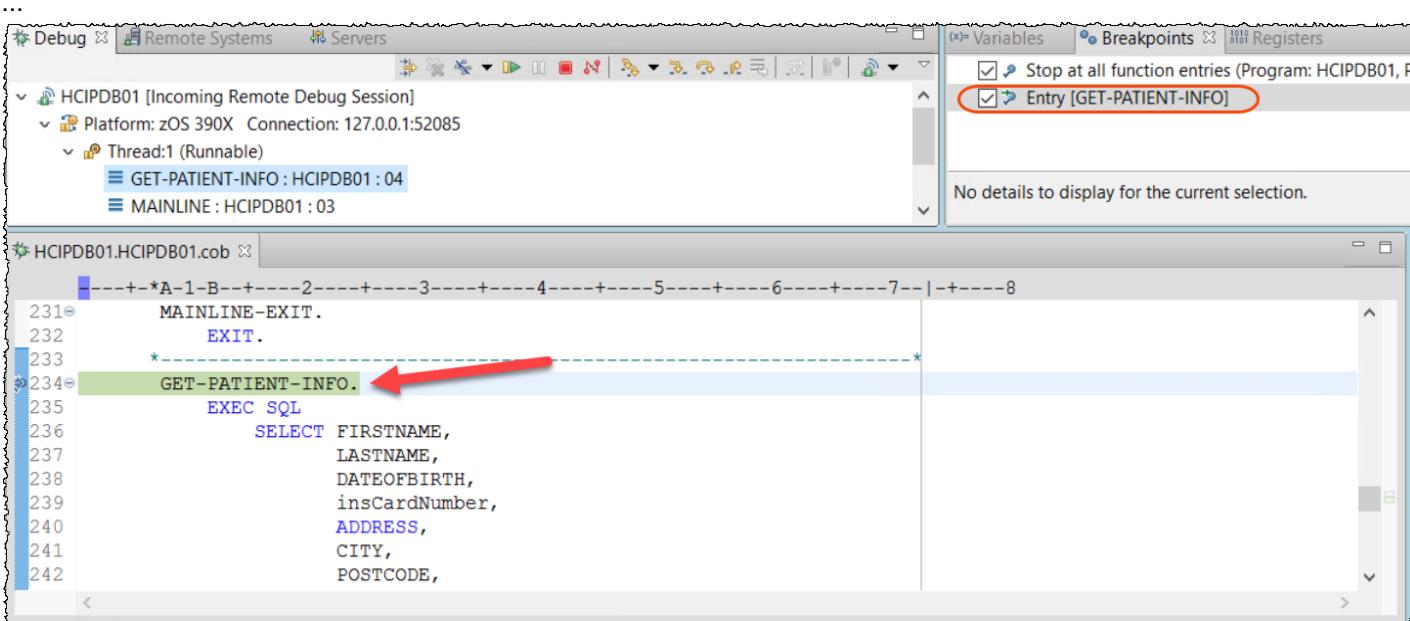
HCPDB01.HCPDB01.cob
-----+---+---+---+---+---+---+---+
231*   MAINLINE-EXIT.
232      EXIT.
233*
234*   GET-PATIENT-INFO.
235      EXEC SQL
236      SELECT FIRSTNAME

```

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'. The left pane displays a 'HCIPDB01 [Incoming Remote Debug Session]' tree view with 'Platform: zOS 390X' and 'Thread:1 (Runnable)' expanded, showing 'GET-PATIENT-INFO : HCIPDB01 : 04' and 'MAINLINE : HCIPDB01 : 03'. The right pane has tabs for 'Variables', 'Breakpoints', and 'Registers'. The 'Breakpoints' tab is active, showing a checkbox for 'Stop at all function entries (Program: HCIPDB01, P)' and another for 'Entry [GET-PATIENT-INFO]' which is checked and highlighted with a red circle. Below these tabs, a message says 'No details to display for the current selection.' The bottom half of the screen shows the COBOL code editor with the following code:

```

HCPDB01.HCPDB01.cob
-----+---+---+---+---+---+---+---+
231*   MAINLINE-EXIT.
232      EXIT.
233*
234*   GET-PATIENT-INFO.          ← Red arrow points here
235      EXEC SQL
236      SELECT FIRSTNAME,
237          LASTNAME,
238          DATEOFBIRTH,
239          insCardNumber,
240          ADDRESS,
241          CITY,
242          POSTCODE,

```

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).

```

-----+*A-1-B-----2-----+---3---+---4---+---5---+---6---+---7---|-----8
      FROM PATIENT
      WHERE PATIENTID = :DB2-PATIENT-ID
      END-EXEC.
259  Evaluate SQLCODE
260  When 0
261    MOVE '00' TO CA-RET
262  When 100
263    MOVE '01' TO CA-RET
264  When -913
265    MOVE '01' TO CA-RET
266  When Other
267    MOVE '90' TO CA-RET

```

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"**.

```

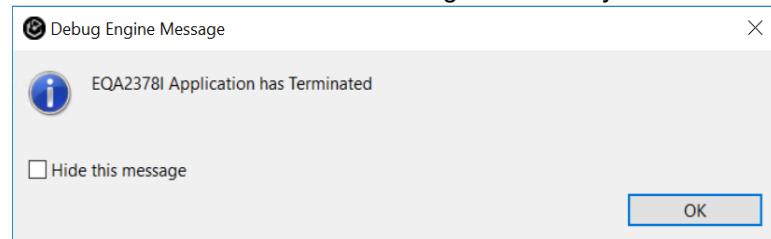
-----+*A-1-B-----2-----+---3---+---4---+---5---+---6---+---7---|-----8
      END-Evaluate.
271  * %bug2 -- the line below will introduce a BUG
272  *
273  IF DB2-PATIENT-ID = 1
274    MOVE "BAD NAME" to CA-FIRST-NAME
275  END-IF
276  *      MOVE "02" to CA-NEWFI
277  *
278  EXIT.
279  *
280  *COPY HCERRSPD.
281  *=====

```

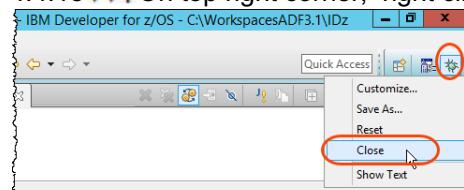
4.4.13 ► Click on icon or press **F8** to resume.. Or if you prefer keep clicking on *Step Into*.



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=BZUPPLAY,
// 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)
//
```

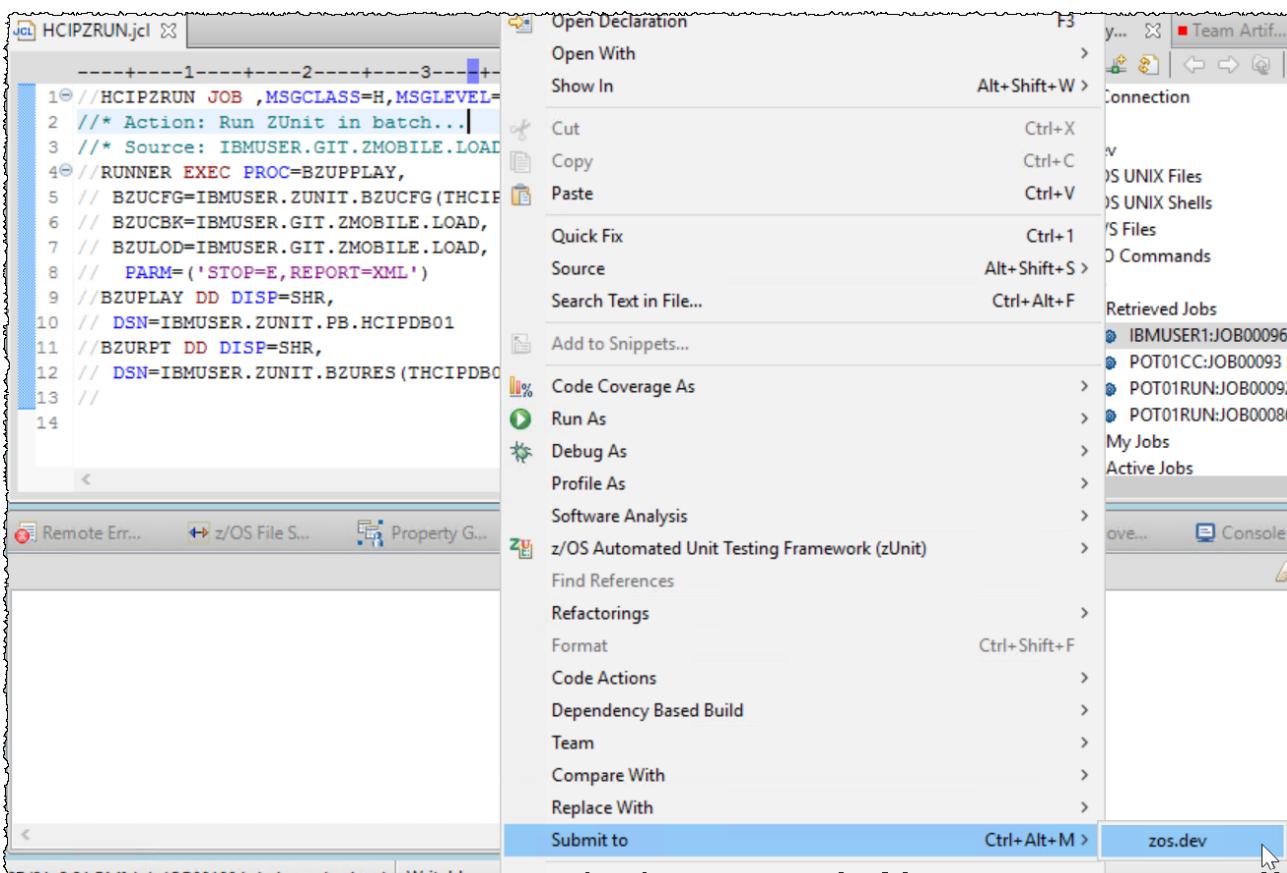
5.1.3 ► Be sure that on the line 10 the dataset name is **IBMUSER.ZUNIT.PB.HCIPDB01** (instead of **PB1**). You may need to change this and then click **Ctrl+S**

```

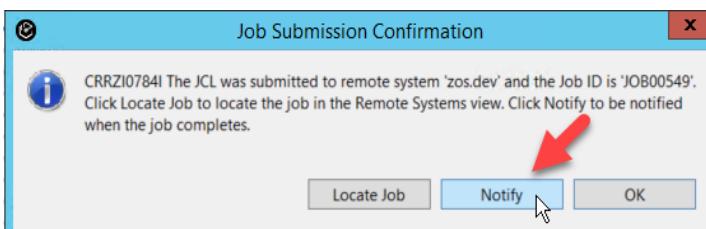
//BZUPLAY DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.PB.HCIPDB01
//BZURPT DD DISP=SHR,
// DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)

```

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

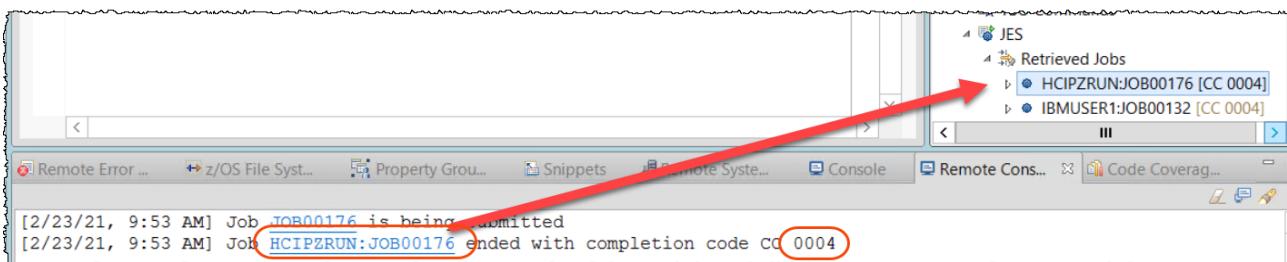


5.1.5 ► Click **Notify** on the Job Submission Confirmation dialog.



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.

The screenshot shows the Rational Developer for z/OS interface. On the left is a terminal window titled "HCIPZRUN.jcl" displaying a log of a CICS transaction. A red box highlights a section of the log output:

```

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.
9 ****
10 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
15 ****
16 TEST_INQ01 SUCCESSFUL.
17

```

A red arrow points from this highlighted section to the "Team Artifacts" panel on the right, which lists several JCL jobs, one of which is "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

The screenshot shows the Rational Developer for z/OS interface. On the left is a terminal window titled "RunZUnitJCLgroovy" displaying a Groovy script. A red box highlights the "RunZUnitJCLgroovy" file in the "zos.dev/z/OS UNIX Files/groovy_samples" directory structure on the right. A red arrow points from the terminal window to this highlighted file.

```

1 import com.ibm.dbb.build.CopyToHFS
2 import com.ibm.dbb.build.DBBConstants
3 import com.ibm.dbb.build.JCLExec
4 /**
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"

```

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

Congratulations! You have completed the Lab 3B.

LAB 3C – (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* FORM 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 prg 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*
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*
214*
215*
216*
217*
218*
219*
220*
221*
222*
223*
224*
225*
226*
227*
228*
229*
230*
231*
232*
233*
234*
235*
236*
237*
238*
239*
240*
241*
242*
243*
244*
245*
246*
247*
248*
249*
250*
251*
252*
253*
254*
255*
256*
257*
258*
259*
259*
260*
261*
262*
263*
264*
265*
266*
267*
268*
269*
270*
271*
272*
273*
274*
275*
276*
277*
278*
279*
280*
281*
282*
283*
284*
285*
286*
287*
288*
289*
289*
290*
291*
292*
293*
294*
295*
296*
297*
298*
299*
299*
300*
301*
302*
303*
304*
305*
306*
307*
308*
309*
309*
310*
311*
312*
313*
314*
315*
316*
317*
318*
319*
319*
320*
321*
322*
323*
324*
325*
326*
327*
328*
329*
329*
330*
331*
332*
333*
334*
335*
336*
337*
338*
339*
339*
340*
341*
342*
343*
344*
345*
346*
347*
348*
349*
349*
350*
351*
352*
353*
354*
355*
356*
357*
358*
359*
359*
360*
361*
362*
363*
364*
365*
366*
367*
368*
369*
369*
370*
371*
372*
373*
374*
375*
376*
377*
378*
379*
379*
380*
381*
382*
383*
384*
385*
386*
387*
388*
389*
389*
390*
391*
392*
393*
394*
395*
396*
397*
398*
399*
399*
400*
401*
402*
403*
404*
405*
406*
407*
408*
409*
409*
410*
411*
412*
413*
414*
415*
416*
417*
418*
419*
419*
420*
421*
422*
423*
424*
425*
426*
427*
428*
429*
429*
430*
431*
432*
433*
434*
435*
436*
437*
438*
439*
439*
440*
441*
442*
443*
444*
445*
446*
447*
448*
449*
449*
450*
451*
452*
453*
454*
455*
456*
457*
458*
459*
459*
460*
461*
462*
463*
464*
465*
466*
467*
468*
469*
469*
470*
471*
472*
473*
474*
475*
476*
477*
478*
479*
479*
480*
481*
482*
483*
484*
485*
486*
487*
488*
489*
489*
490*
491*
492*
493*
494*
495*
496*
497*
498*
499*
499*
500*
501*
502*
503*
504*
505*
506*
507*
508*
509*
509*
510*
511*
512*
513*
514*
515*
516*
517*
518*
519*
519*
520*
521*
522*
523*
524*
525*
526*
527*
528*
529*
529*
530*
531*
532*
533*
534*
535*
536*
537*
538*
539*
539*
540*
541*
542*
543*
544*
545*
546*
547*
548*
549*
549*
550*
551*
552*
553*
554*
555*
556*
557*
558*
559*
559*
560*
561*
562*
563*
564*
565*
566*
567*
568*
569*
569*
570*
571*
572*
573*
574*
575*
576*
577*
578*
579*
579*
580*
581*
582*
583*
584*
585*
586*
587*
588*
589*
589*
590*
591*
592*
593*
594*
595*
596*
597*
598*
599*
599*
600*
601*
602*
603*
604*
605*
606*
607*
608*
609*
609*
610*
611*
612*
613*
614*
615*
616*
617*
618*
619*
619*
620*
621*
622*
623*
624*
625*
626*
627*
628*
629*
629*
630*
631*
632*
633*
634*
635*
636*
637*
638*
639*
639*
640*
641*
642*
643*
644*
645*
646*
647*
648*
649*
649*
650*
651*
652*
653*
654*
655*
656*
657*
658*
659*
659*
660*
661*
662*
663*
664*
665*
666*
667*
668*
669*
669*
670*
671*
672*
673*
674*
675*
676*
677*
678*
679*
679*
680*
681*
682*
683*
684*
685*
686*
687*
688*
689*
689*
690*
691*
692*
693*
694*
695*
696*
697*
698*
699*
699*
700*
701*
702*
703*
704*
705*
706*
707*
708*
709*
709*
710*
711*
712*
713*
714*
715*
716*
717*
718*
719*
719*
720*
721*
722*
723*
724*
725*
726*
727*
728*
729*
729*
730*
731*
732*
733*
734*
735*
736*
737*
738*
739*
739*
740*
741*
742*
743*
744*
745*
746*
747*
748*
749*
749*
750*
751*
752*
753*
754*
755*
756*
757*
758*
759*
759*
760*
761*
762*
763*
764*
765*
766*
767*
768*
769*
769*
770*
771*
772*
773*
774*
775*
776*
777*
778*
779*
779*
780*
781*
782*
783*
784*
785*
786*
787*
788*
788*
789*
789*
790*
791*
792*
793*
794*
795*
796*
797*
798*
799*
799*
800*
801*
802*
803*
804*
805*
806*
807*
808*
809*
809*
810*
811*
812*
813*
814*
815*
816*
817*
818*
819*
819*
820*
821*
822*
823*
824*
825*
826*
827*
828*
829*
829*
830*
831*
832*
833*
834*
835*
836*
837*
838*
839*
839*
840*
841*
842*
843*
844*
845*
846*
847*
848*
849*
849*
850*
851*
852*
853*
854*
855*
856*
857*
858*
859*
859*
860*
861*
862*
863*
864*
865*
866*
867*
868*
869*
869*
870*
871*
872*
873*
874*
875*
876*
877*
878*
878*
879*
880*
881*
882*
883*
884*
885*
886*
887*
888*
888*
889*
889*
890*
891*
892*
893*
894*
895*
896*
897*
898*
898*
899*
899*
900*
901*
902*
903*
904*
905*
906*
907*
908*
909*
909*
910*
911*
912*
913*
914*
915*
916*
917*
918*
919*
919*
920*
921*
922*
923*
924*
925*
926*
927*
928*
929*
929*
930*
931*
932*
933*
934*
935*
936*
937*
938*
939*
939*
940*
941*
942*
943*
944*
945*
946*
947*
948*
949*
949*
950*
951*
952*
953*
954*
955*
956*
957*
958*
959*
959*
960*
961*
962*
963*
964*
965*
966*
967*
968*
969*
969*
970*
971*
972*
973*
974*
975*
976*
977*
978*
978*
979*
979*
980*
981*
982*
983*
984*
985*
986*
987*
988*
988*
989*
989*
990*
991*
992*
993*
994*
995*
996*
997*
998*
999*
999*
1000*
1001*
1002*
1003*
1004*
1005*
1006*
1007*
1008*
1009*
1009*
1010*
1011*
1012*
1013*
1014*
1015*
1016*
1017*
1018*
1019*
1019*
1020*
1021*
1022*
1023*
1024*
1025*
1026*
1027*
1028*
1029*
1029*
1030*
1031*
1032*
1033*
1034*
1035*
1036*
1037*
1038*
1039*
1039*
1040*
1041*
1042*
1043*
1044*
1045*
1046*
1047*
1048*
1049*
1049*
1050*
1051*
1052*
1053*
1054*
1055*
1056*
1057*
1058*
1059*
1059*
1060*
1061*
1062*
1063*
1064*
1065*
1066*
1067*
1068*
1069*
1069*
1070*
1071*
1072*
1073*
1074*
1075*
1076*
1077*
1078*
1078*
1079*
1080*
1081*
1082*
1083*
1084*
1085*
1086*
1087*
1088*
1088*
1089*
1089*
1090*
1091*
1092*
1093*
1094*
1095*
1095*
1096*
1097*
1098*
1099*
1099*
1100*
1101*
1102*
1103*
1104*
1105*
1106*
1107*
1108*
1109*
1109*
1110*
1111*
1112*
1113*
1114*
1115*
1116*
1117*
1118*
1119*
1119*
1120*
1121*
1122*
1123*
1124*
1125*
1126*
1127*
1128*
1129*
1129*
1130*
1131*
1132*
1133*
1134*
1135*
1136*
1137*
1138*
1139*
1139*
1140*
1141*
1142*
1143*
1144*
1145*
1146*
1147*
1148*
1149*
1149*
1150*
1151*
1152*
1153*
1154*
1155*
1156*
1157*
1158*
1159*
1159*
1160*
1161*
1162*
1163*
1164*
1165*
1166*
1167*
1168*
1169*
1169*
1170*
1171*
1172*
1173*
1174*
1175*
1176*
1177*
1178*
1178*
1179*
1180*
1181*
1182*
1183*
1184*
1185*
1186*
1187*
1188*
1188*
1189*
1189*
1190*
1191*
1192*
1193*
1194*
1195*
1195*
1196*
1197*
1198*
1199*
1199*
1200*
1201*
1202*
1203*
1204*
1205*
1206*
1207*
1208*
1209*
1209*
1210*
1211*
1212*
1213*
1214*
1215*
1216*
1217*
1218*
1219*
1219*
1220*
1221*
1222*
1223*
1224*
1225*
1226*
1227*
1228*
1229*
1229*
1230*
1231*
1232*
1233*
1234*
1235*
1236*
1237*
1238*
1239*
1239*
1240*
1241*
1242*
1243*
1244*
1245*
1246*
1247*
1248*
1249*
1249*
1250*
1251*
1252*
1253*
1254*
1255*
1256*
1257*
1258*
1259*
1259*
1260*
1261*
1262*
1263*
1264*
1265*
1266*
1267*
1268*
1269*
1269*
1270*
1271*
1272*
1273*
1274*
1275*
1276*
1277*
1278*
1278*
1279*
1279*
1280*
1281*
1282*
1283*
1284*
1285*
1286*
1287*
1288*
1288*
1289*
1289*
1290*
1291*
1292*
1293*
1294*
1295*
1295*
1296*
1297*
1298*
1299*
1299*
1300*
1301*
1302*
1303*
1304*
1305*
1306*
1307*
1308*
1309*
1309*
1310*
1311*
1312*
1313*
1314*
1315*
1316*
1317*
1318*
1319*
1319*
1320*
1321*
1322*
1323*
1324*
1325*
1326*
1327*
1328*
1329*
1329*
1330*
1331*
1332*
1333*
1334*
1335*
1336*
1337*
1338*
1339*
1339*
1340*
1341*
1342*
1343*
1344*
1345*
1346*
1347*
1348*
1349*
1349*
1350*
1351*
1352*
1353*
1354*
1355*
1356*
1357*
1358*
1359*
1359*
1360*
1361*
1362*
1363*
1364*
1365*
1366*
1367*
1368*
1369*
1369*
1370*
1371*
1372*
1373*
1374*
1375*
1376*
1377*
1378*
1378*
1379*
1379*
1380*
1381*
1382*
1383*
1384*
1385*
1386*
1387*
1388*
1388*
1389*
1389*
1390*
1391*
1392*
1393*
1394*
1395*
1396*
1397*
1398*
1398*
1399*
1399*
1400*
1401*
1402*
1403*
1404*
1405*
1406*
1407*
1408*
1409*
1409*
1410*
1411*
1412*
1413*
1414*
1415*
1416*
1417*
1418*
1419*
1419*
1420*
1421*
1422*
1423*
1424*
1425*
1426*
1427*
1428*
1429*
1429*
1430*
1431*
1432*
1433*
1434*
1435*
1436*
1437*
1438*
1439*
1439*
1440*
1441*
1442*
1443*
1444*
1445*
1446*
1447*
1448*
1449*
1449*
1450*
1451*
1452*
1453*
1454*
1455*
1456*
1457*
1458*
1459*
1459*
1460*
1461*
1462*
1463*
1464*
1465*
1466*
1467*
1468*
1469*
1469*
1470*
1471*
1472*
1473*
1474*
1475*
1476*
1477*
1478*
1478*
1479*
1479*
1480*
1481*
1482*
1483*
1484*
1485*
1486*
1487*
1488*
1488*
1489*
1489*
1490*
1491*
1492*
1493*
1494*
1495*
1495*
1496*
1497*
1498*
1499*
1499*
1500*
1501*
1502*
1503*
1504*
1505*
1506*
1507*
1508*
1509*
1509*
1510*
1511*
1512*
1513*
1514*
1515*
1516*
1517*
1518*
1519*
1519*
1520*
1521*
1522*
1523*
1524*
1525*
1526*
1527*
1528*
1529*
1529*
1530*
1531*
1532*
1533*
1534*
1535*
1536*
1537*
1538*
1539*
1539*
1540*
1541*
1542*
1543*
1544*
1545*
1546*
1547*
1548*
1549*
1549*
1550*
1551*
1552*
1553*
1554*
1555*
1556*
1557*
1558*
1559*
1559*
1560*
1561*
1562*
1563*
1564*
1565*
1566*
1567*
1568*
1569*
1569*
1570*
1571*
1572*
1573*
1574*
1575*
1576*
1577*
1578*
1578*
1579*
1579*
1580*
1581*
1582*
1583*
1584*
1585*
1586*
1587*
1588*
1588*
1589*
1589*
1590*
1591*
1592*
1593*
1594*
1595*
1595*
1596*
1597*
1598*
1599*
1599*
1600*
1601*
1602*
1603*
1604*
1605*
1606*
1607*
1608*
1609*
1609*
1610*
1611*
1612*
1613*
1614*
1615*
1616*
1617*
1618*
1619*
1619*
1620*
1621*
1622*
1623*
1624*
1625*
1626*
1627*
1628*
1629*
1629*
1630*
1631*
1632*
1633*
1634*
1635*
1636*
1637*
1638*
1639*
1639*
1640*
1641*
1642*
1643*
1644*
1645*
1646*
1647*
1648*
1649*
1649*
1650*
1651*
1652*
1653*
1654*
1655*
1656*
1657*
1658*
1659*
1659*
1660*
1661*
1662*
1663*
1664*
1665*
1666*
1667*
1668*
1669*
1669*
1670*
1671*
1672*
1673*
1674*
1675*
1676*
1677*
1678*
1678*
1679*
1679*
1680*
1681*
1682*
1683*
1684*
1685*
1686*
1687*
1688*
1688*
1689*
1689*
1690*
1691*
1692*
1693*
1694*
1695*
1695*
1696*
1697*
1698*
1699*
1699*
1700*
1701*
1702*
1703*
1704*
1705*
1706*
1707*
1708*
1709*
1709*
1710*
1711*
1712*
1713*
1714*
1715*
1716*
1717*
1718*
1719*
1719*
1720*
1721*
1722*
1723*
1724*
1725*
1726*
1727*
1728*
1729*
1729*
1730*
1731*
1732*
1733*
1734*
1735*
1736*
1737*
1738*
1739*
1739*
1740*
1741*
1742*
1743*
1744*
1745*
1746*
1747*
1748*
1749*
1749*
1750*
1751*
1752*
1753*
1754*
1755*
1756*
1757*
1758*
1759*
1759*
1760*
1761*
1762*
1763*
1764*
1765*
1766*
1767*
1768*
1769*
1769*
1770*
1771*
1772*
1773*
1774*
1775*
1776*
1777*
1778*
1778*
1779*
1779*
1780*
1781*
1782*
1783*
1784*
1785*
1786*
1787*
1788*
1788*
1789*
1789*
1790*
1791*
1792*
1793*
1794*
1795*
1795*
1796*
1797*
1798*
1799*
1799*
1800*
1801*
1802*
1803*
1804*
1805*
1806*
1807*
1808*
1809*
1809*
1810*
1811*
1812*
1813*
1814*
1815*
1816*
1817*
1818*
1819*
1819*
1820*
1821*
1822*
1823*
1824*
1825*
1826*
1827*
1828*
1829*
1829*
1830*
1831*
1832*
1833*
1834*
1835*
1836*
1837*
1838*
1839*
1839*
1840*
1841*
1842*
1843*
1844*
1845*
1846*
1847*
1848*
1849*
1849*
1850*
1851*
1852*
1853*
1854*
1855*
1856*
1857*
1858*
1859*
1859*
1860*
1861*
1862*
1863*
1864*
1865*
1866*
1867*
1868*
1869*
1869*
1870*
1871*
1872*
1873*
1874*
1875*
1876*
1877*
1878*
1878*
1879*
1879*
1880*
1881*
1882*
1883*
1884*
1885*
1886*
1887*
1888*
1888*
1889*
1889*
1890*
1891*
1892*
1893*
1894*
1895*
1895*
1896*
1897*
1898*
1899*
1899*
1900*
1901*
1902*
1903*
1904*
1905*
1906*
1907*
1908*
1909*
1909*
1910*
1911*
1912*
1913*
1914*
1915*
1916*
1917*
1918*
1919*
1919*
1920*
1921*
1922*
1923*
1924*
1925*
1926*
1927*
1928*
1929*
1929*
1930*
1931*
1932*
1933*
1934*
1935*
1936*
1937*
1938*
1939*
1939*
1940*
1941*
1942*
1943*
1944*
1945*
1946*
1947*
1948*
1949*
1949*
1950*
1951*
1952*
1953*
1954*
1955*
1956*
1957*
1958*
1959*
1959*
1960*
1961*
1962*
1963*
1964*
1965*
1966*
1967*
1968*
1969*
1969*
1970*
1971*
1972*
1973*
1974*
1975*
1976*
1977*
1978*
1978*
1979*
1979*
1980*
1981*
1982*
1983*
1984*
1985*
1986*
1987*
1988*
1988*
1989*
1989*
1990*
1991*
1992*
1993*
1994*
1995*
1996*
1997*
1998*
1999*
1999*
2000*
2001*
2002*
2003*
2004*
2005*
2006*
2007*
2008*
2009*
2009*
2010*
2011*
2012*
2013*
2014*
2015*
2016*
2017*
2018*
2019*
2020*
2021*
2022*
2023*
2024*
2025*
2026*
2027*
2028*
2029*
2030*
2031*
2032*
2033*
2034*
2035*
2036*
2037*
2038*
2039*
2039*
2040*
2041*
2042*
2043*
2044*
2045*
2046*
2047*
2048*
2049*
2049*
2050*
2051*
2052*
2053*
2054*
2055*
2056*
2057*
2058*
2059*
2059*
2060*
2061*
2062*
2063*
2064*
2065*
2066*
2067*
2068*
2069*
2069*
2070*
2071*
2072*
2073*
2074*
2075*
2076*
2077*
2078*
2078*
2079*
2079*
2080*
2081*
2082*
2083*
2084*
2085*
2086*
2087*
2088*
2088*
2089*
2089*
2090*
2091*
2092*
2093*
2094*
2095*
2095*
2096*
2097*
2098*
2099*
2099*
2100*
2101*
2102*
2103*
2104*
2105*
2106*
2107*
2108*
2109*
2109*
2110*
2111*
2112*
2113*
2114*
2115*
2116*
2117*
2118*
2119*
2119*
2120*
2121*
2122*
2123*
2124*
2125*
2126*
2127*
2128*
2129*
2129*
2130*
2131*
2132*
2133*
2134*
2135*
2136*
2137*
2138*
2139*
2139*
2140*
2141*
2142*
2143*
2144*
2145*
2146*
2147*
2148*
2149*
2149*
2150*
2151*
2152*
2153*
2154*
2155*
2156*
2157*
2158*
2159*
2159*
2160*
2161*
2162*
2163*
2164*
2165*
2166*
2167*
2168*
2169*
2169*
2
```

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

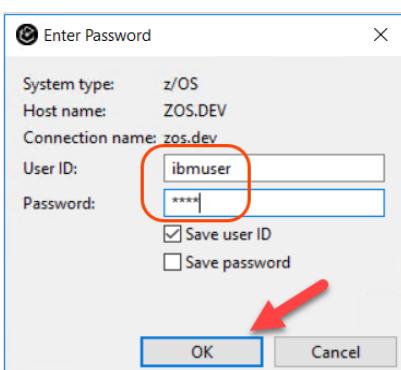
Nothing will happen you are already at this [perspective].

- 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.

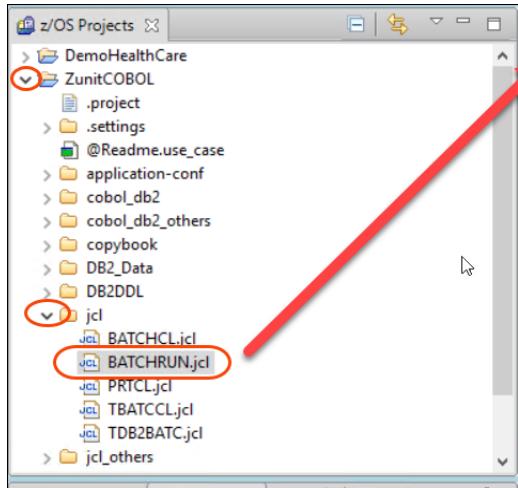
► Using *Remote Systems* view, right click on **zos.dev** and select **Disconnect** and then right click on **zos.dev** again and select **Connect**

- 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
> DemoHealthCare
  > ZunitCOBOL
    > .project
    > .settings
    > @Readme.use_case
    > application-conf
    > cobol_db2
    > cobol_db2_others
    > copybook
    > DB2_Data
    > DB2DDL
    > jcl
      > BATCHCL.jcl
      > BATCHRUN.jcl (highlighted)
      > PRTCL.jcl
      > TBATCL.jcl
      > TDB2BATC.jcl
    > jcl_others
  > jcl_others

```

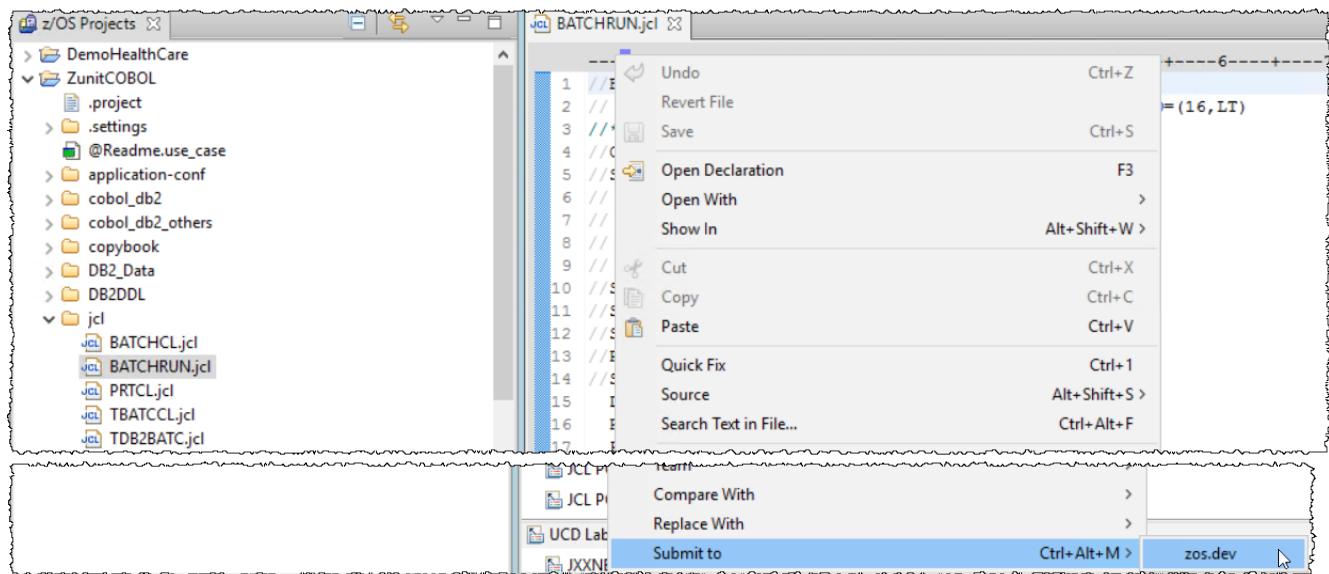
JCL BATCHRUN.jcl

```

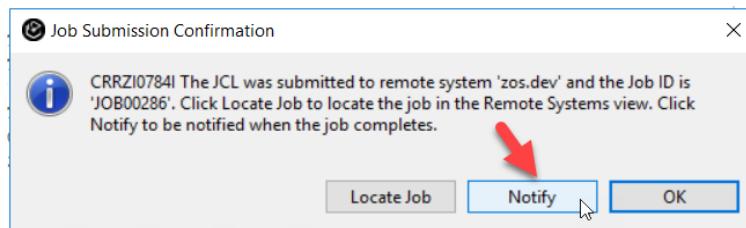
1 //> BATCHRUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 /* Execute DB2BATCH Batch program
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=IBMMUSER.POT.LOAD,DISP=SHR
6 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
7 //          DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
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
19 /*

```

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

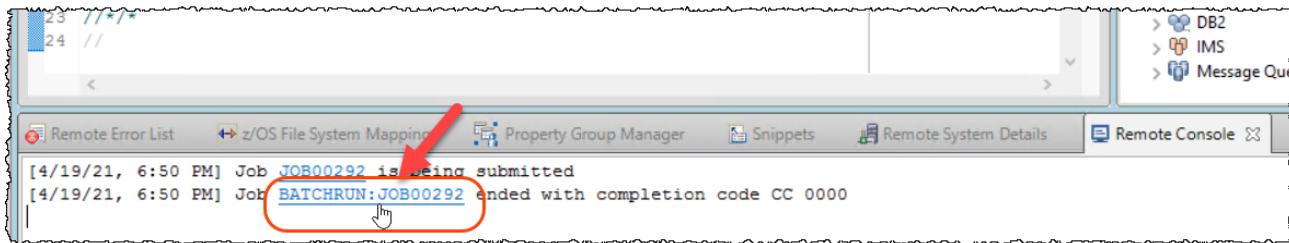


1.1.6 ► Click **Notify** to be notified when the execution is complete.



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.

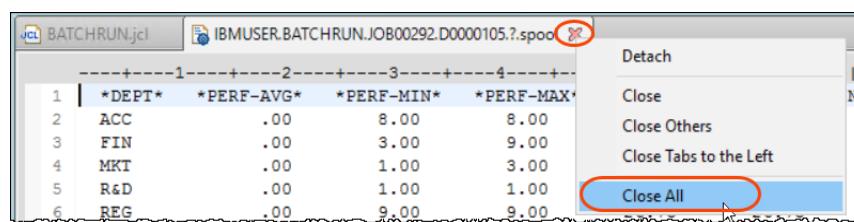
Tip: If there is no jobs under “Retrieved Jobs”, is because you did not click on link as stated at 1.1.7. You can also see this output once you right click on “My Jobs” under *JES* and select **Refresh**.

The screenshot shows the Rational Application Developer interface with the 'Remote Systems' view expanded to show the 'JES' node. Under 'JES', the 'Retrieved Jobs' node is expanded, showing the job 'BATCHRUN:JOB00292 [CC 0000]'. This job has several steps listed, with 'CURSRAV4::RPTPRINT' highlighted by a red circle and a red arrow pointing to it from the left.

On the left, a code editor window titled 'BATCHRUN.jcl' shows the JCL code for the job. A yellow callout bubble points to the first line of the output, which contains the header for a data table and the first row where the DEPT value is 'ACC'.

Notice on first line the **DEPT** value of **ACC**.
When you introduce a bug this value will be different.

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 (**REG10C**) using a static call.

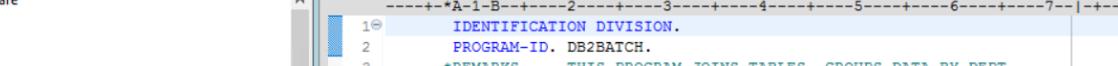
2.1 Understanding the main COBOL program that reads from DB2 table

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..

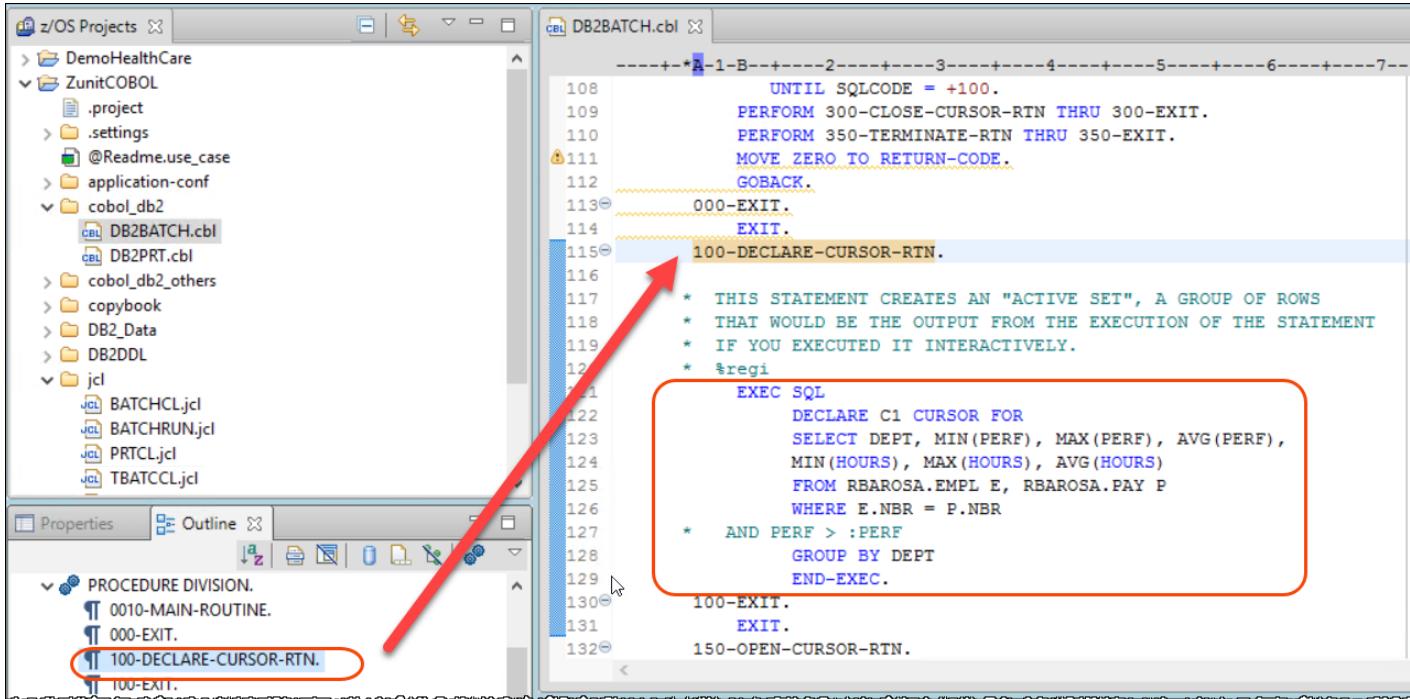


The screenshot shows the z/OS Projects interface with the DB2BATCH.cbl file selected in the left pane. A red arrow points from the file name in the project tree to the file content in the right pane.

```
--A-1-B-+---2---+---3---+---4---+---5---+---6---+---7--|---8
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 ro 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.
```

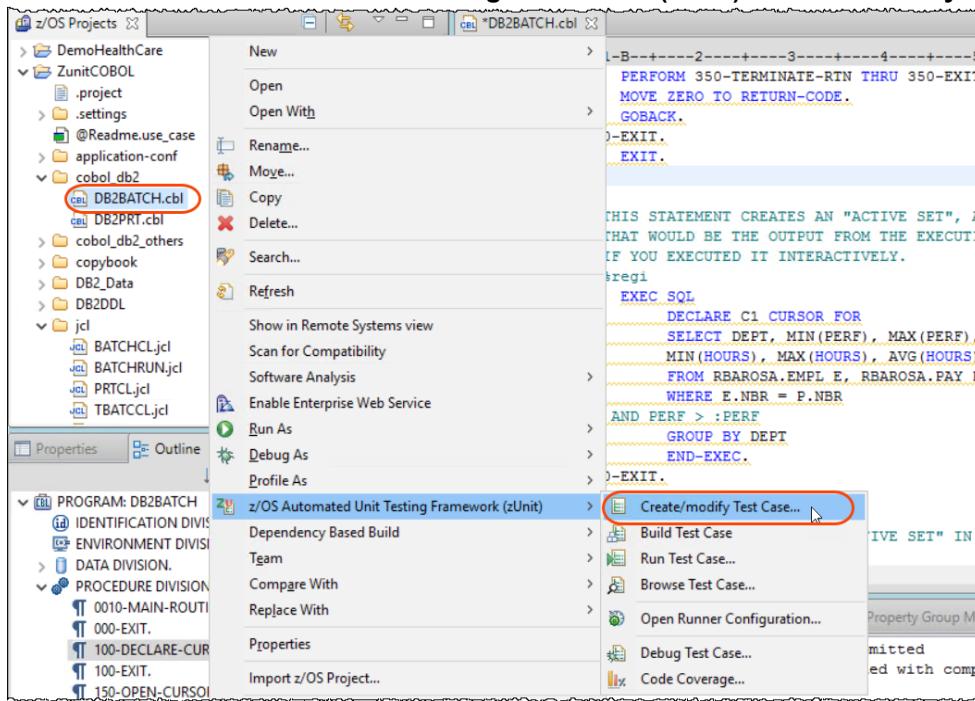
2.1.2 ► Using the **Outline** view on left expand **PROCEDURE DIVISION** and click on **100-DECLARE-CURSOR-RTN**.

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.



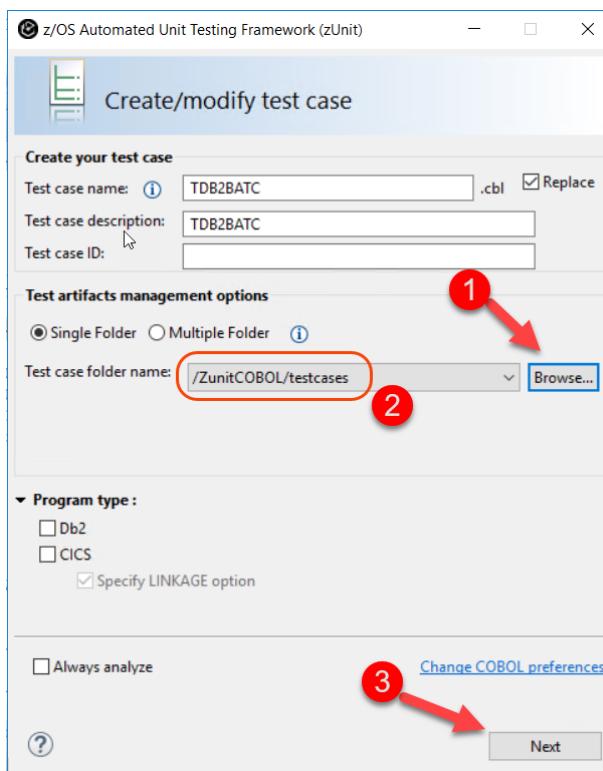
2.2 Recording the batch JCL execution

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..

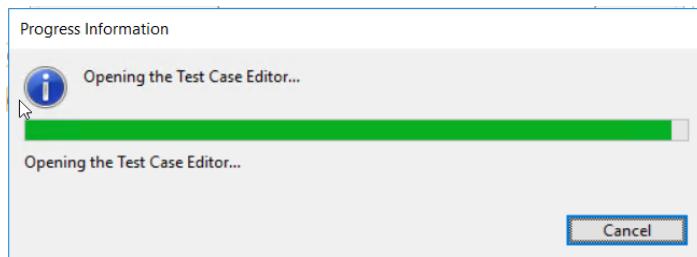


2.2.2 This opens a dialog where you can name your test case.

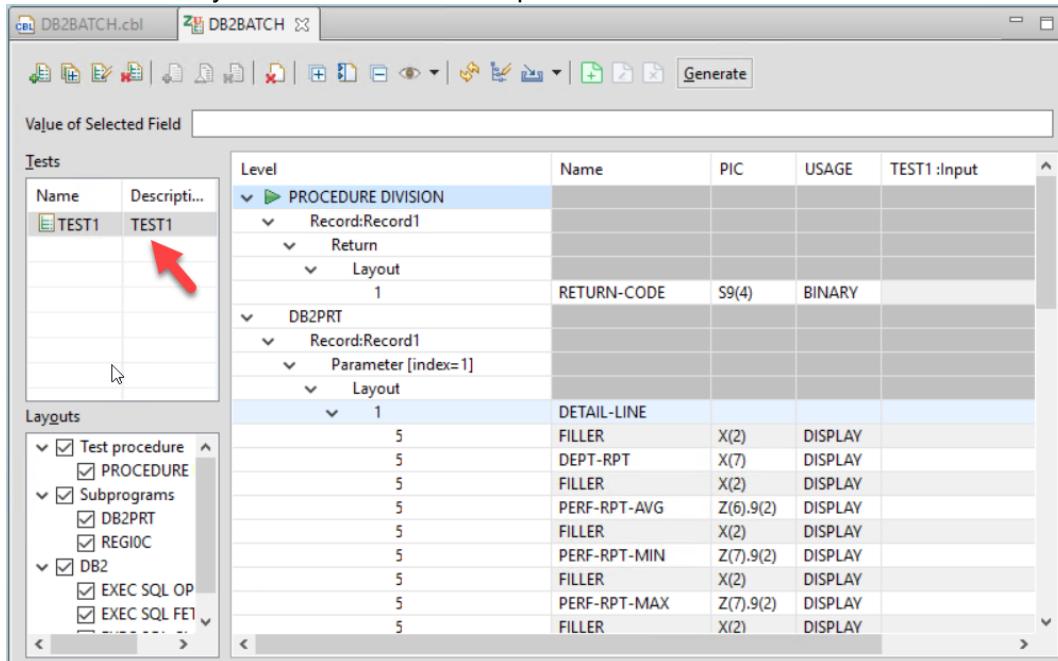
Using the Browse button select /ZunitCOBOL/testcases as folder name and click Next.



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



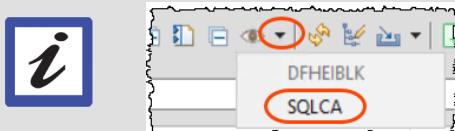
2.2.4 The *Test Case Editor*, as shown below. TEST1 may or may not be on your screen. If TEST1 is there you will delete on next step.



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.

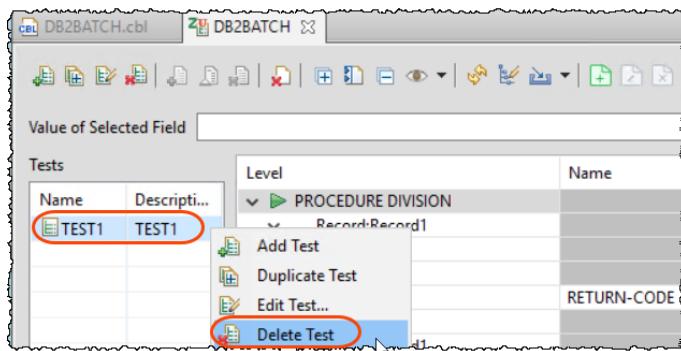


The test entry Input and Expected output columns represent the flow of data into and out of the main program, that is, the program being tested by zUnit. When data is added to these columns in a subroutine, the flow of data is:

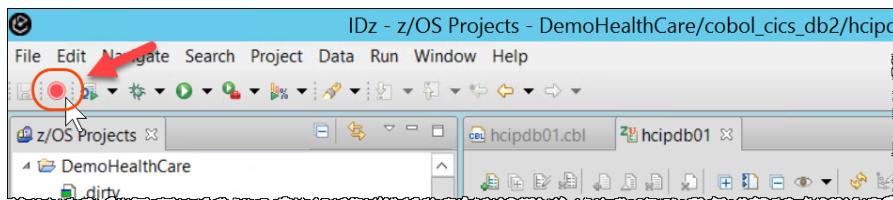
Input: data passed into the main program: that is, what is passed to the main program when the subroutine completes execution.

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.

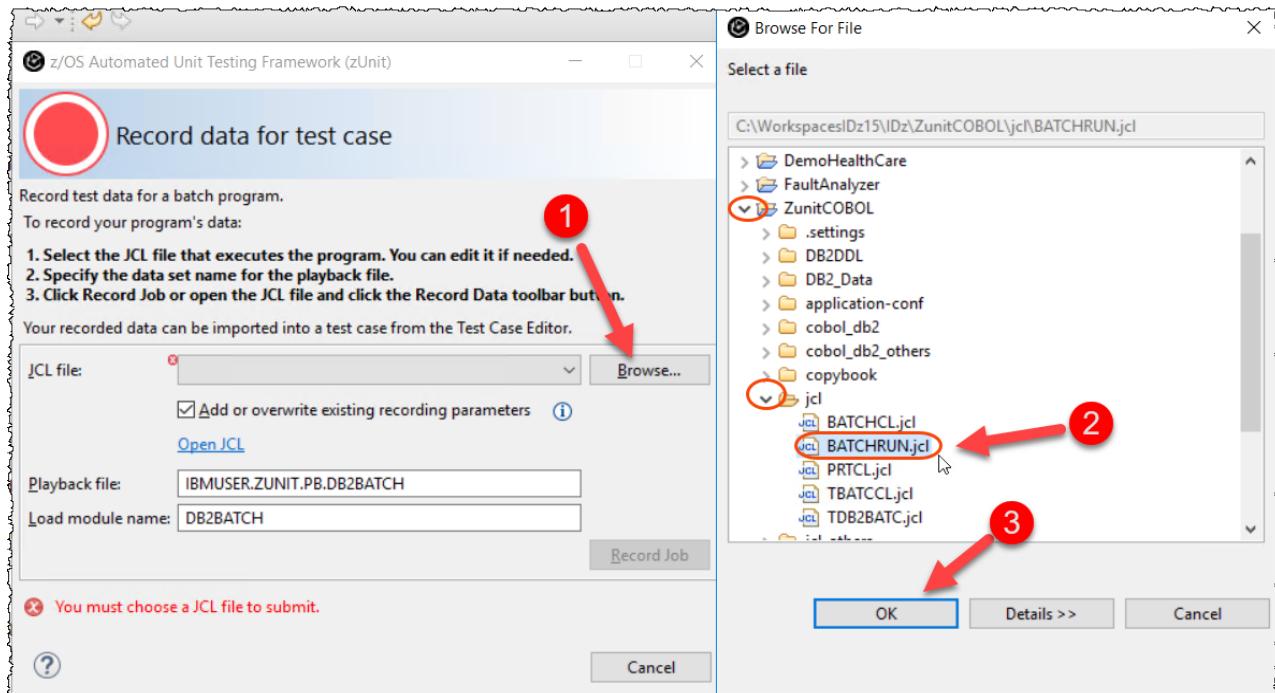
2.2.5 ► Since we will be recording the program execution, delete the **TEST1** or any other entry (if it exists) by right clicking and selecting **Delete Test**



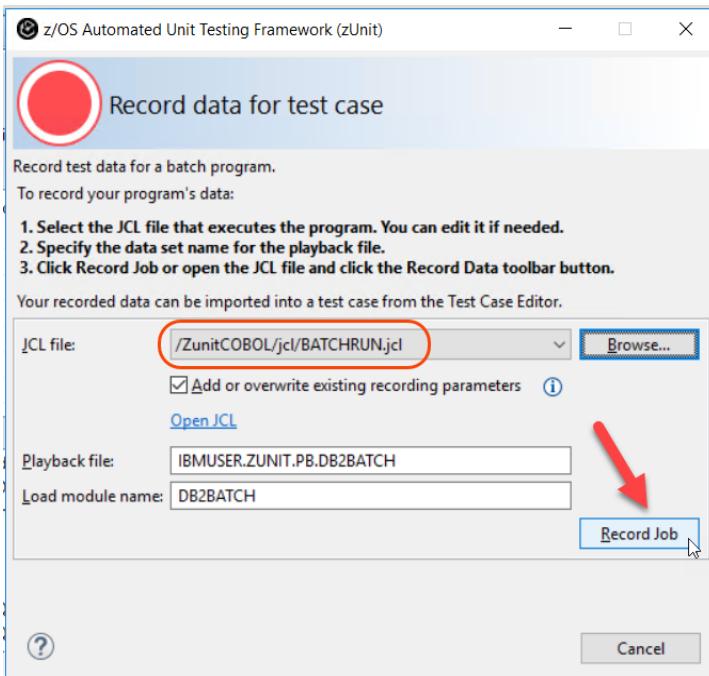
2.2.6 ► To record from a JCL batch execution into the test case, click the **Record** button on the IDz toolbar.



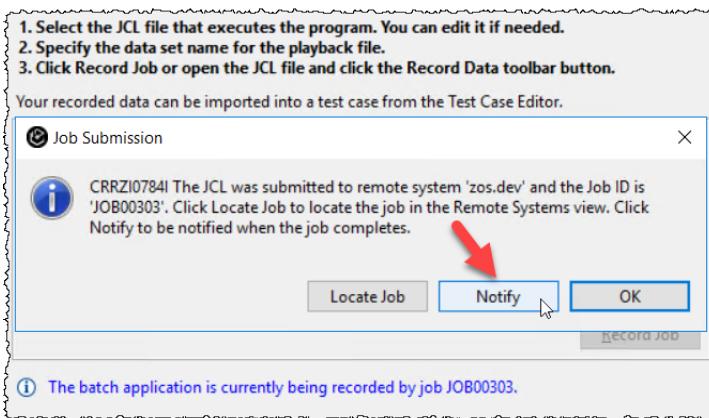
2.2.7 ► In the dialog that comes up, use the **Browse** button to select **BATCHRUN.jcl** under **ZunitCOBOL/jcl** and click **OK**.



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

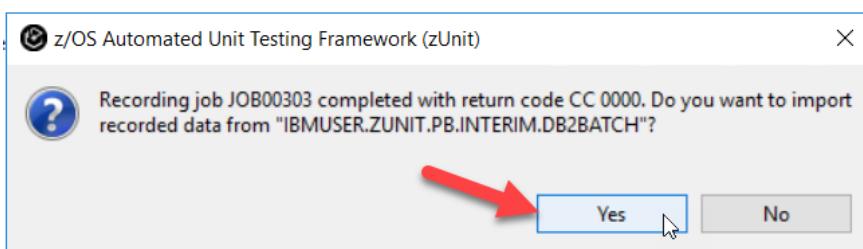


2.2.9 ► Click **Notify** for the dialog below. The JCL will be submitted to run on z/OS.

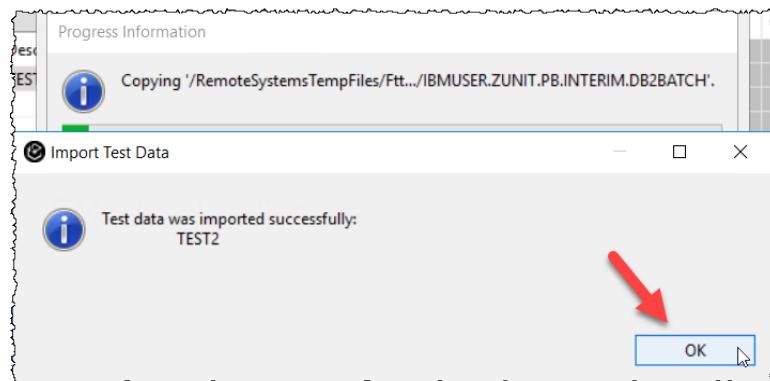


2.2.10 When the job is completed the dialog below is displayed.

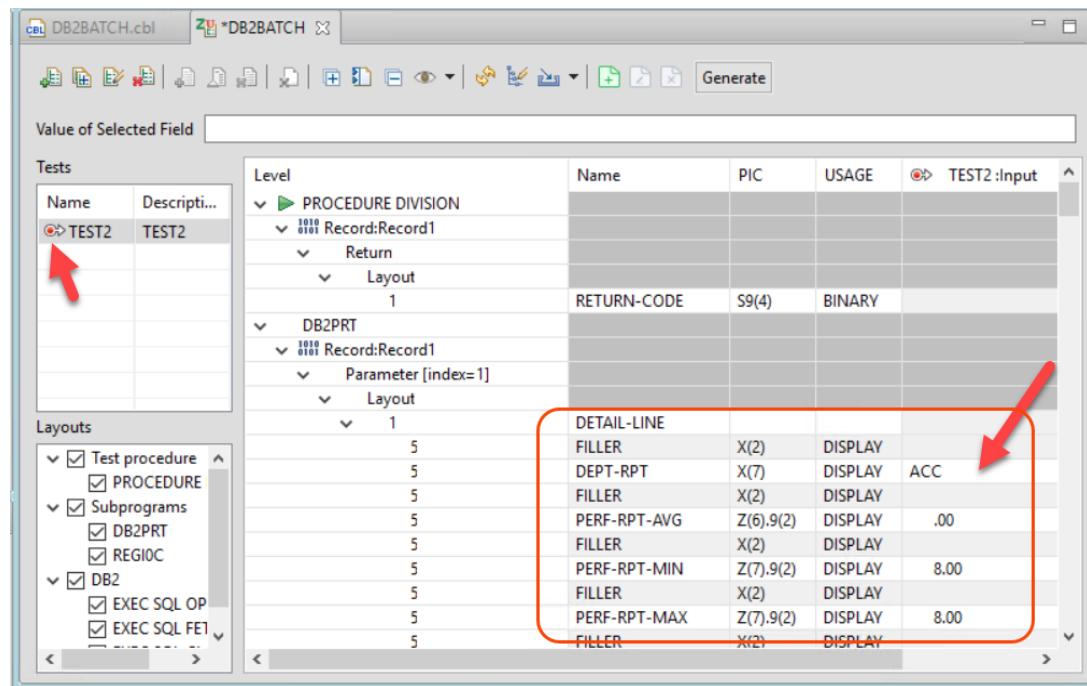
► Click **Yes** to create the playbackfile and import the test data.



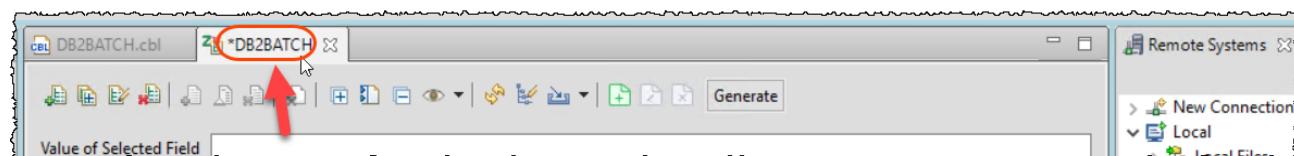
2.2.11 ► Click **OK** after the data is successfully imported to the test data and playback file.



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.

The screenshot shows the IBM zUnit COBOL Test Editor interface. On the left, there's a tree view of 'Tests' and 'Layouts'. Under 'Tests', 'TEST2' is selected. Under 'Layouts', several options are checked, including 'Test procedure', 'PROCEDURE DIVISION', 'Subprograms' (with 'DB2PRT' checked), 'REGIOC', and 'DB2' (with 'EXEC SQL OPEN [C1]', 'EXEC SQL FETCH [C1]', and 'EXEC SQL CLOSE [C1]' checked). The main pane displays a table with data for the DB2PRT subprogram. A red box highlights the first six rows of the table, which represent the data passed to the subprogram. A red arrow points from the number '1' to the top of this highlighted area. Another red arrow points from the number '1' to the 'Value of Selected Field' input field at the top of the editor.

Level	Name	PIC	USAGE	TEST2 : Input	TEST2 : Expected
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.

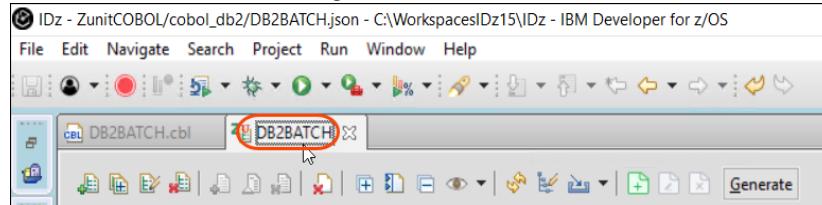
2 Click on **ACC** . and notice that value is displayed on the 3 **Value of Selected Field**

The screenshot shows the IBM zUnit COBOL Test Editor interface. On the left, there's a tree view of 'Tests' and 'Layouts'. Under 'Tests', 'TEST2' is selected. Under 'Layouts', 'DB2' is checked, and 'EXEC SQL OPEN [C1]', 'EXEC SQL FETCH [C1]', and 'EXEC SQL CLOSE [C1]' are also checked. The main pane displays a table with data for the EXEC SQL FETCH statement. A red box highlights the row for 'INTO'. A red arrow points from the number '2' to the 'ACC' value in the 'DISPLAY' column of this row. Another red arrow points from the number '3' to the 'Value of Selected Field' input field at the top of the editor. A red circle labeled '1' is placed near the 'EXEC SQL FETCH [C1]' checkbox in the tree view.

Level	Name	PIC	USAGE	TEST2 : Input	TEST2 : Expected
1	line=135				
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	
3	line=199				
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.

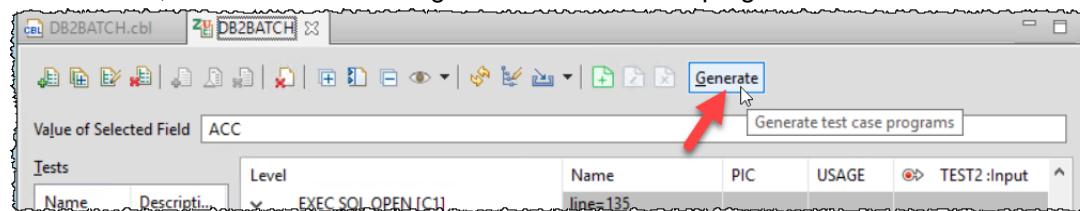


Section 3. Generate, build, and run the unit test generated program.

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

3.1 Generating the COBOL test case programs.

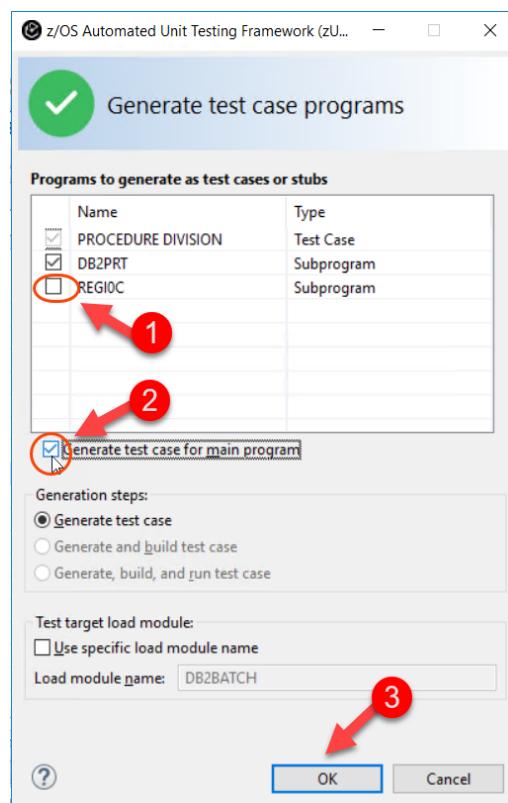
3.1.1 ►► Now that the expected input and output values have been set in the test case editor from the recorded run, click on **Generate** to generate the test case programs.



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**.

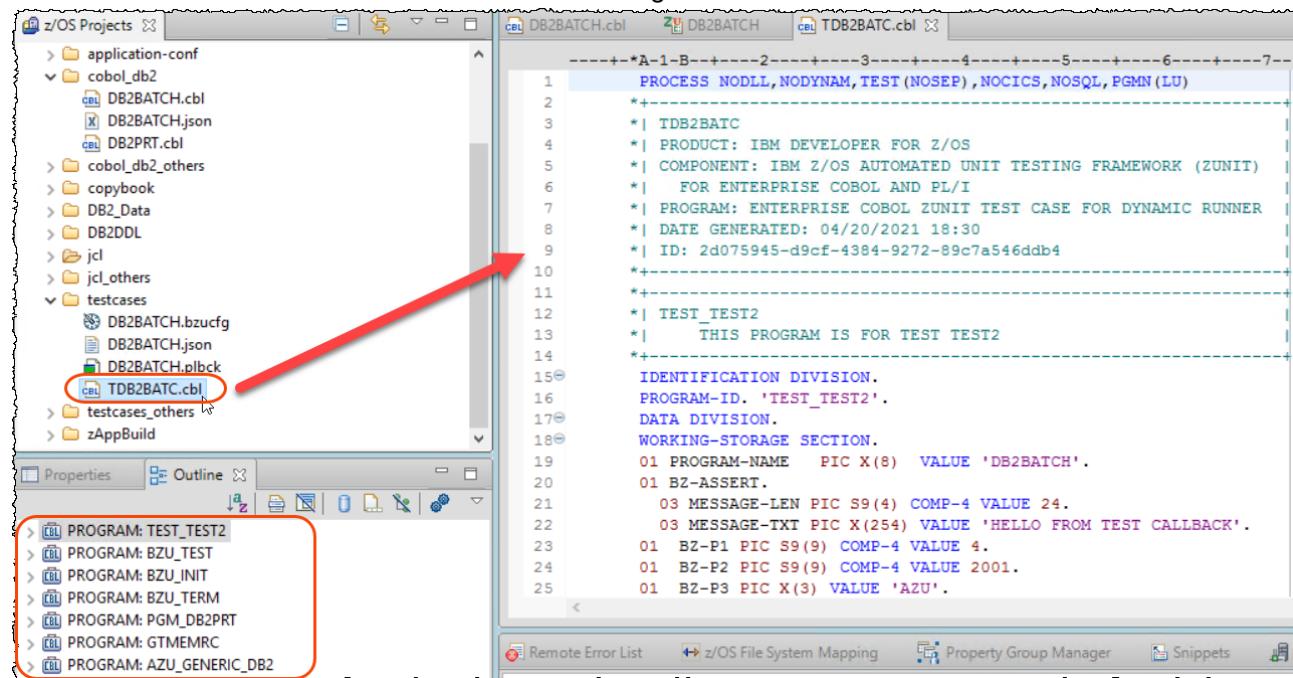


3.1.3 The COBOL programs that will run the test case are generated under the folder **testcases**.

► 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 z/OS Studio interface. On the left, the 'z/OS Projects' view displays a project structure with several subfolders like 'application-conf', 'cobol_db2', 'testcases', etc. Inside 'testcases', there are files: 'DB2BATCH.bzucfg', 'DB2BATCH.json', 'DB2BATCH.plbck', and 'TDB2BATC.cbl'. The 'Outline' view on the right shows the contents of 'TDB2BATC.cbl'. A red arrow points from the 'Outline' view to the 'Properties' view at the bottom left, which lists the seven generated programs: 'PROGRAM: TEST_TEST2', 'PROGRAM: BZU_TEST', 'PROGRAM: BZU_INIT', 'PROGRAM: BZU_TERM', 'PROGRAM: PGM_DB2PRT', 'PROGRAM: GTMEMRC', and 'PROGRAM: AZU_GENERIC_DB2'. The 'Properties' view has a red box around it.

```

-----*A-1-B-----2-----3-----4-----5-----6-----7-----
1      PROCESS NODLL,NODYNAM,TEST(NOSEP),NOCICS,NOSQL,PGMN(LU)
2
3      *+
4      *| TDB2BATC
5      *| PRODUCT: IBM DEVELOPER FOR Z/OS
6      *| COMPONENT: IBM Z/OS AUTOMATED UNIT TESTING FRAMEWORK (ZUNIT)
7      *| FOR ENTERPRISE COBOL AND PL/I
8      *| PROGRAM: ENTERPRISE COBOL ZUNIT TEST CASE FOR DYNAMIC RUNNER
9      *| DATE GENERATED: 04/20/2021 18:30
10     *| ID: d2075945-d9cf-4384-9272-89c7a546dd4
11
12     *+
13     *| TEST_TEST2
14     *| THIS PROGRAM IS FOR TEST TEST2
15     *+
16     IDENTIFICATION DIVISION.
17     PROGRAM-ID. 'TEST_TEST2'.
18     DATA DIVISION.
19     WORKING-STORAGE SECTION.
20     01 PROGRAM-NAME PIC X(8) VALUE 'DB2BATCH'.
21     01 BZ-ASSERT.
22       03 MESSAGE-LEN PIC S9(4) COMP-4 VALUE 24.
23       03 MESSAGE-TXT PIC X(254) VALUE 'HELLO FROM TEST CALLBACK'.
24     01 BZ-P1 PIC S9(9) COMP-4 VALUE 4.
25     01 BZ-P2 PIC S9(9) COMP-4 VALUE 2001.
26     01 BZ-P3 PIC X(3) VALUE 'AZU'.

```

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

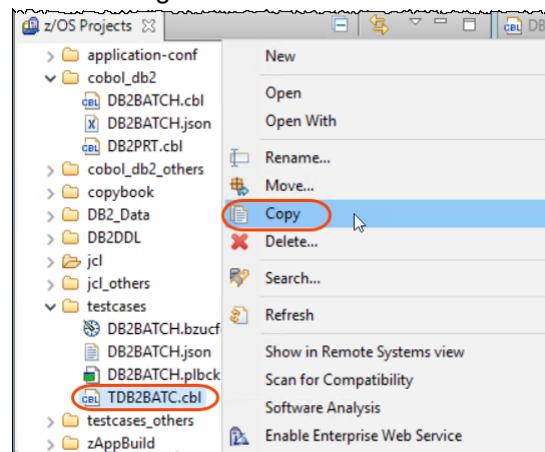
The 7 generated COBOL programs need to be compiled/link edited to be executed. This must be done on z/OS using the COBOL compiler.

To make this easier you could use the IBM DBB (Dependency Based Build) that is part of IDz.

But you also could use the traditional JCL once you moved the generated programs to the z/OS (PDS) to be complied and linked. Here we will show the traditional JCL way.

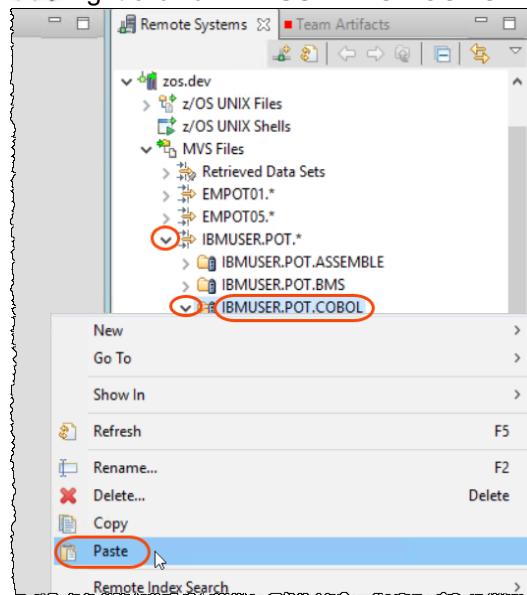
3.2.1 ► Use **Ctrl + Shift + F4** to close all opened editors.

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

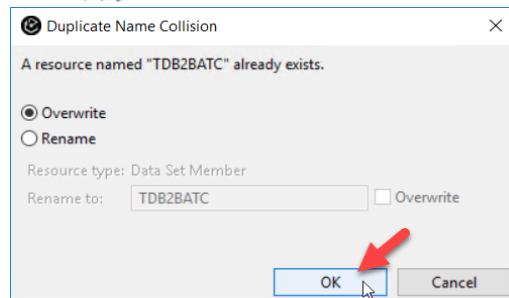


3.2.3 ► Using the Remote systems (on right), expand **IBMUSER.POT.*** and **IBMUSER.POT.COBOL**

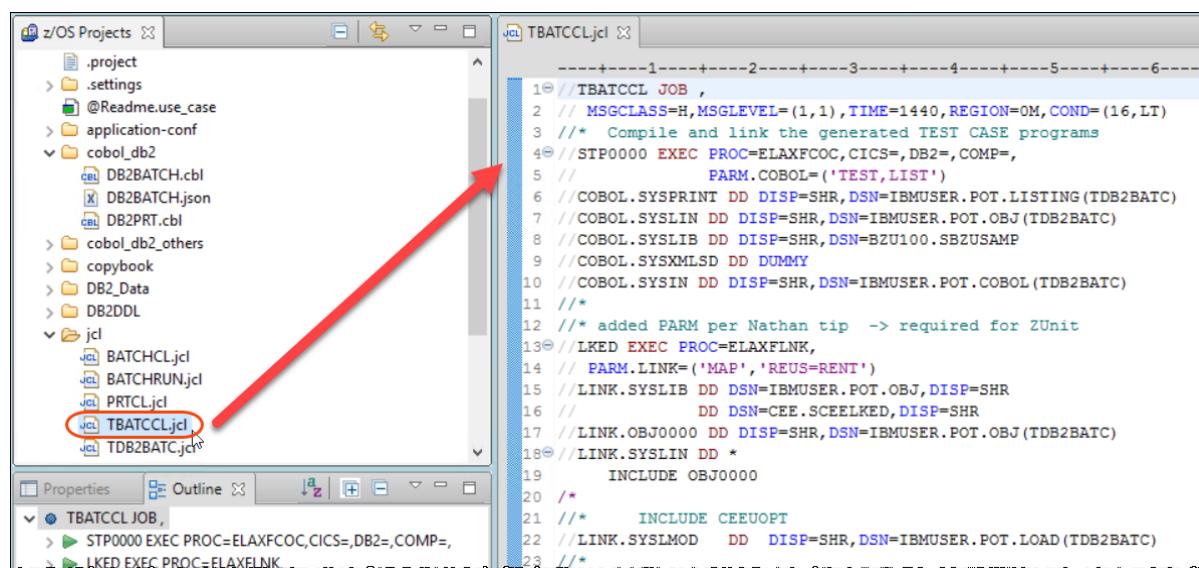
► Right click on **IBMUSER.POT.COBOL** and select **Paste**



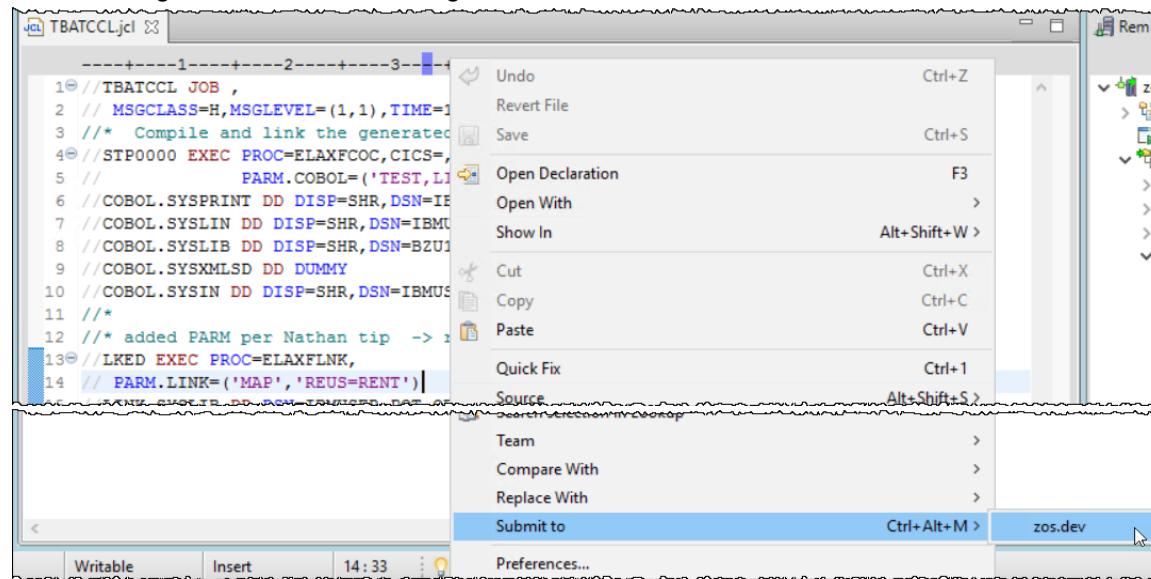
3.2.4 ► Select **OK** to overwrite this member since it already existed on z/OS



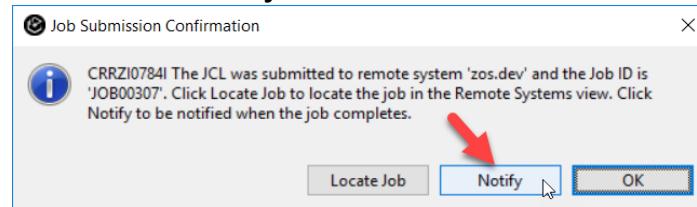
3.2.5 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **TBATCCL.jcl**. This JCL will compile and link the 7 programs that will create the test case load module.



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

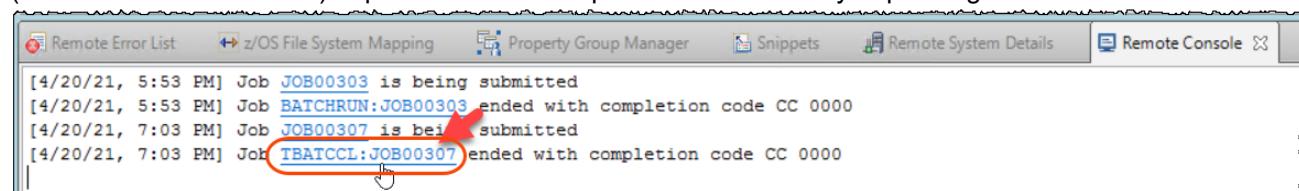


3.2.7 ► Click **Notify** to be notified when the execution is complete.



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

► Once the execution ends, **click on the link TBATCCL:JOB00xxx** (where 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?



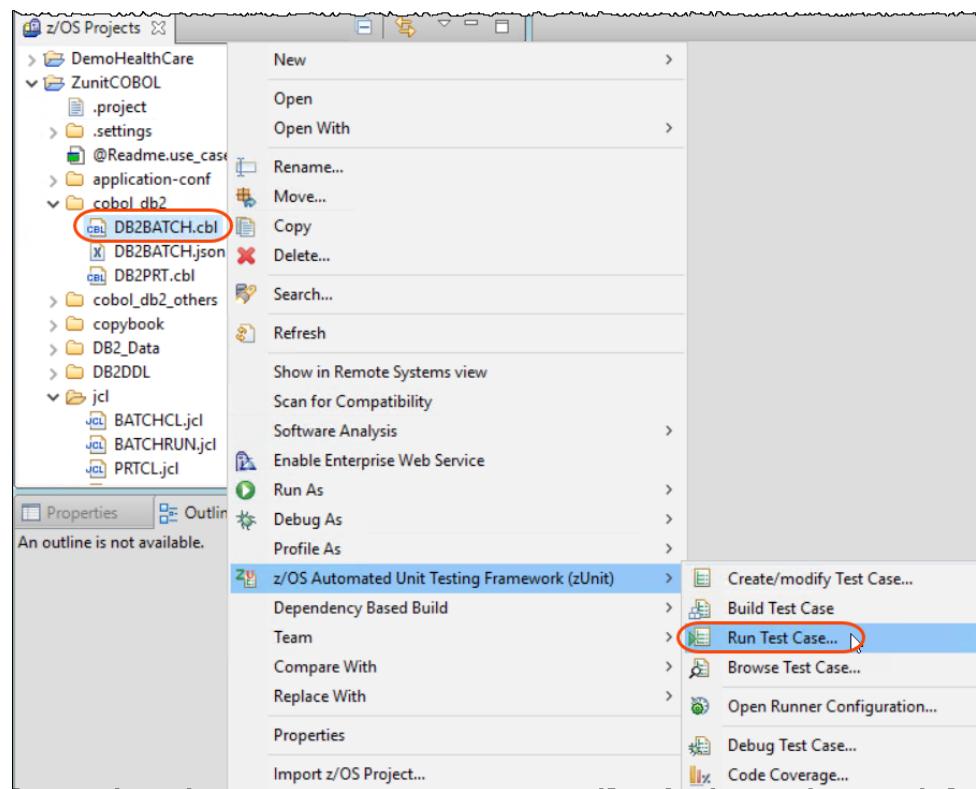
You generated a test case load module (**TDB2BATC**). This load module when executed can verify the correct input and output values handled by the program being verified. The test case can run each time that the program is modified, and the test case must pass it. You will see that

Notice that even though 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,

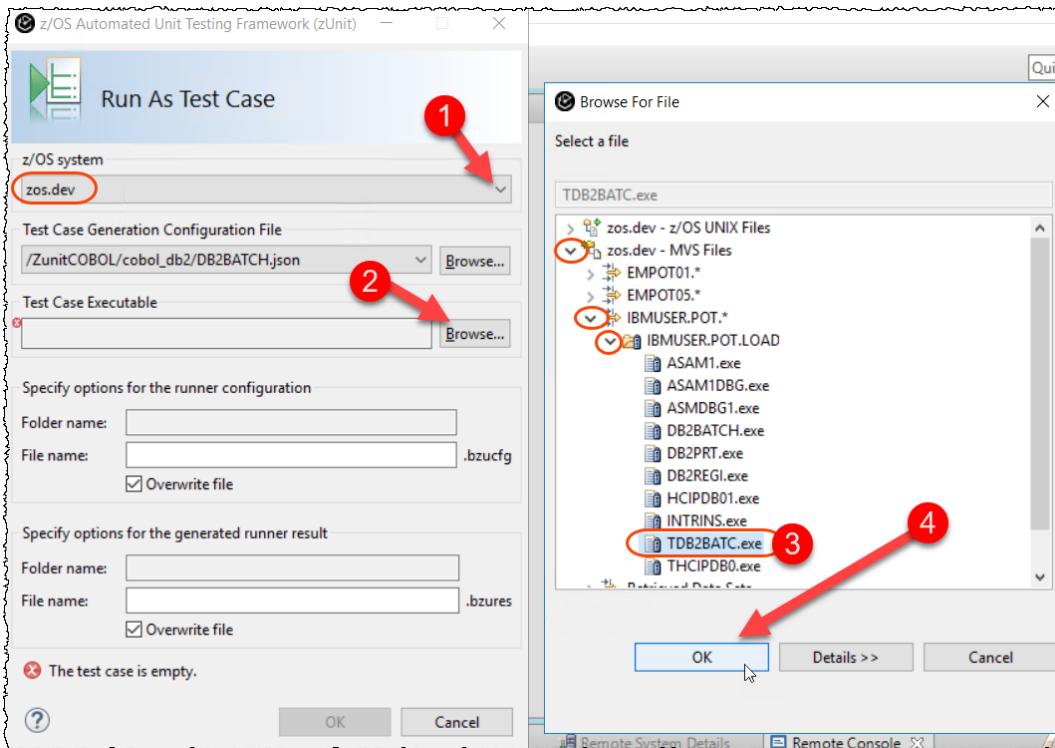
Once you have the test cases load module created you can run the test case against the *DB2BATCH* COBOL program. Since you did not make changes to the program the return code must be zero and all tests should pass.

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...**

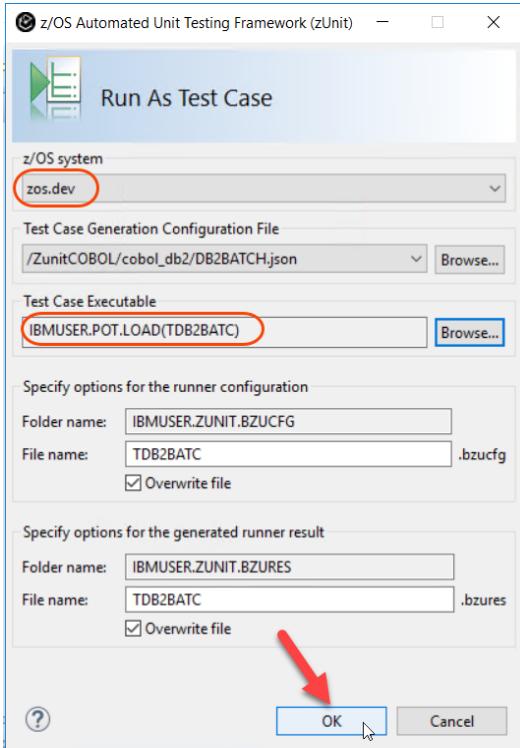


3.3.2  Select **zos.dev** for z/OS system,
Use the second **Browse** button to select **IBMUSER.POT.LOAD (TDB2BATC)**
under **MVS Files** and **IBMUSER.POT.***.

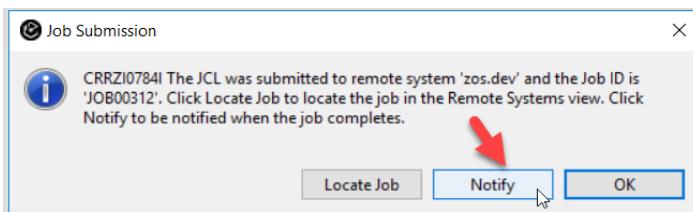
 Click **OK**.



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.

Category	Value
Test count	1
Tests passed	1
Tests failed	0
Tests with errors	0
Tests with severe errors	0

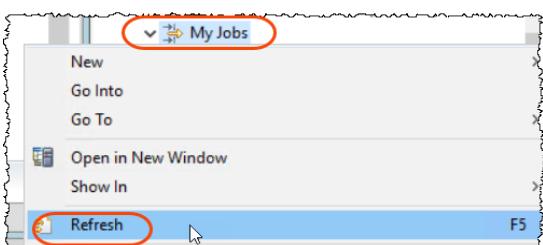
You have now created a test case with data imported from a recorded run and have been successful in testing a COBOL/DB2 batch program without the need of the DB2 environment.

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

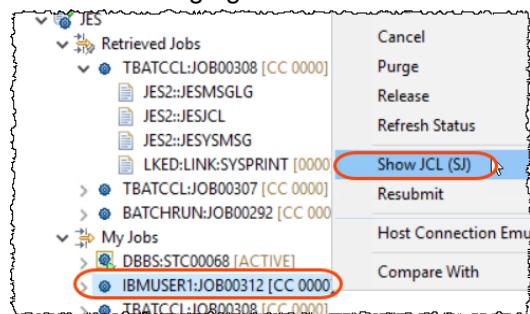
3.4 Verify the JCL submitted that runs the test case

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

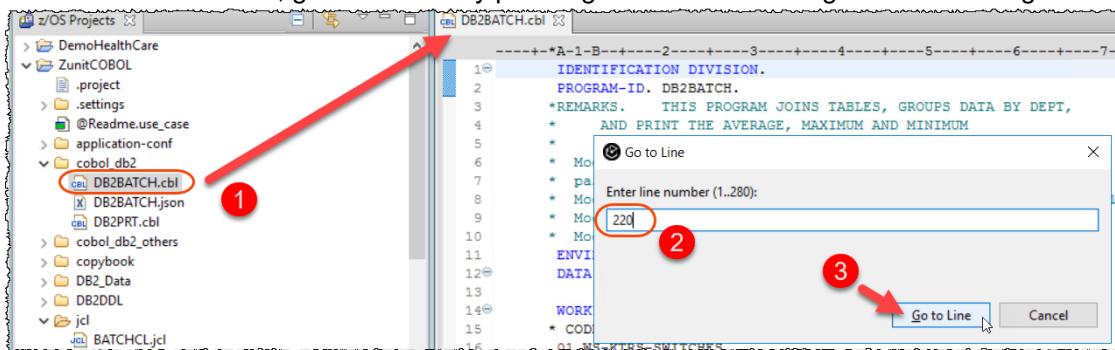
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

4.1 Modifying the program and introduce a bug

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

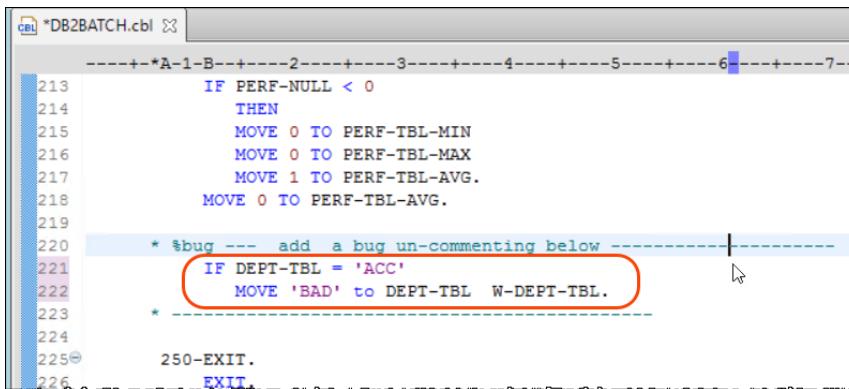
► Open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it in the **z/OS Projects** view.

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.



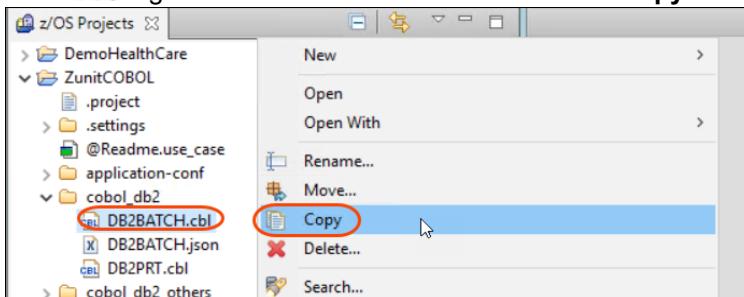
```

DB2BATCH.cbl
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

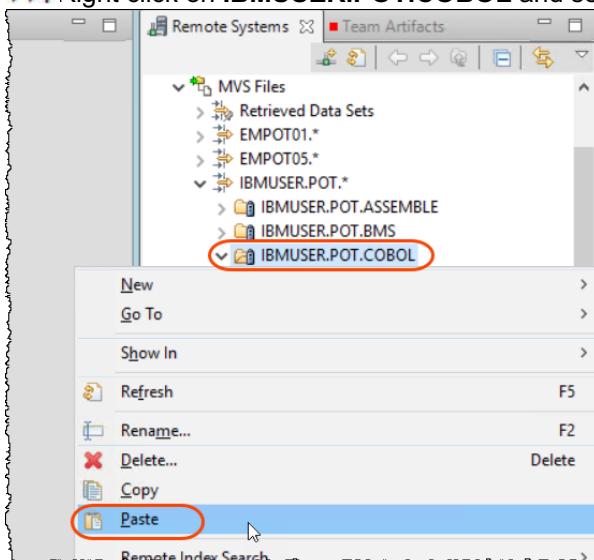
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar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

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

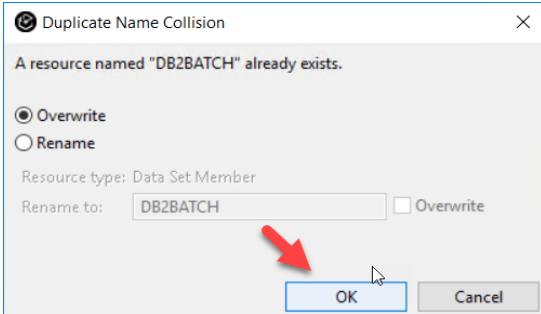


4.2.2 ► Using the Remote systems (on right),

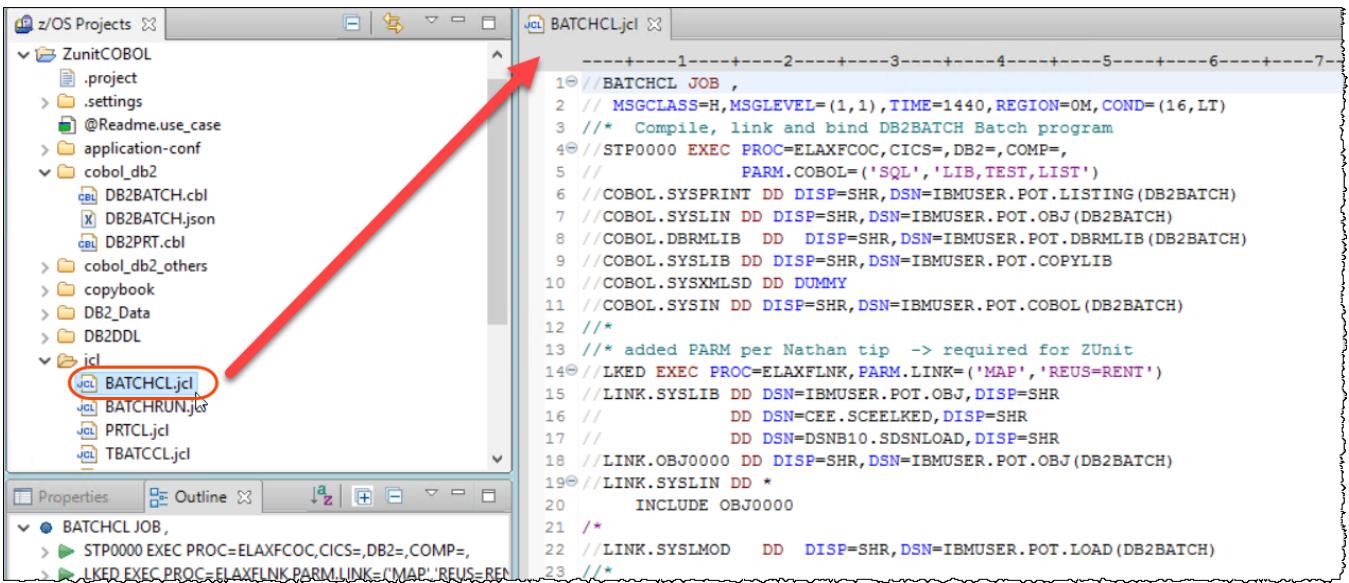
► Right click on **IBMUSER.POT.COBOL** and select **Paste**



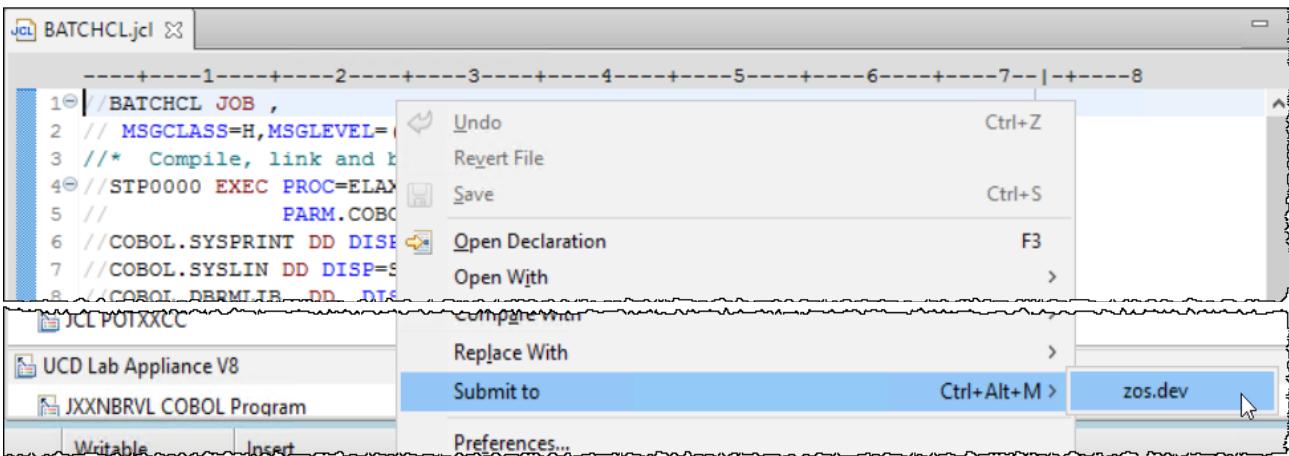
4.2.3 ► Select **OK** to overwrite this member since it already existed on z/OS



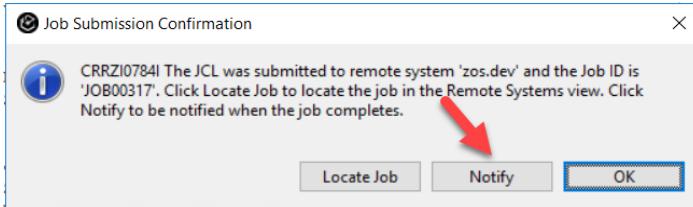
4.2.4 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl**. This JCL will compile, link and bind the program being tested.



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

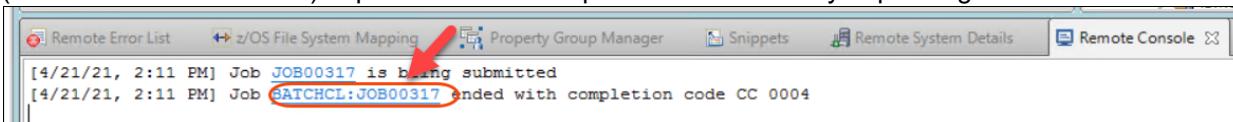


4.2.6 ➡ Click **Notify to be notified when the execution is complete.**



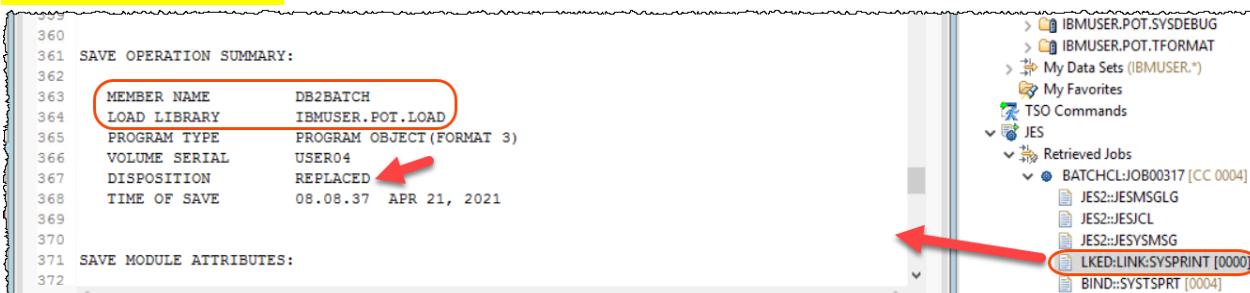
4.2.7 Under **Remote Console, you will be notified when execution is completed. The completion code must be 4.**

➡ 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 **LKED:LINK:SYSPRINT** step.**

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

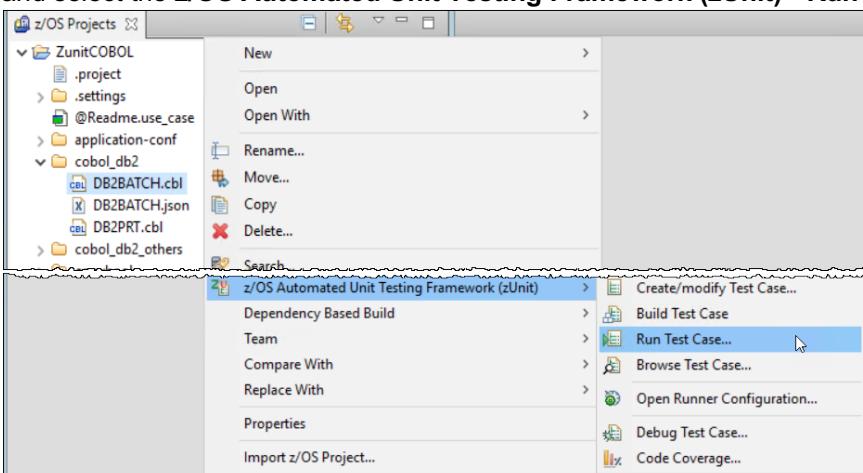


4.2.9 ➡ Use **Ctrl + Shift + F4 to close all opened editors.**

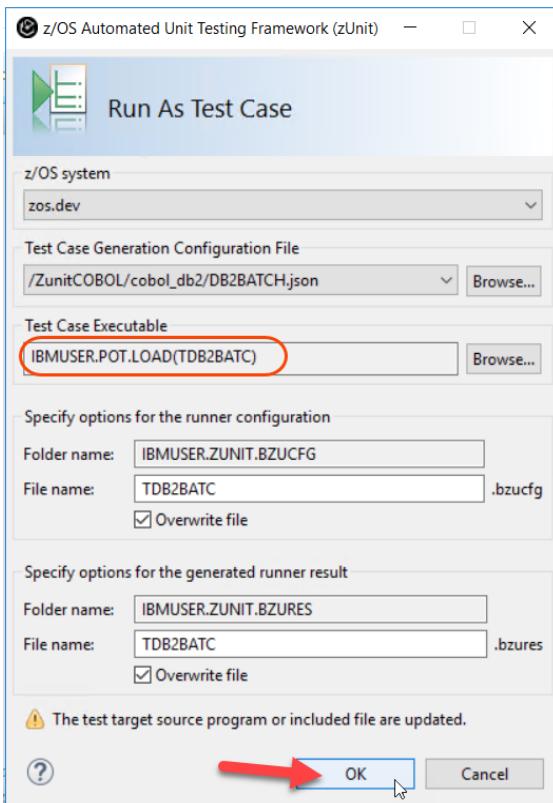
4.3 Running the test case again

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

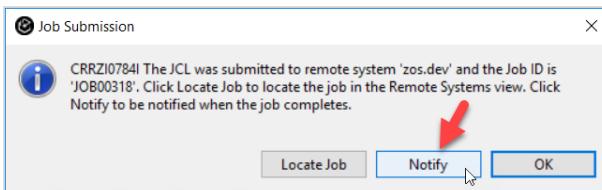
4.3.1 ➡ On the z/OS Projects view, select **DB2BATCH.cbl, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..****



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

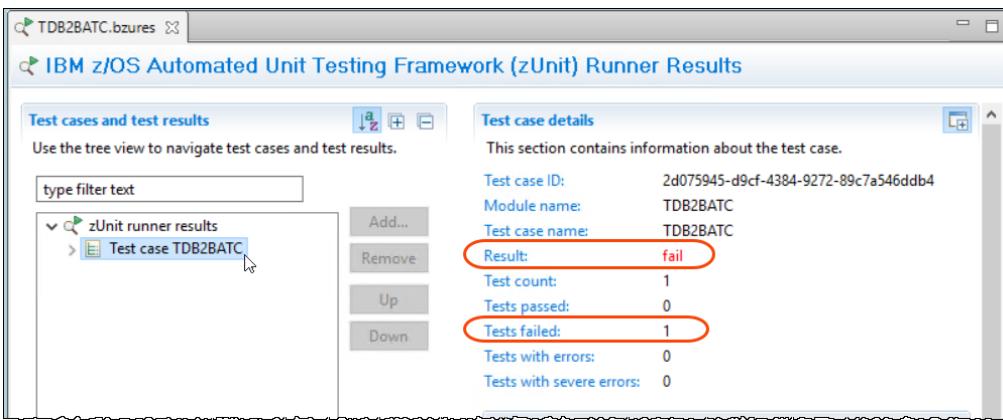


4.3.3 ► Click **Notify** on the Job Submission Confirmation dialog.



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

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



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

The screenshot shows the 'Test cases and test results' pane with the tree view expanded to show 'zUnit runner results', 'Test case TDB2BATC', and 'Test TEST2 (fail)'. Under 'Test TEST2 (fail)', 'Exception message number' is selected and highlighted with a red arrow. In the 'Exception details' pane, there is a message box containing a detailed error log. A red box highlights a specific part of the log where 'EXIT POINT CHANGE FOR=TEST2 MODE=' is followed by several lines of data.

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.

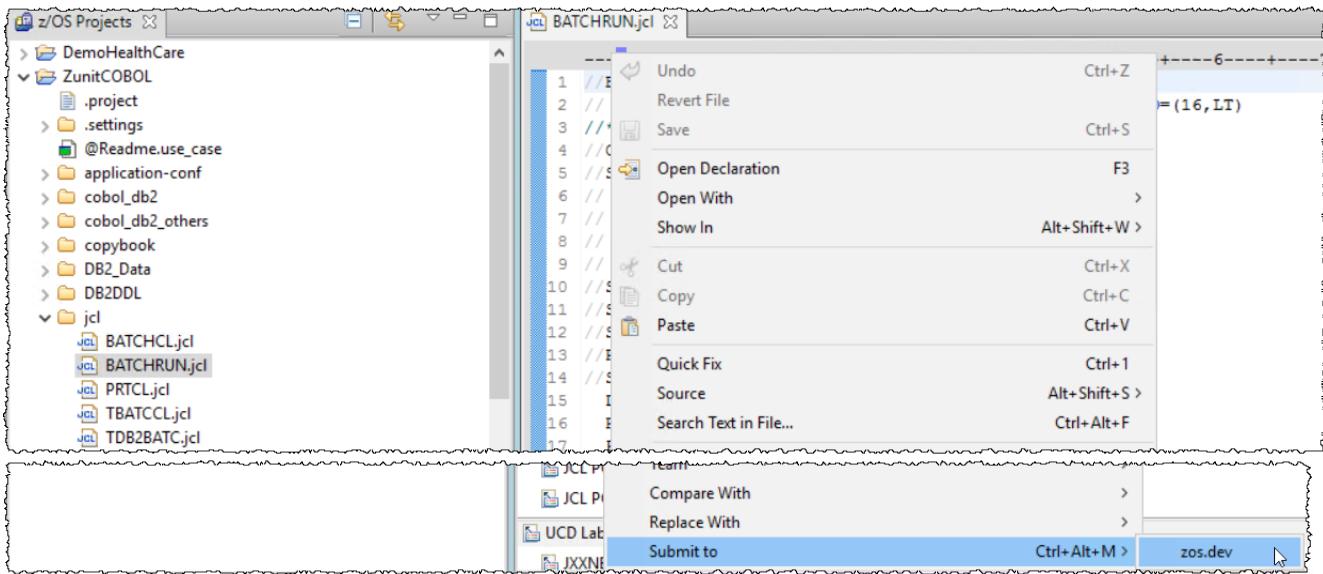
The screenshot shows the 'z/OS Projects' interface with the 'ZunitCOBOL' project selected. The 'jcl' folder is expanded, and the 'BATCHRUN.jcl' file is highlighted with a red circle and a red arrow pointing to it. The right-hand panel displays the JCL code for the 'BATCHRUN' job, which includes various DD statements and system parameters.

```

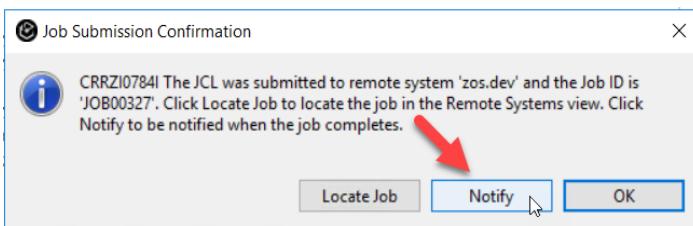
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7-----
1 // BATCHRUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 //   Execute DB2BATCH Batch program
4 // CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 // STEPLIB  DD DSN=IBMUSER.POT.LOAD,DISP=SHR
6 //           DSN=DSNB10.SDSNLOAD,DISP=SHR
7 //           DSN=DSNB10.DBG.RUNLIB.LOAD,DISP=SHR
8 //           DISP=SHR,DSN=BZU100.SBZULOAD
9 //           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(DBBG)
16 // RUN PROGRAM(DB2BATCH) -
17 // PLAN(DB2BATCH)
18 END

```

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

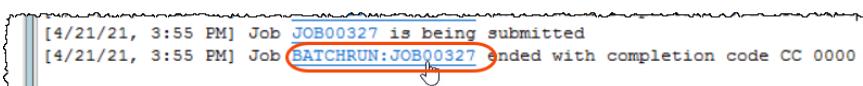


5.1.3 ► Click **Notify** to be notified when the execution is complete.

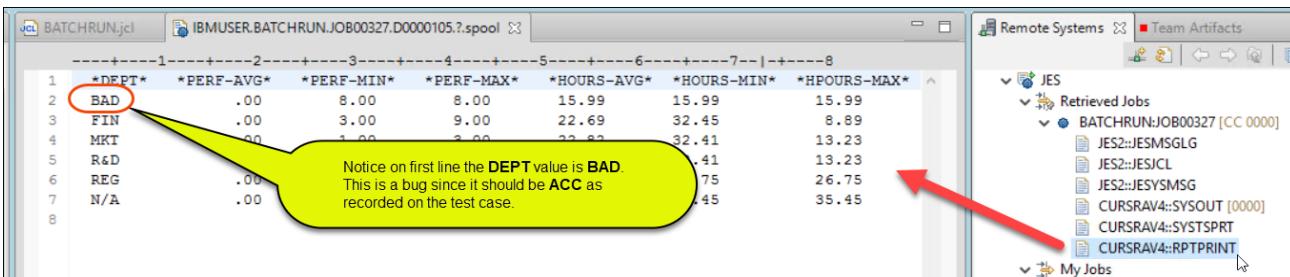


5.1.4 Under *Remote Console*, you will be notified when execution is completed.

► 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**,

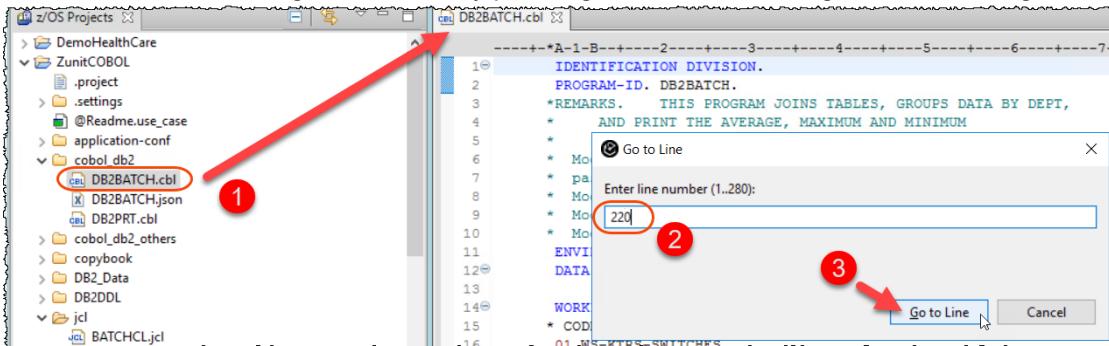
Section 6. Use IDz to fix the bug, recompile the COBOL/DB2 program.

You will fix the using IDz, and rebuilt the executable

6.1 Modifying the program to eliminate the bug

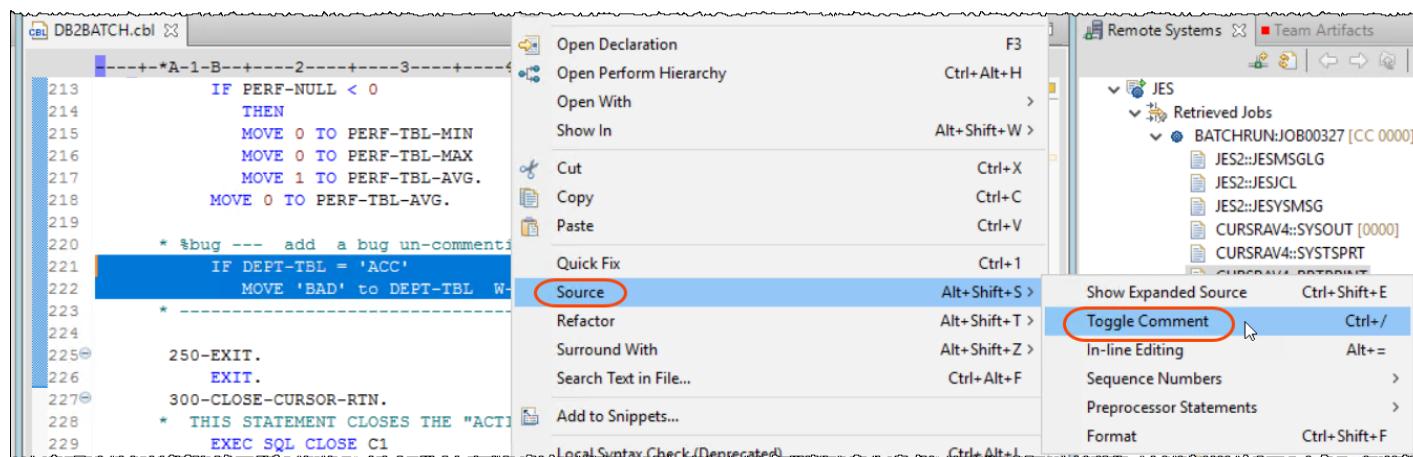
6.1.1 ► Using z/OS Projects view open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it

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

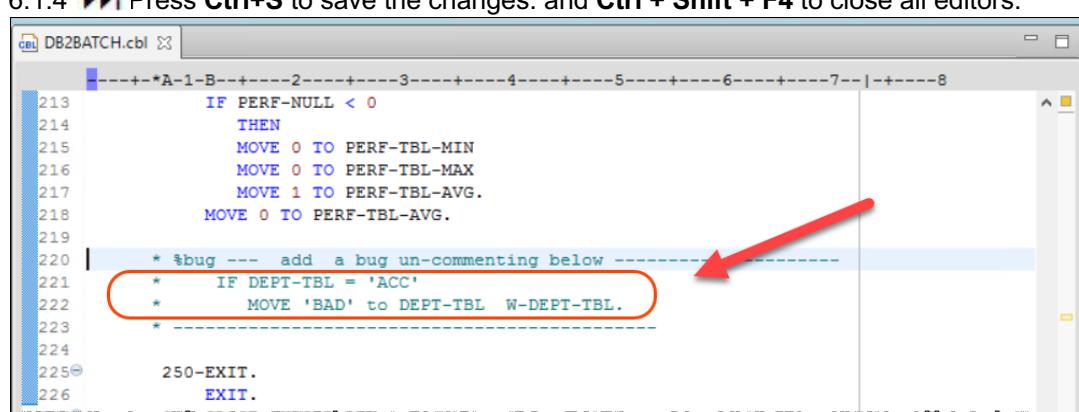


6.1.3 ► Change the lines 221 and 222 adding an * on the column 7.

The easiest way to do that is **selecting those 2 lines**, right click and select **Source > Toggle Comment**



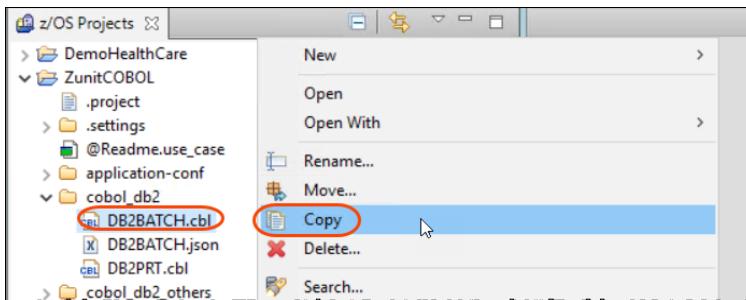
6.1.4 ► Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



6.2 Re-compile and link the changed program

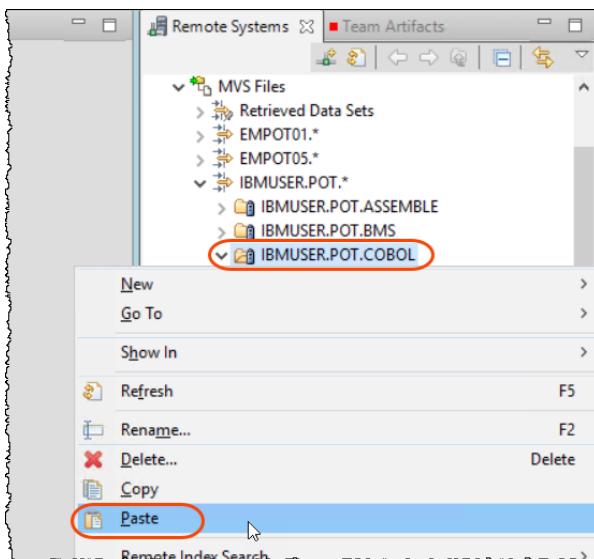
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the “old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar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

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

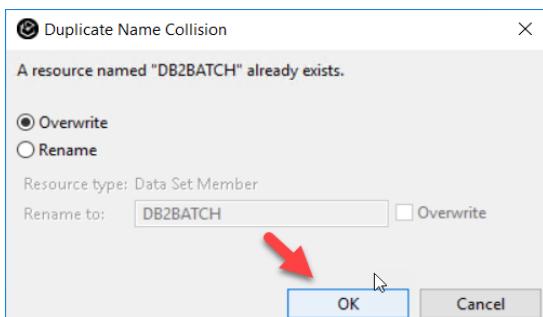


6.2.2 Using the Remote systems (on right),

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



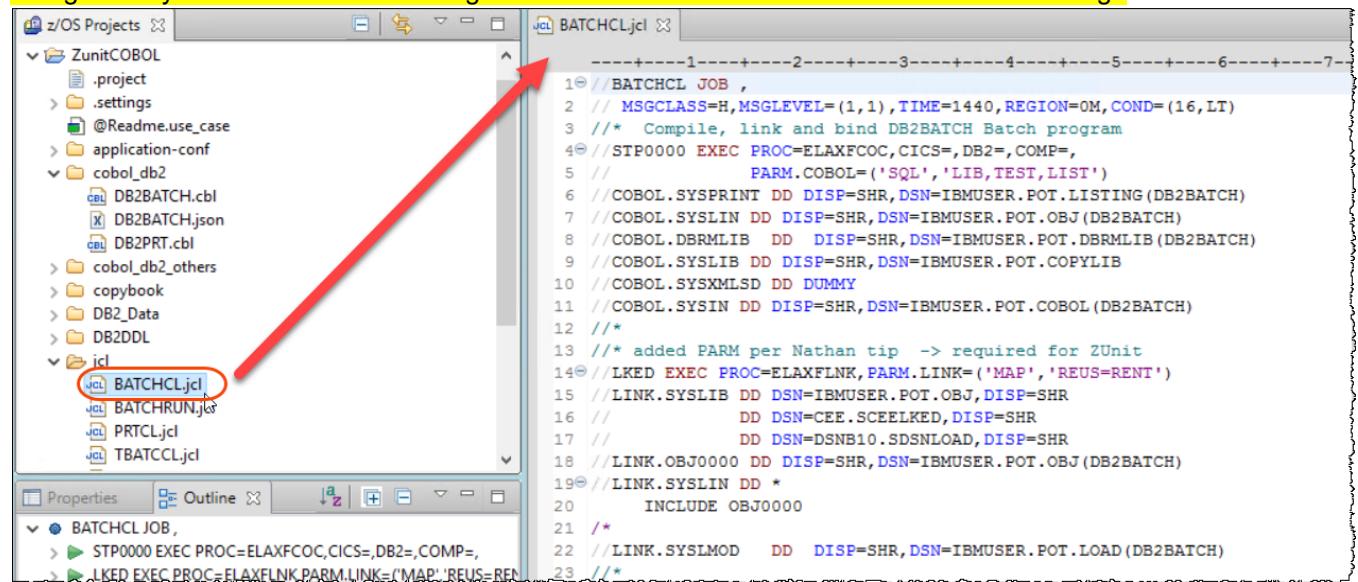
6.2.3 Select **OK** to overwrite this member since it already existed on z/OS



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..



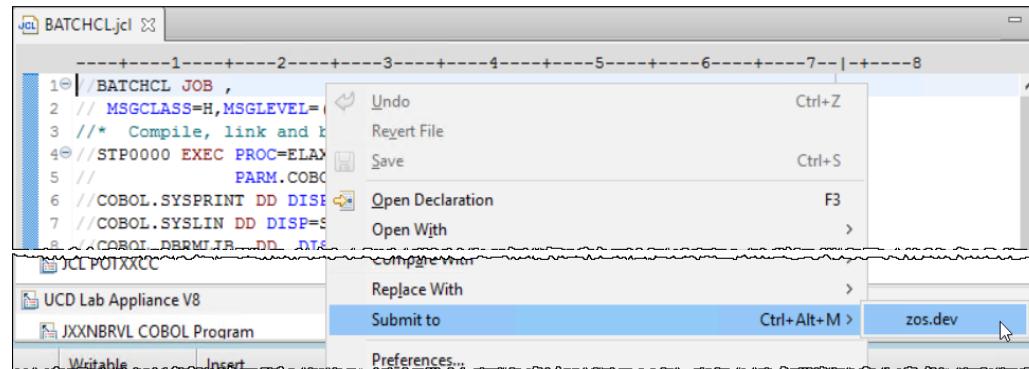
The screenshot shows the IBM Rational Developer for z/OS interface. On the left, the 'z/OS Projects' view displays the ZunitCOBOL project structure, including subfolders like .project, .settings, @Readme.use_case, application-conf, cobol_db2, and icl. Inside the icl folder, several JCL files are listed: BATCHCL.jcl, BATCHRUN.jcl, PRTCL.jcl, and TBATCL.jcl. The BATCHCL.jcl file is highlighted with a red circle and has a red arrow pointing from the project tree towards it. The main editor window on the right shows the content of the BATCHCL.jcl file:

```

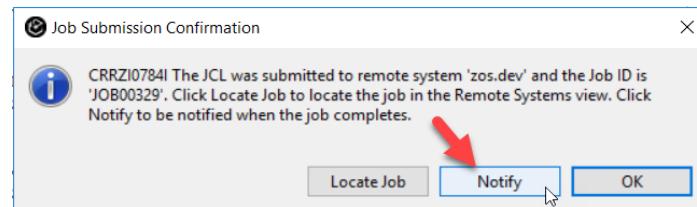
1 //BATCHCL JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=0M,COND=(16,LT)
3 /* Compile, link and bind DB2BATCH Batch program
4 //STP0000 EXEC PROC=ELAXFCOC,CICS=,DB2=,COMP=
5 //
6 //PARM.COBOL=(SQL,'LIB,TEST,LIST')
7 //COBOL.SYSPRINT DD DISP=SHR,DSN=IBMUSER.POT.LISTING(DB2BATCH)
8 //COBOL.SYSLIN DD DISP=SHR,DSN=IBMUSER.POT.OBJ(DB2BATCH)
9 //COBOL.DBRMLIB DD DISP=SHR,DSN=IBMUSER.POT.DBRMLIB(DB2BATCH)
10 //COBOL.SYSLIB DD DISP=SHR,DSN=IBMUSER.POT.COPYLIB
11 //COBOL.SYSXMLSD DD DUMMY
12 ///*
13 /* added PARM per Nathan tip -> required for ZUnit
14 //LKED EXEC PROC=ELAXFLNK,PARM.LINK='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

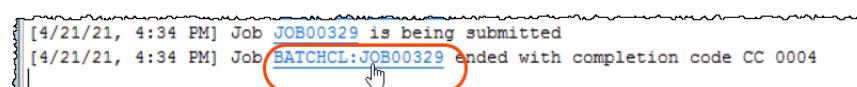


6.2.6 ► Click **Notify** to be notified when the execution is complete.



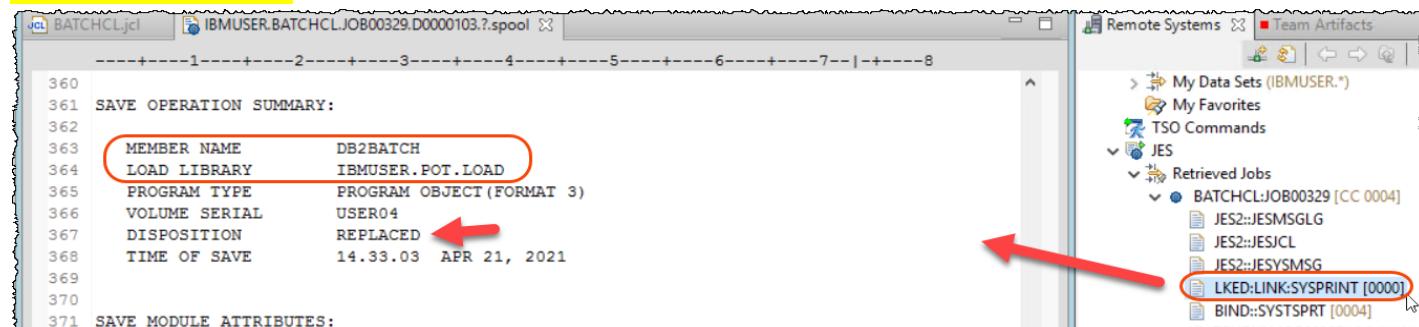
6.2.7 Under **Remote Console**, you will be notified when execution is completed.
The completion code must be 4.

► 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.

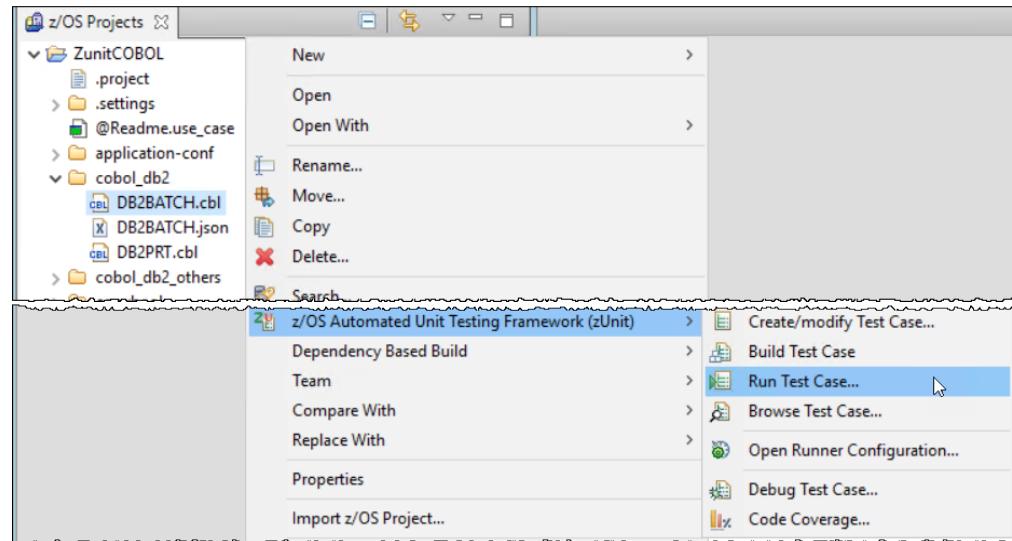
Section 7. Rerun the zUnit and verify that the bug is eliminated.

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

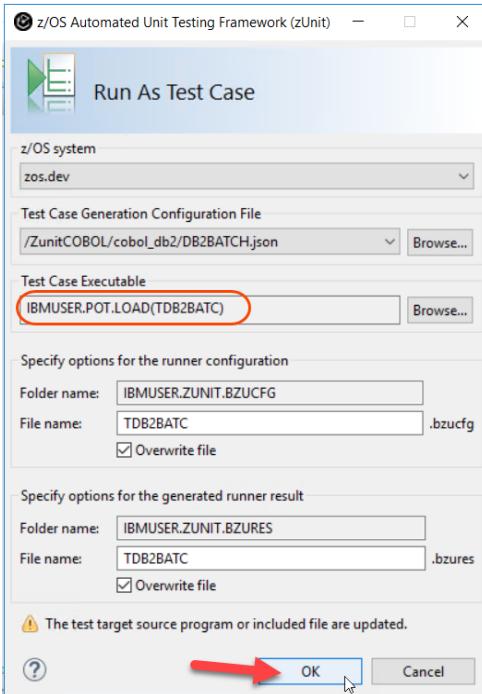
7.1 Running the test case again

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

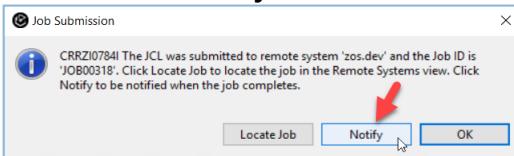
7.1.1 ► On the **z/OS Projects** view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..**



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

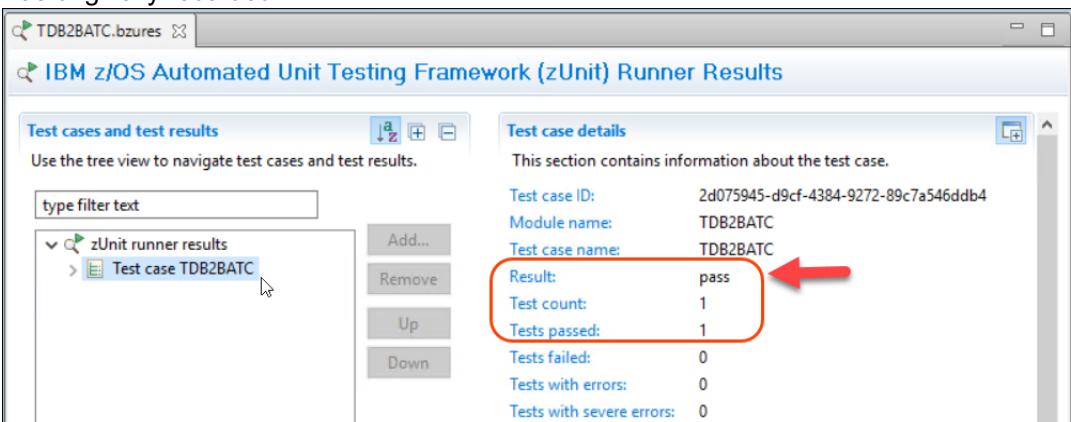


- 7.1.3 ► Click **Notify** on the Job Submission Confirmation dialog.



- 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 3C.

LAB 4 – 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

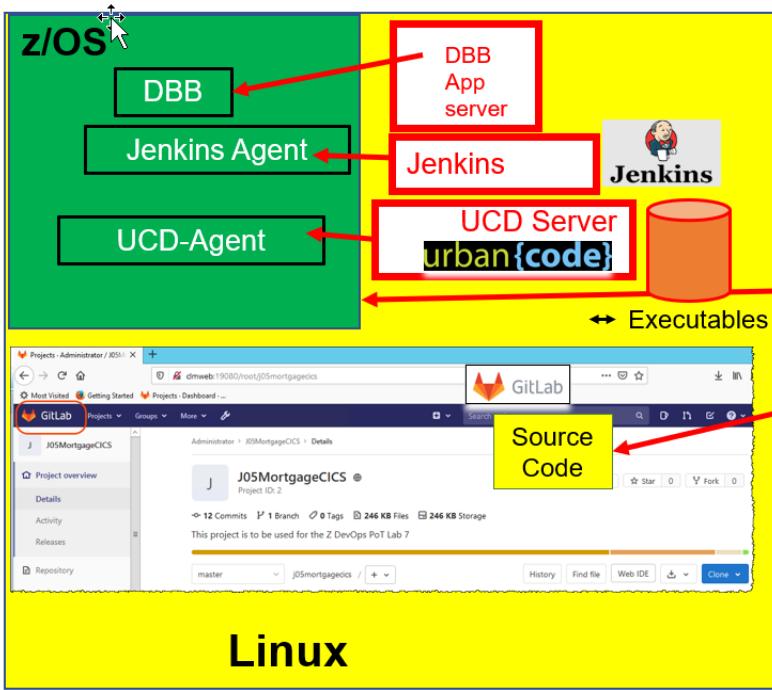
The environment used on this Lab is pictured below.

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

zD&T on Cloud



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

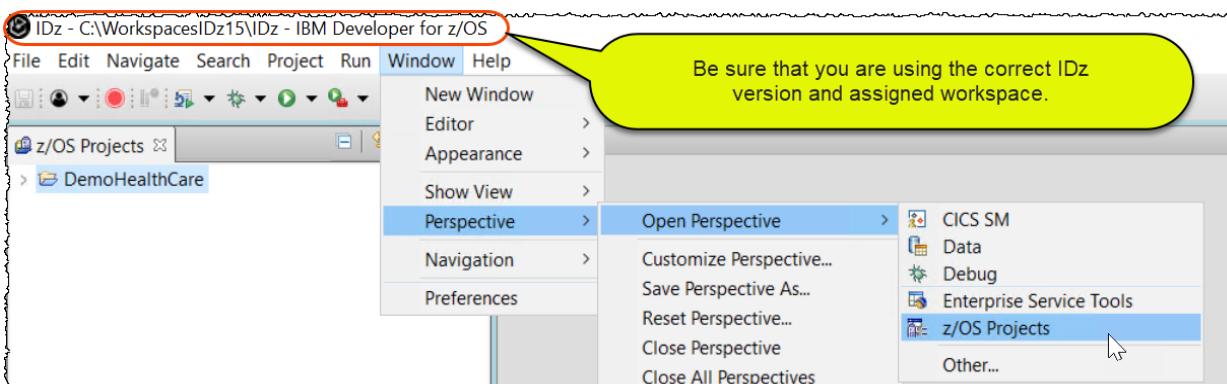
► Using the desktop double click on **IDz V15** icon.

► 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**



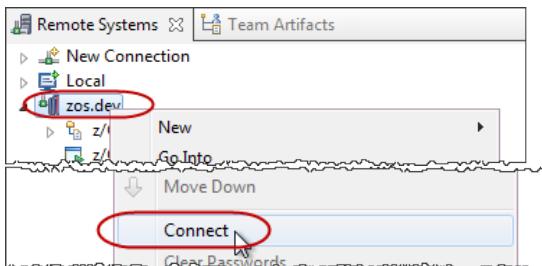
1.1.3 On other labs you used userid **empot01** or **ibmuser**.

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**.



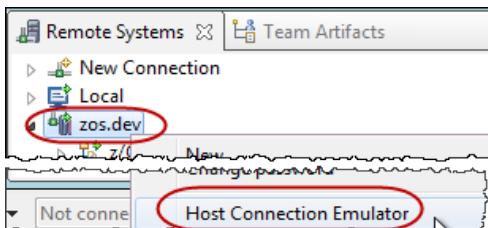
1.1.5 ► Type **empot05** as userid and **empot05** as password.

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

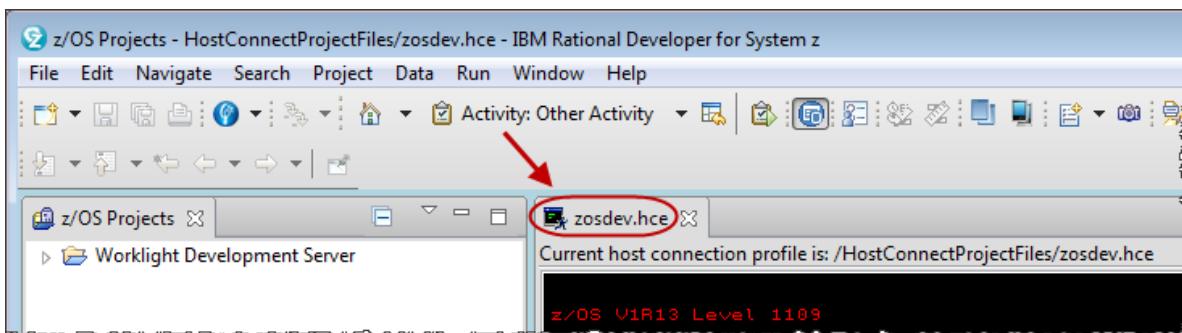
Notice → On this lab your userid will be **empot05** (not empot01 used in previous labs)



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

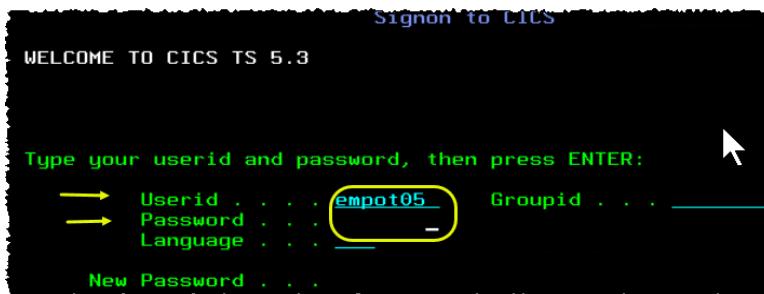


- 1.1.8 ► Type **I cicsts53**. (where "I" is the lower case of letter "L") and press **Enter key**.



```
==> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"
l cicsts53
MA a
24/011
```

- 1.1.9 ► Logon using **empot05** and password **empot05** and press **Enter**.



- 1.1.10 The sign-on message is displayed

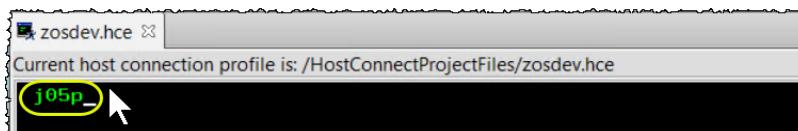


```
DFHCE3549 Sign-on is complete (Language ENU).
MA a
```

1.2 Run CICS transaction J05P

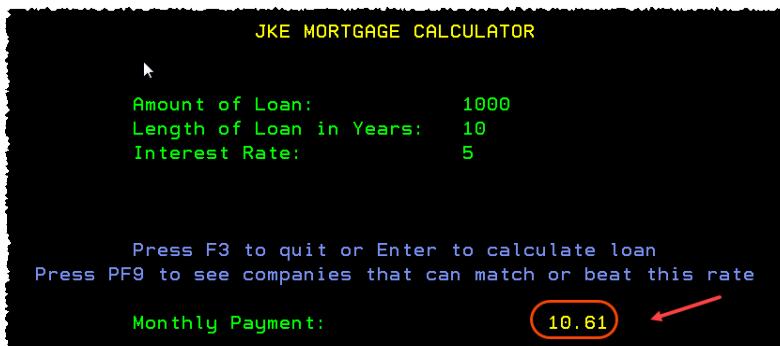
You should now be in the z/OS CICS region named *C/CSTS53*. This is the CICS instance where you will make the program changes.

- 1.2.1 ► Type the CICS transaction **j05p** and press the **Enter key**.



```
zosdev.hce
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
j05p
```

- 1.2.2 ► Press **enter** to see the monthly payment for the default data .



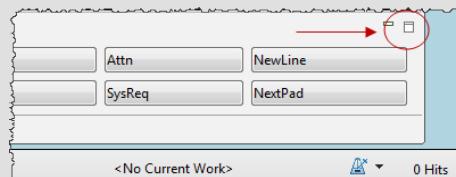
JKE MORTGAGE CALCULATOR

Amount of Loan: 1000
Length of Loan in Years: 10
Interest Rate: 5

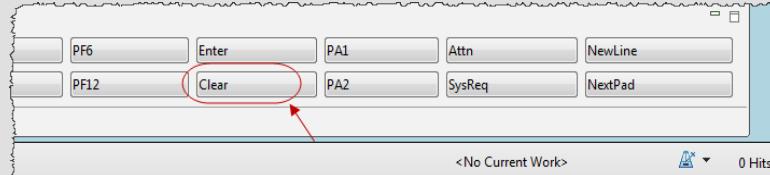
Press F3 to quit or Enter to calculate loan
Press PF9 to see companies that can match or beat this rate

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

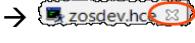
Monthly Payment:      10.61

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

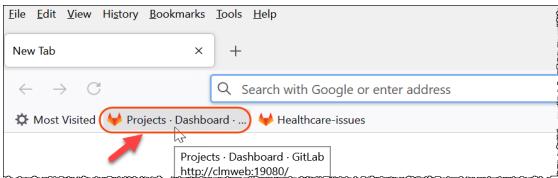
The Mortgage Application used in this lab has its source code in a *Git Repository* that is on Linux system. You must connect to the Git Repository and clone the Git project into the IDz. This brings the source code into your IDz workspace for edits and build. *GitLab* is open source and for that reason 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



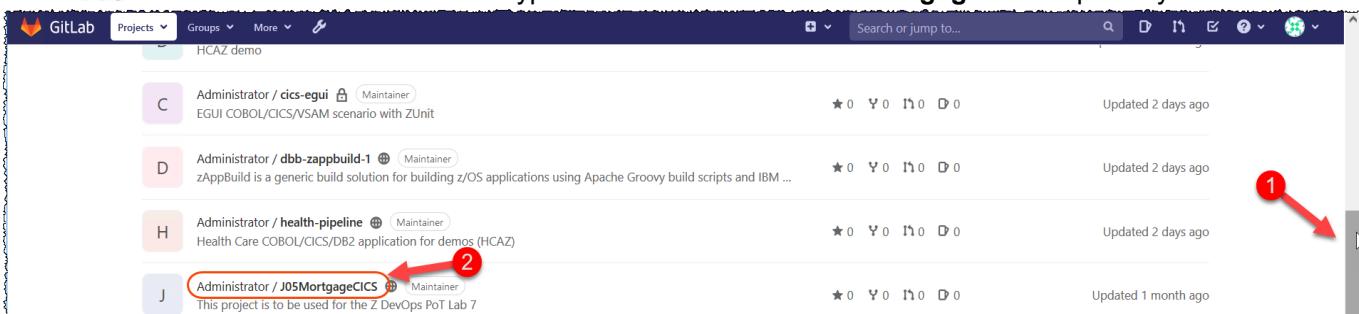
2.1.2 ► Click on this bookmark below to start **GitLab**. The URL used is <http://clmweb:19080/>



► If asked for credentials use **root** and password: **zdtlinux**

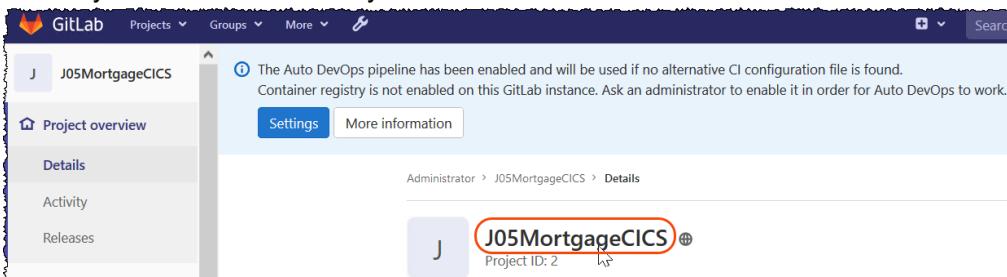


2.1.3 ► Scroll down and click on the hyperlink below to see the **J05MortgageCICS** repository.



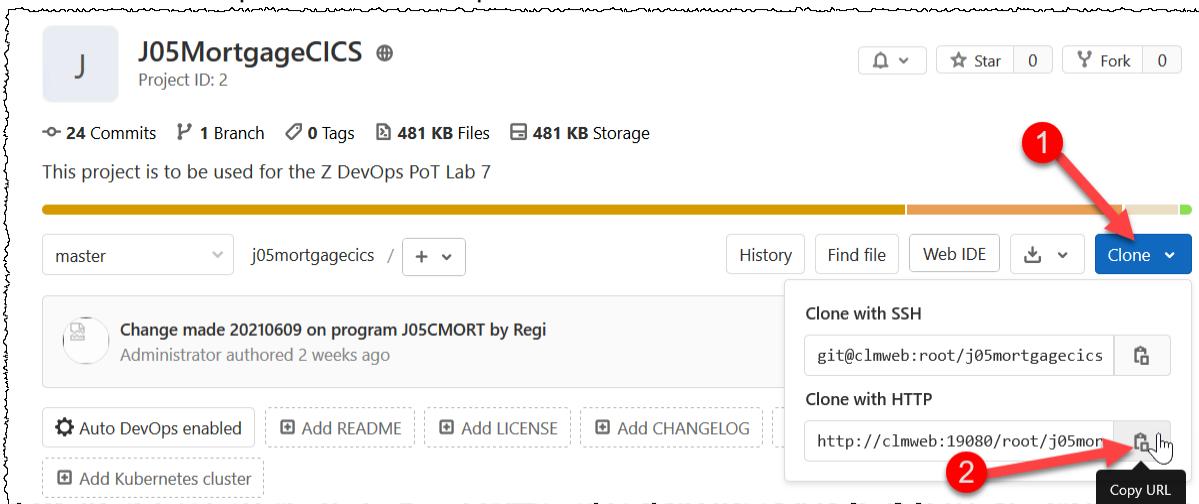
2.1.4 ► You can see the **J05MortgageCICS** repository.

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*.

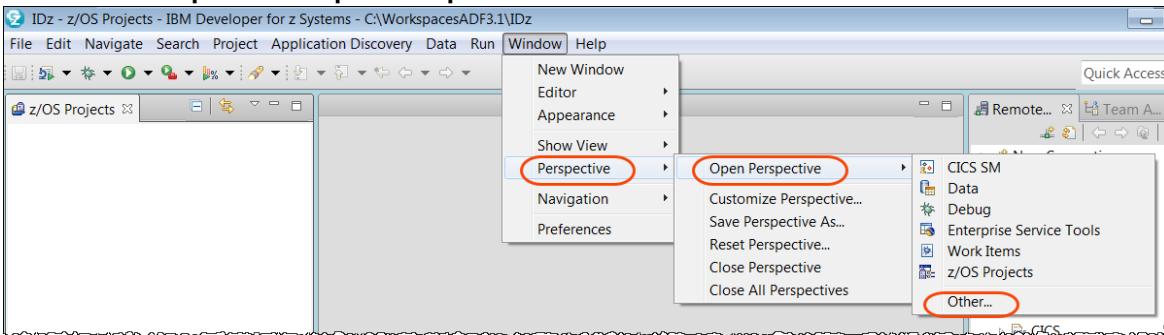
- ▶ 1 Click on **Clone** (the blue button) and 2 in the icon to **copy the URL**
This value will be kept in the windows clipboard.



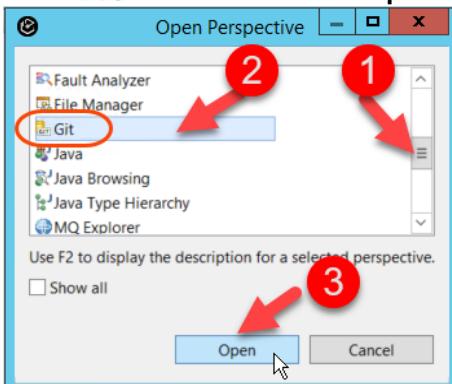
2.1.6 ▶ Go back to IDz using the icon that is on the base of your screen:



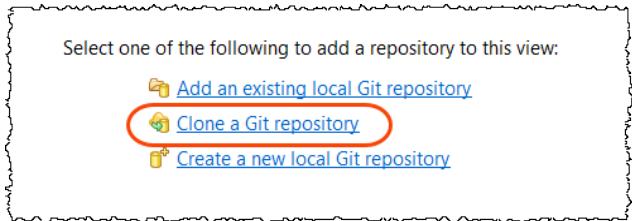
2.1.7 ▶ Open the **Git** perspective by selecting
Window > Perspective > Open Perspective > Other...



2.1.8 ▶ Select **Git** and click **Open**

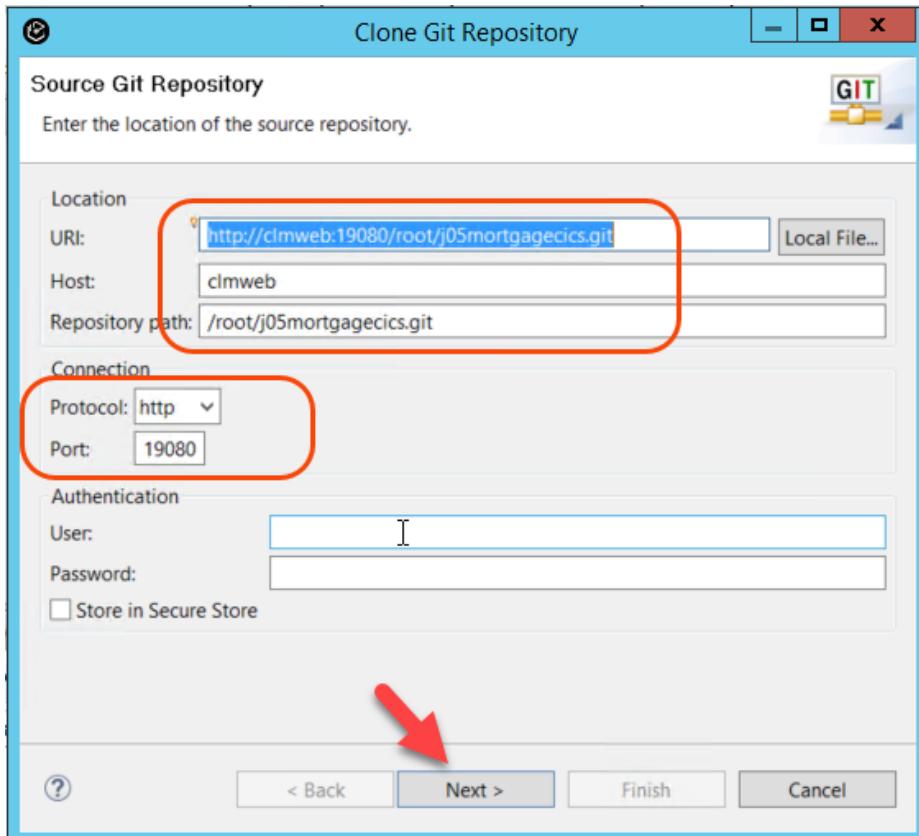


2.1.9 ► In the *Git Repositories* tab, click on the hyperlink to ‘Clone a Git repository’.



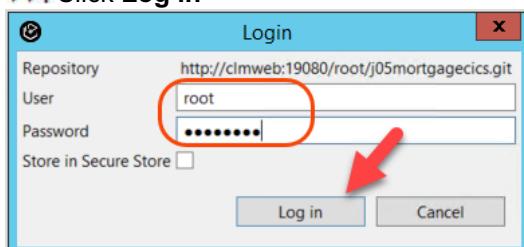
2.1.9 ► The values copied from the web page (copy URL) will be shown in the Clone Git Repository.
Tip: In case you don't see it, got back to the page and copy it again (steps 2.1.1-2.1.5 above).

► Click **Next**



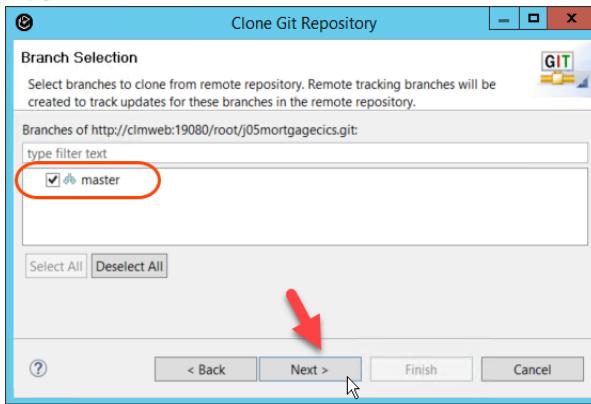
2.1.10 If a dialog asks for login the credentials are a user user: **root** and password **zdtlinux**

► Click **Log in**



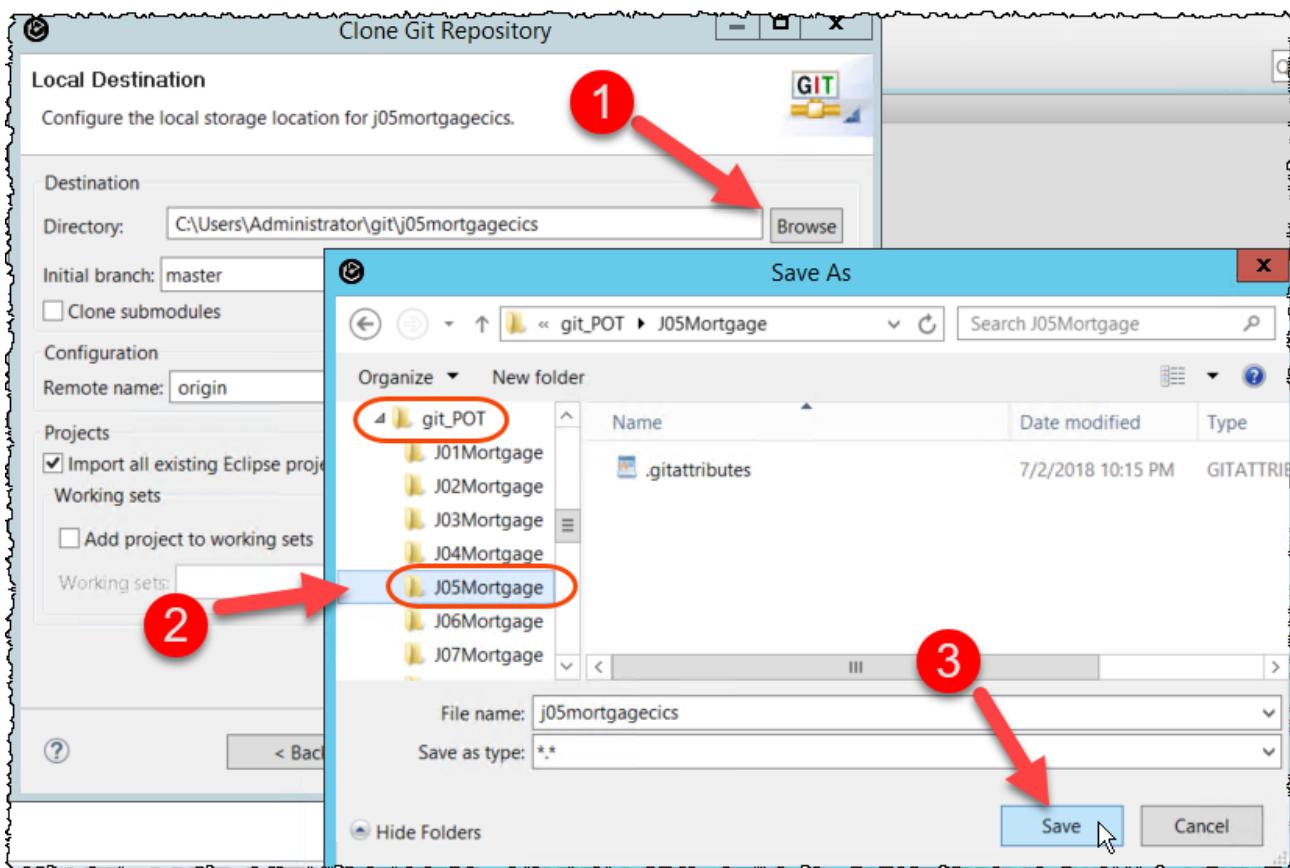
2.1.11 We have only one branch named master.

▶▶ Click **Next**



2.1.12 We will clone the repository on your local windows and import to be shown on IDz

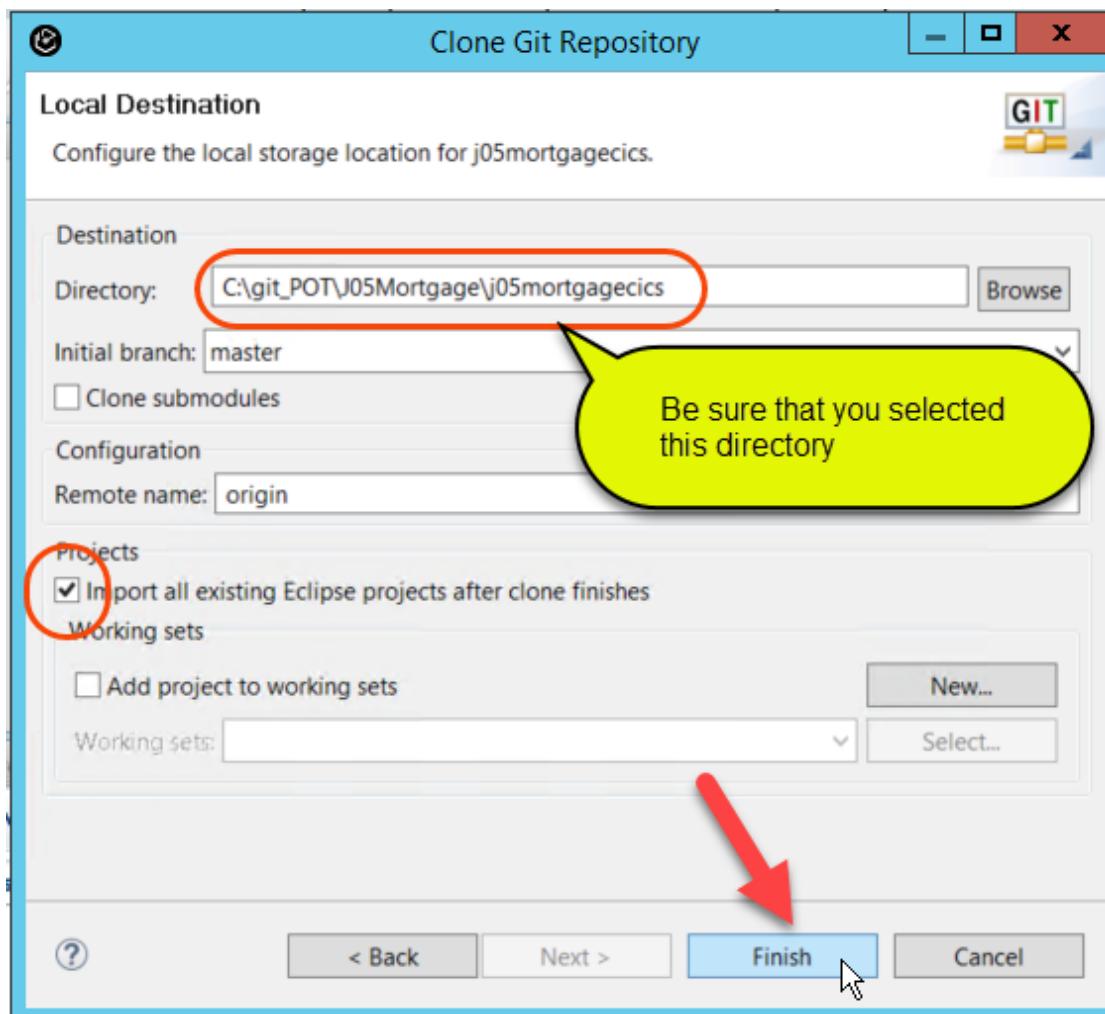
- 1 ▶▶ Use **Browse** button to select the directory **C:\git_POT\J05Mortgage** (on left).
- 2 ▶▶ Click folder **J05Mortgage** on left
- 3 ▶▶ Click **Save**



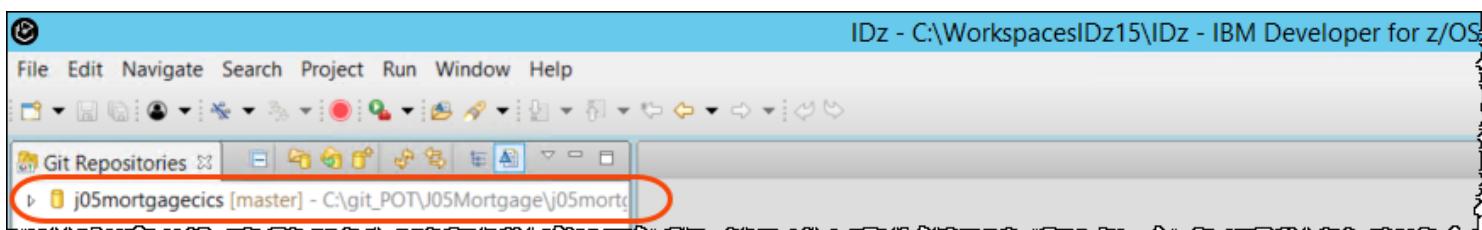
2.1.13 The *Directory* now should point to C:\git_POT\J05Mortgage\j05mortgagecics

►► Be sure that you selected **Import all existing Eclipse projects after clone finishes**

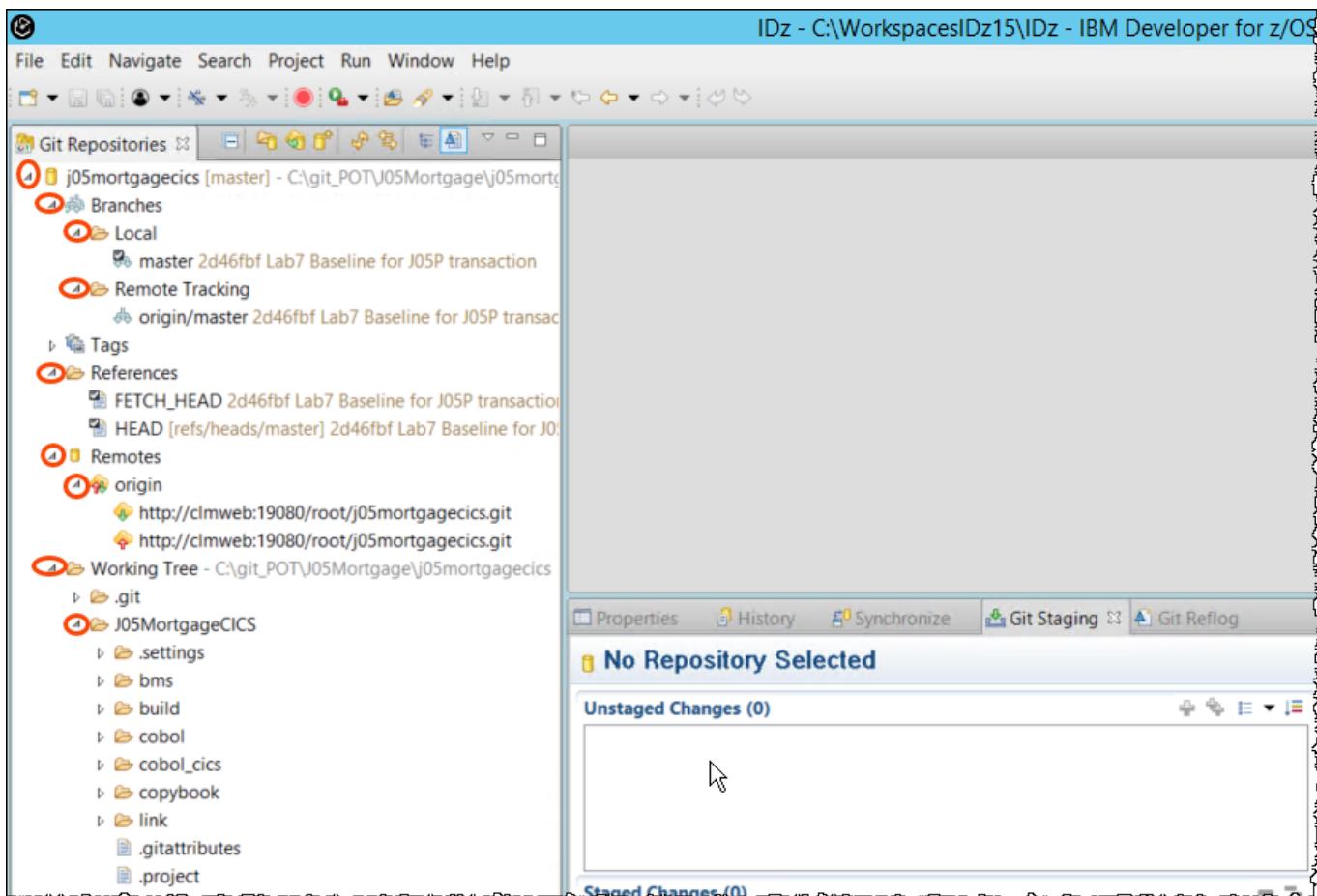
►► Click **Finish**



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:

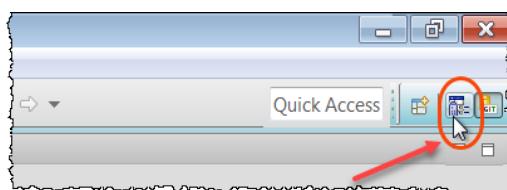


2.2 Verify the code cloned using z/OS projects perspective

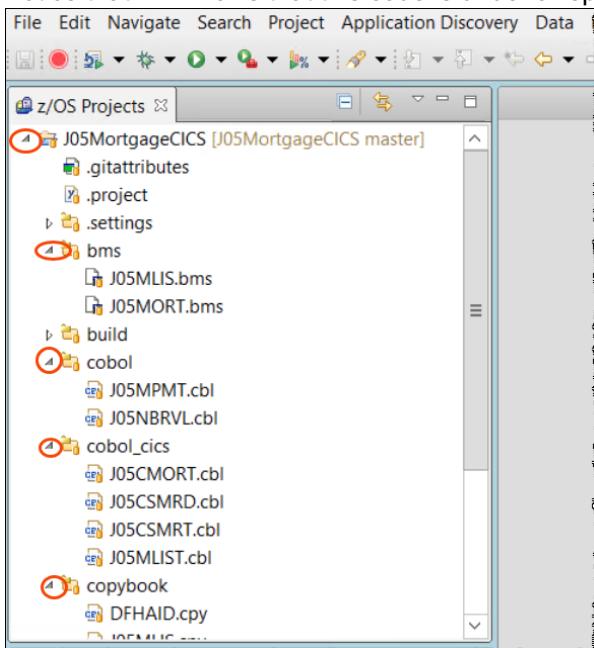
The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

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.



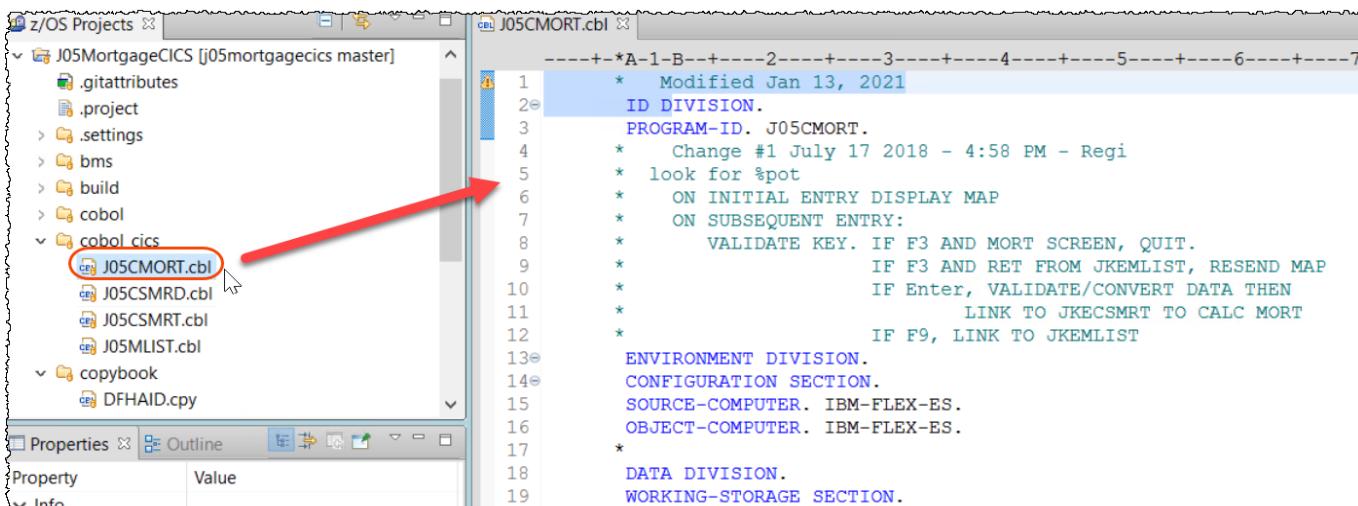
Section 3 –Modify the COBOL code using IDz.

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..

The screenshot shows the J05CMORT.cbl source code in the editor. The 'Find/Replace' dialog is open, with the 'Find' field containing '%pot'. The 'Find' button in the dialog is highlighted with a red arrow. Line 141 is circled with a red number 3.

```

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           *          MOVE 'INVALID KEY PRESSED.' TO MSGERRO
140
141           MOVE 'INVALID KEY PRESSED.' TO MSGERRO
142           SET SEND-DATAONLY TO TRUE
143           PERFORM A300-SEND-MAP
144
145           END-EVALUATE
146           EXEC CICS
147               RETURN TRANSID(EIBTRNID)
148               COMMAREA(W-COMMUNICATION-AREA)
149               LENGTH(W-COMAREA-LENGTH)
150           END-EXEC.

```

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..**

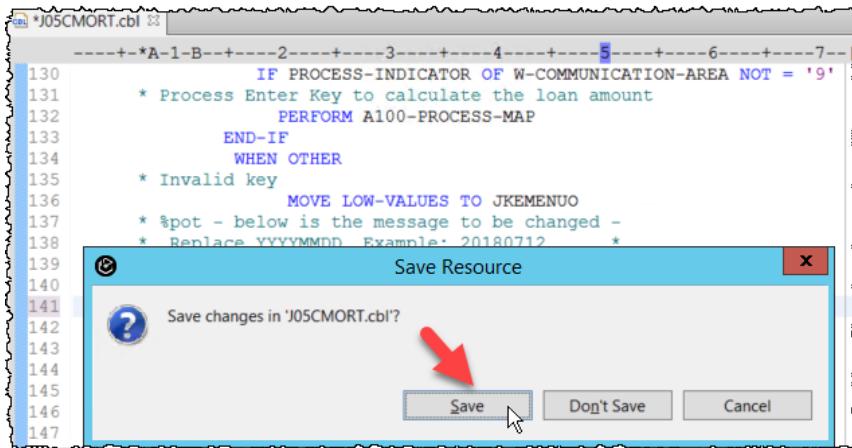
The screenshot shows the J05CMORT.cbl source code in the editor. Line 141 is circled with a red arrow, indicating it has been modified.

```

129
130           WHEN EIBAID = DFHENTER
131           IF PROCESS-INDICATOR OF W-COMMUNICATION-AREA NOT = '9'
132           * Process Enter Key to calculate the loan amount
133           PERFORM A100-PROCESS-MAP
134           END-IF
135           WHEN OTHER
136           * Invalid key
137           MOVE LOW-VALUES TO JKEMENUO
138           * %pot - below is the message to be changed -
139           * Replace YYYYMMDD Example: 20180712      *
140           *          MOVE '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
145           END-EVALUATE
146           EXEC CICS
147               RETURN TRANSID(EIBTRNID)
148               COMMAREA(W-COMMUNICATION-AREA)
149
150

```

3.1.4 ►| Use **Ctrl + Shift + F4** to close all code being edited. Click **Save** to save the modified code.

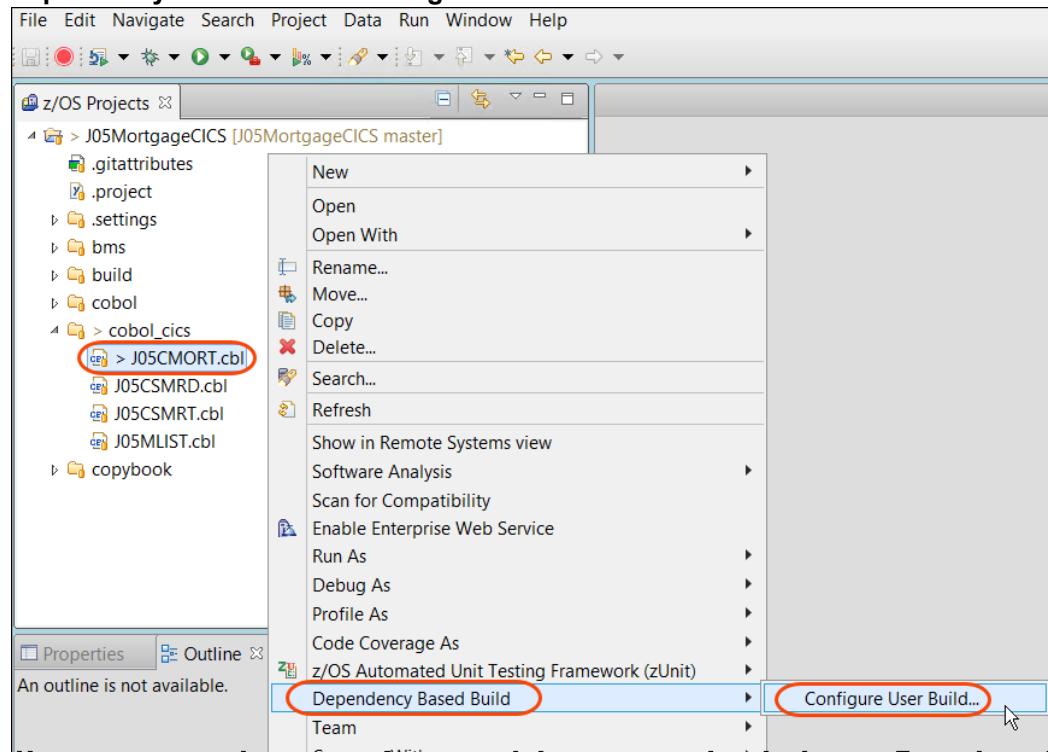


Section 4. Use IDz DBB User Build to compile/bind and perform personal tests.

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

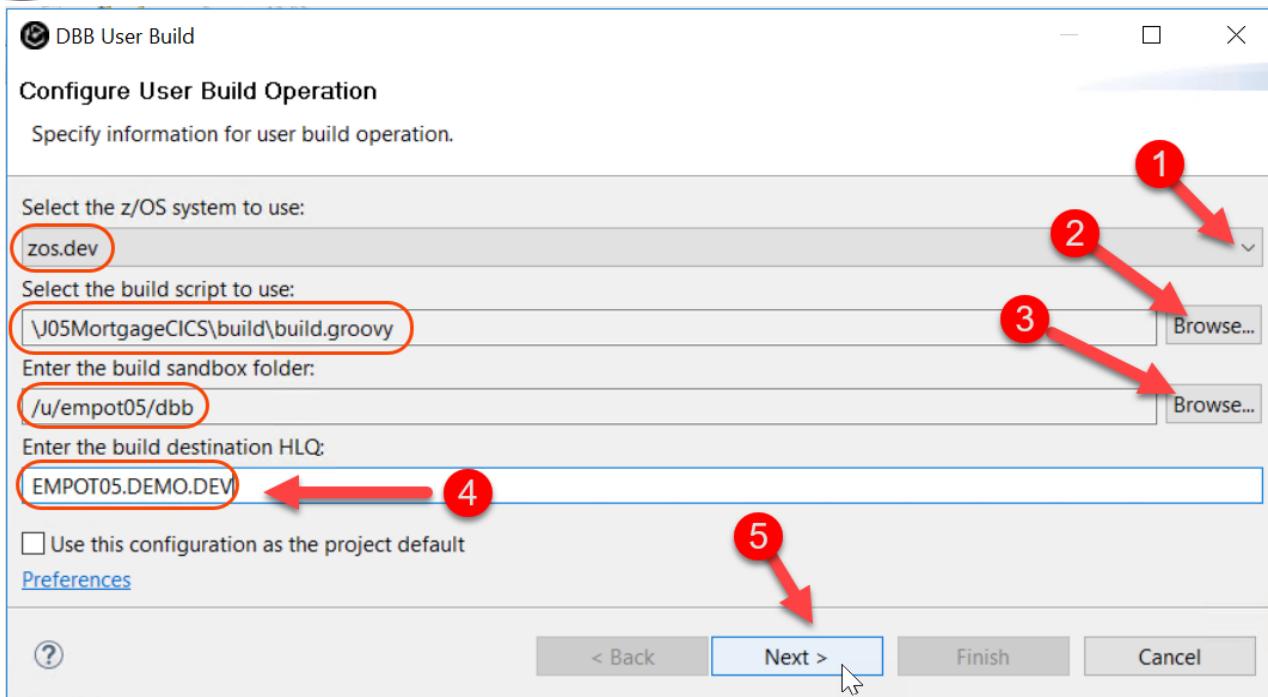
4.1.1 ►| Under **z/OS Projects** view right click **J05CMORT** and select **Dependency Based Build > Configure User Build...**



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

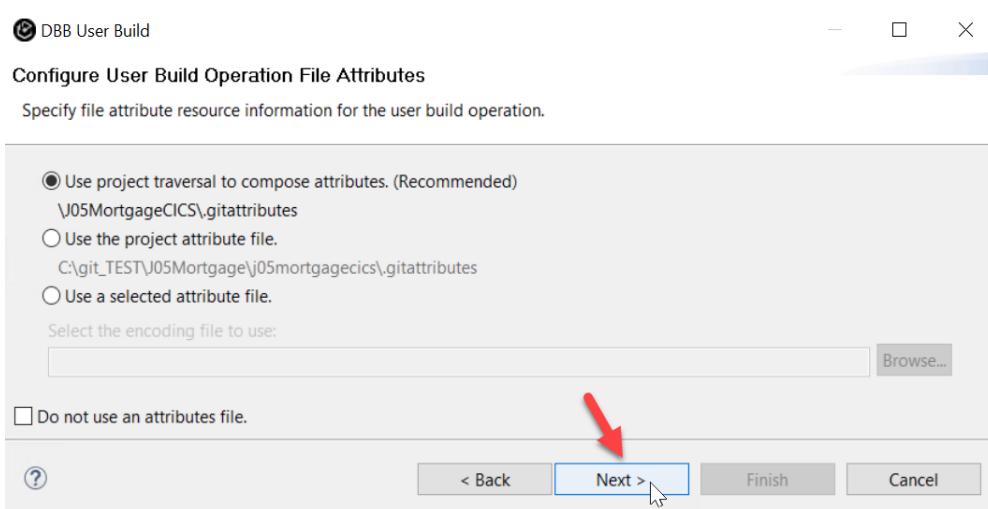
(Tip → Use the notepad and edit `c:\ADF_POT\LAB7\LAB7_Copy_Paste.txt` if want to copy/paste).

- 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**

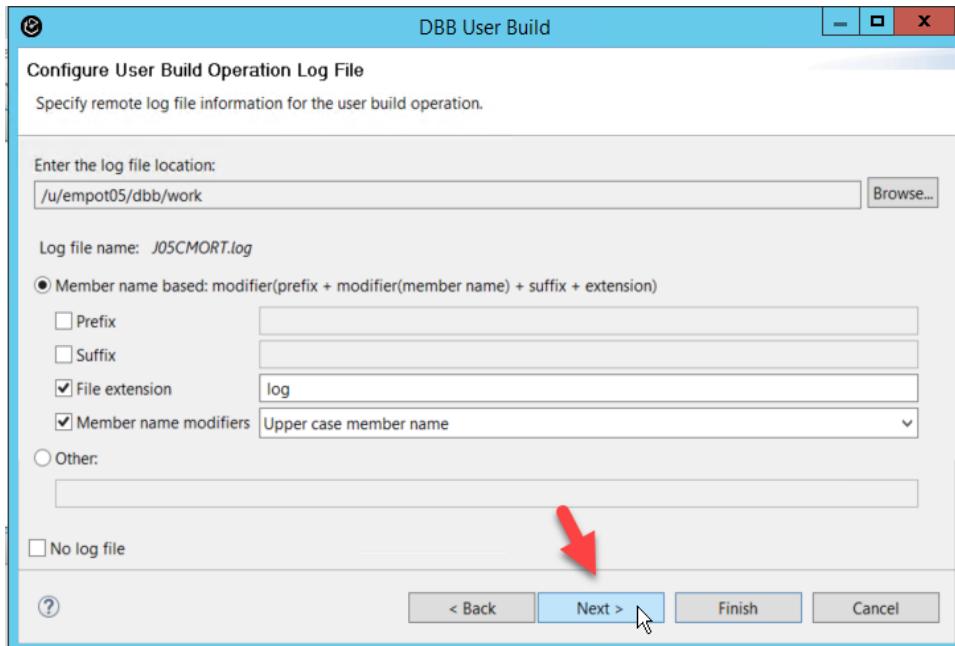


4.1.3 ► On *Configure User Build Operation File Attributes* click **Next**

The `.gitattributes` have the information to translate from *UTF-8* to *EBCIDIC* when moving the code to z/OS.

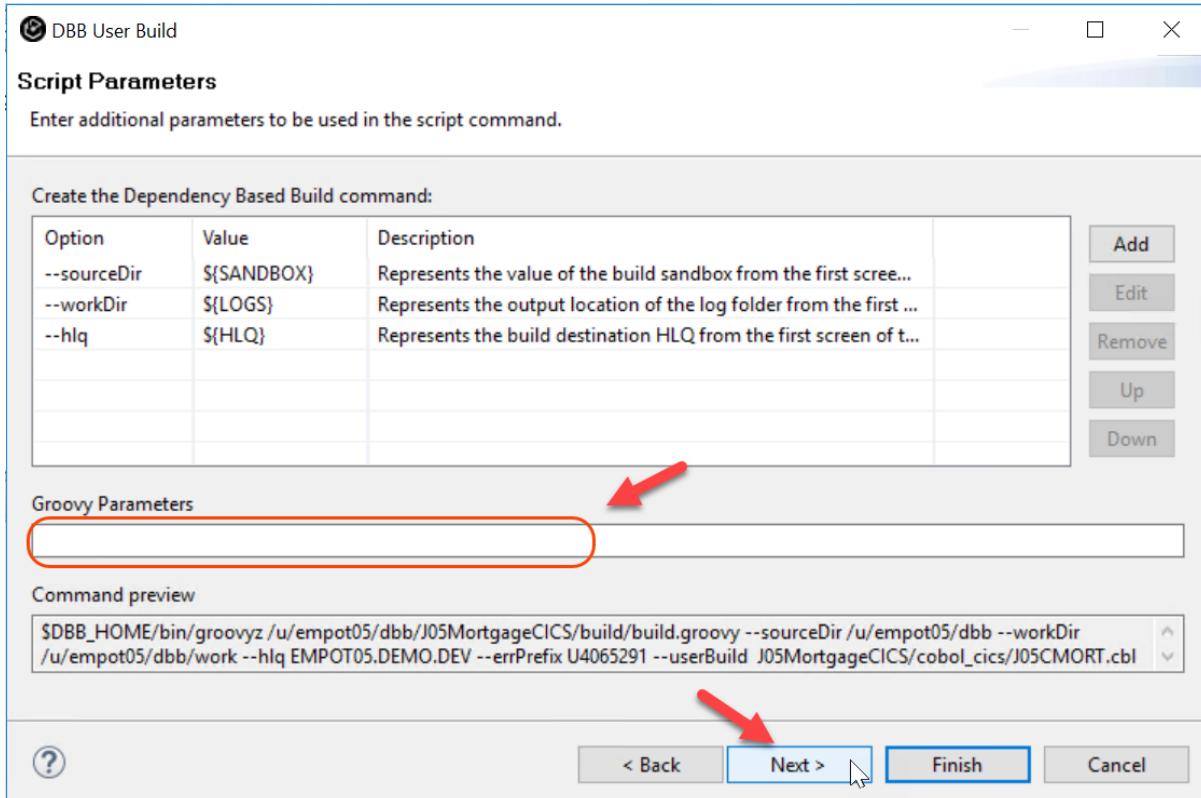


- 4.1.4 ► On *Configure User Build Operation Log File* click **Next**
 You will use the default values.



- 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.

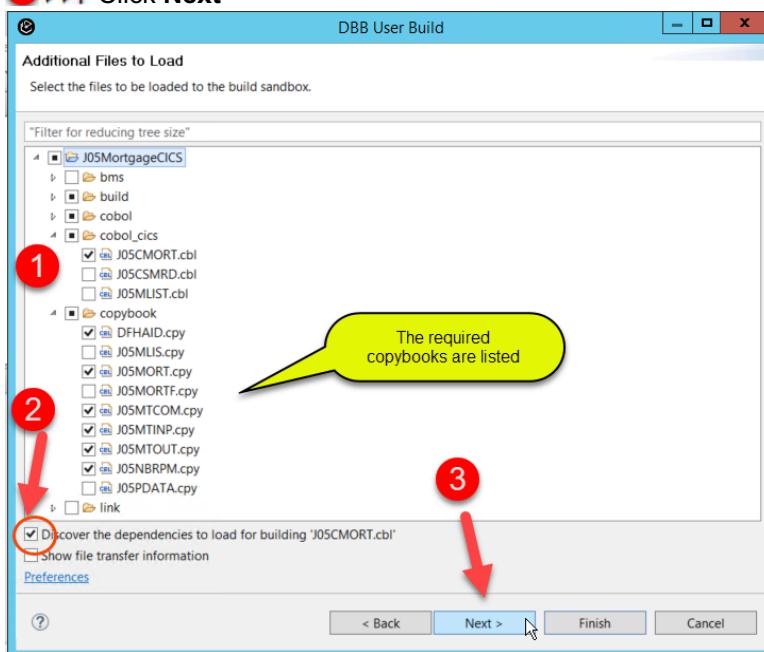


4.1.6 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

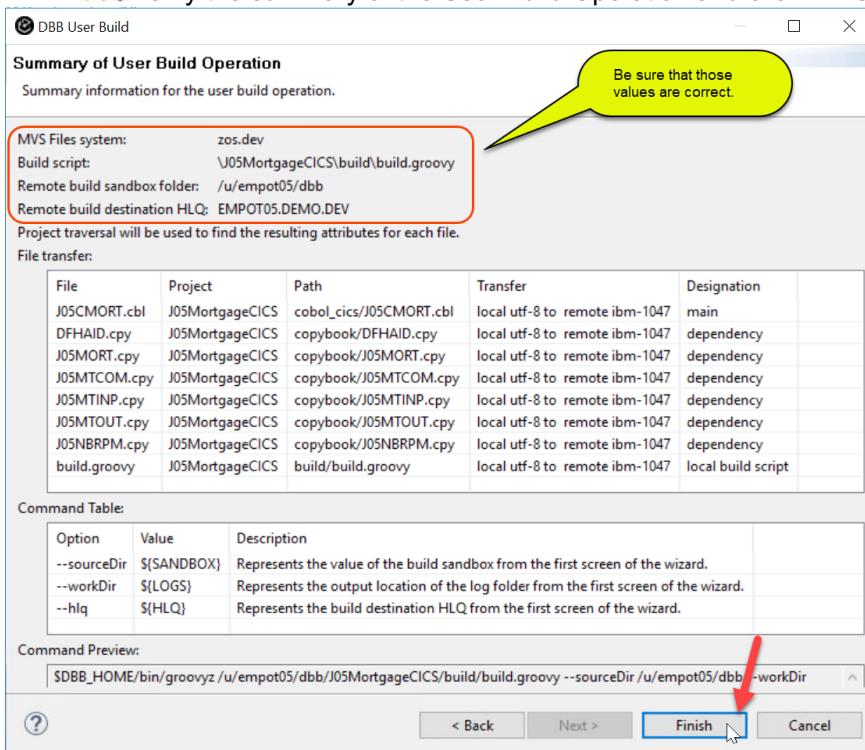
1 ►| Expand J05Mortgage CICS and the **cobol_cics** and **copybook** directories

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).

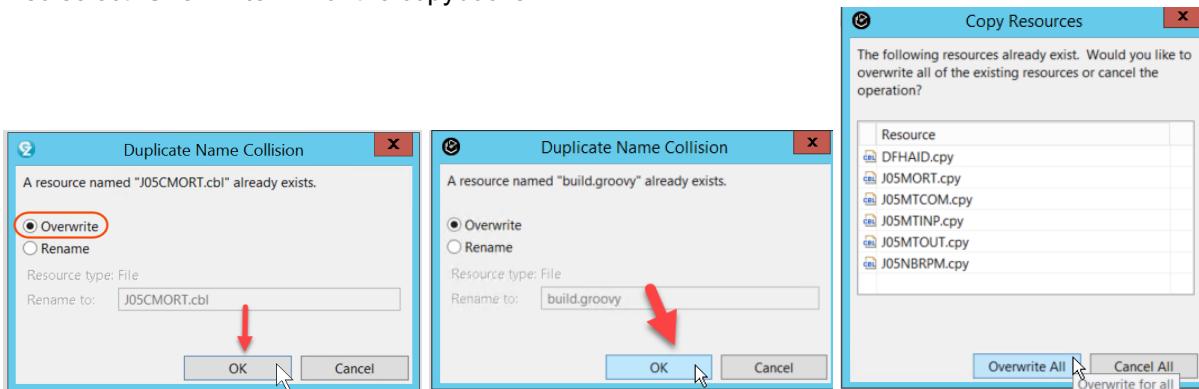
3 ►| Click Next



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

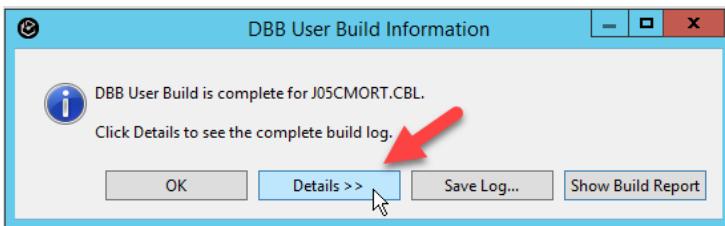


The DBB User build will be running and the messages will be shown at the Console view..

- 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:

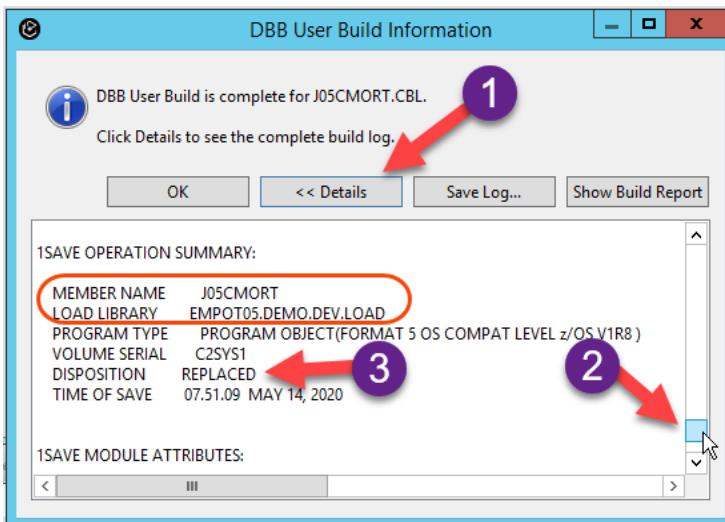
- Click **Details**



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.

The screenshot shows the DBB Console window with the following output:

```
** Build State : CLEAN
** Total files processed : 1
** Total build time : 44.516 seconds
/u/empot05/dbb>
** Build finished
/u/empot05/dbb>
```

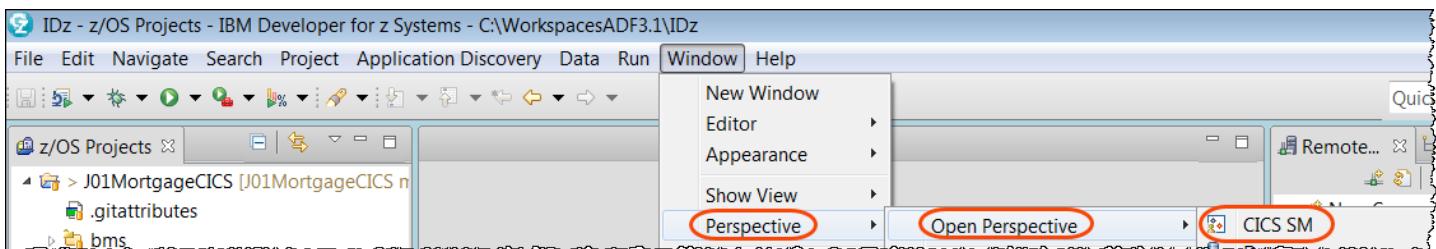
A red arrow points to the line "Build State : CLEAN", which is highlighted with a red oval. A red circle also highlights the "Console" tab in the top right corner of the window.

4.3 Issuing a CICS New copy

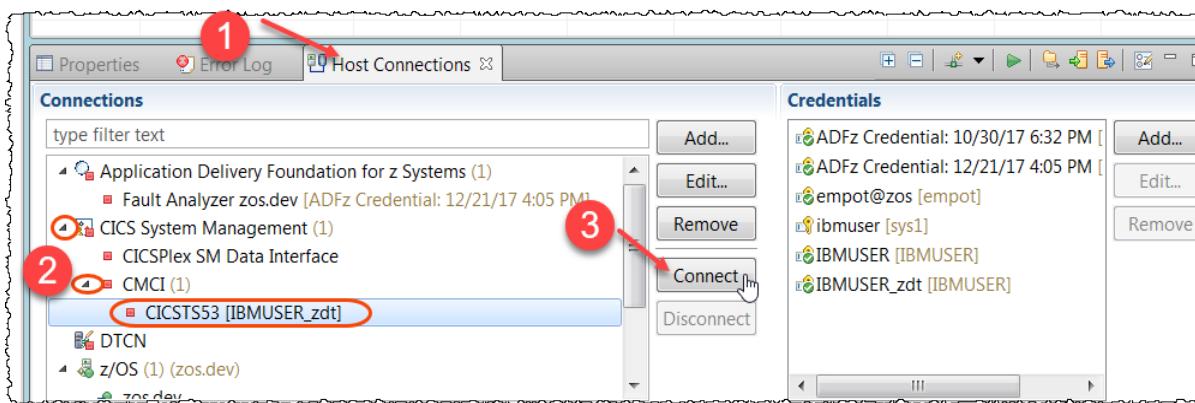
4.3.1 At this point you may test your new load module. But since this is a CICS system you will need to make a **CICS NEWCOPY**. Let's use IDz and CICS Explorer to do that.

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:

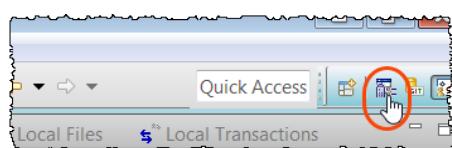
Region	Name	Status	Use Count	Concurrent Use C...	Language	Share Status	CEDF Status
CICSTS53	J05CMORT	ENABLED	3	0	COBOL	PRIVATE	CEDF
CICSTS53	J05CSMRD	ENABLED	0	0	COBOL	PRIVATE	CEDF
CICSTS53	J05CSMRT	ENABLED	1	0	COBOL	PRIVATE	CEDF
CICSTS53	J05MLIS	ENABLED	0	0	NOTAPPLIC	PRIVATE	NOTAPPLIC
CICSTS53	J05MLISD	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

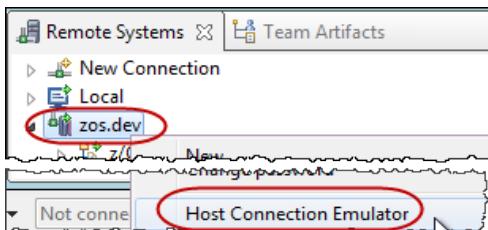
4.3.6 ► Select OK

4.3.7 Let's start the 3270 emulation again ..

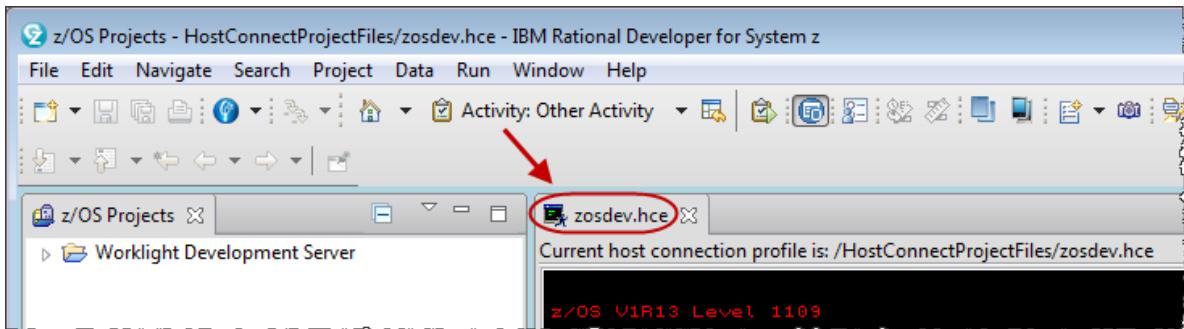
► Go to the **z/OS Projects** perspective



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



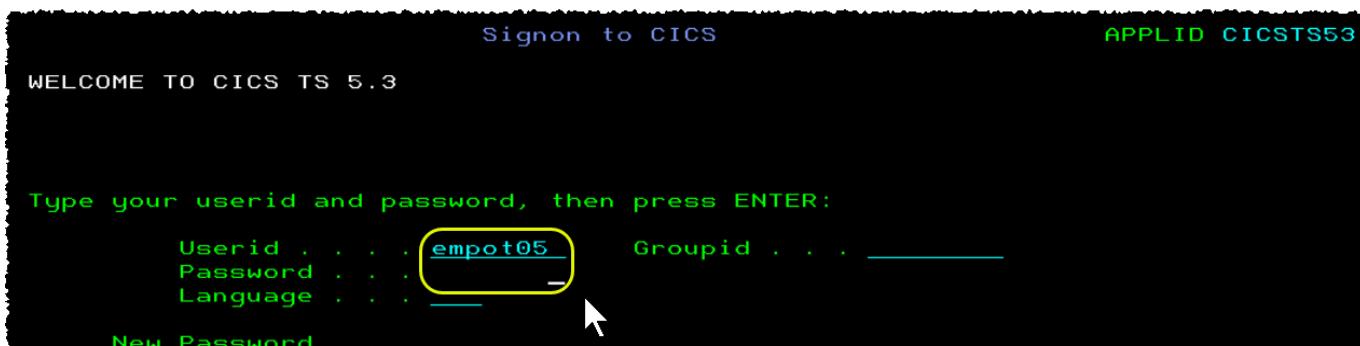
4.3.9 ► Since you will need more space, **double-click** on the **zos.dev.hce** title



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

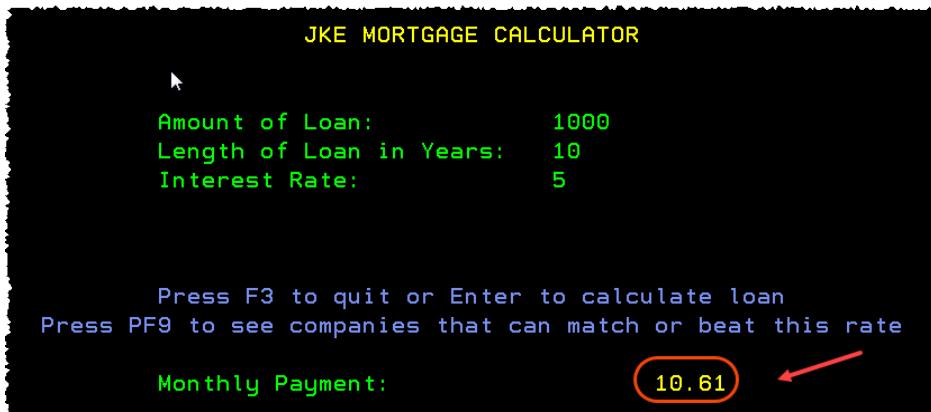


4.4 Running **j05p** CICS transaction

You should now be on the z/OS CICS region named *C/CSTS53*. 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 .



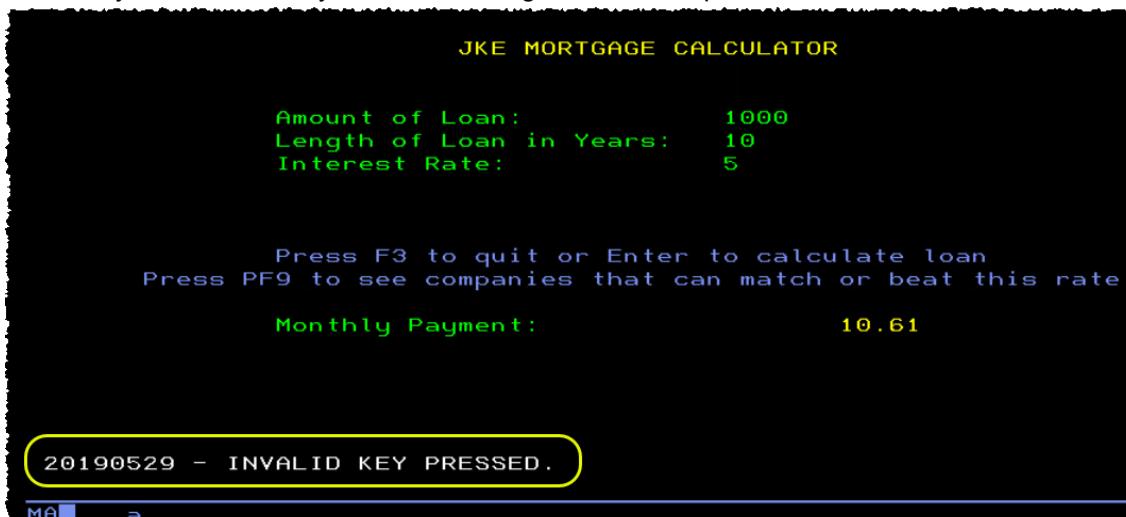
```
JKE MORTGAGE CALCULATOR

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
```

4.4.3 ► Press **F1** key and verify the message displayed
This was your mission. As you see the change has been implemented.



```
JKE MORTGAGE CALCULATOR

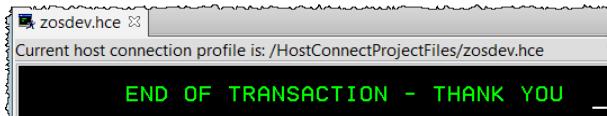
Amount of Loan: 1000
Length of Loan in Years: 10
Interest Rate: 5

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

20190529 - INVALID KEY PRESSED.
```

4.4.4 ► Press **F3** to end the CICS transaction.



```
zosdev.hce >
Current host connection profile is: /HostConnectProjectFiles/zosdev.hce
END OF TRANSACTION - THANK YOU
```

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

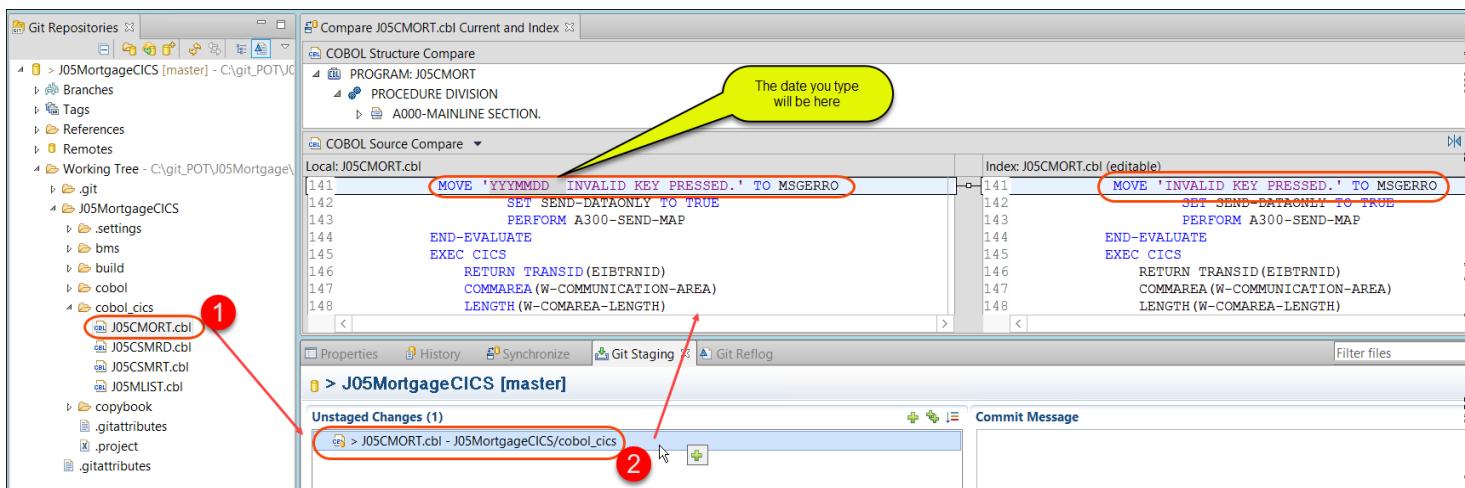
5.1.1 ► Switch to the **Git Perspective** selecting the icon on top and right



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.

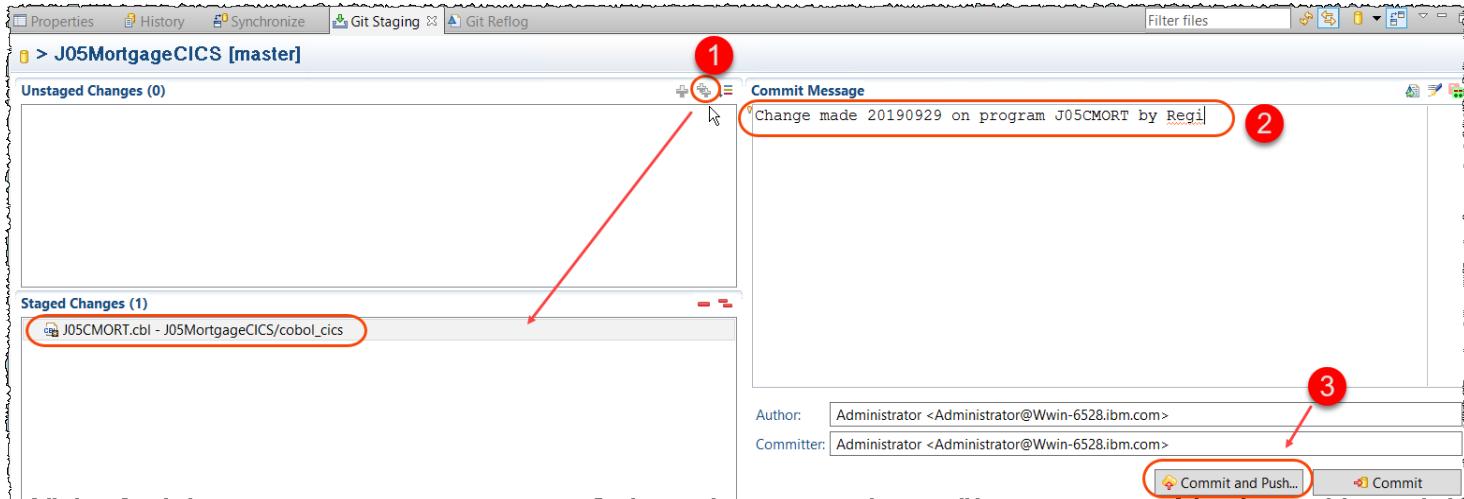


5.1.3 ► Use **Ctrl + Shift + F4** to close the editors.

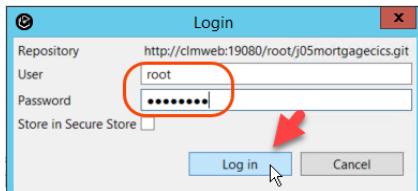
5.2 Push and Commit the changes to Git

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 ...**

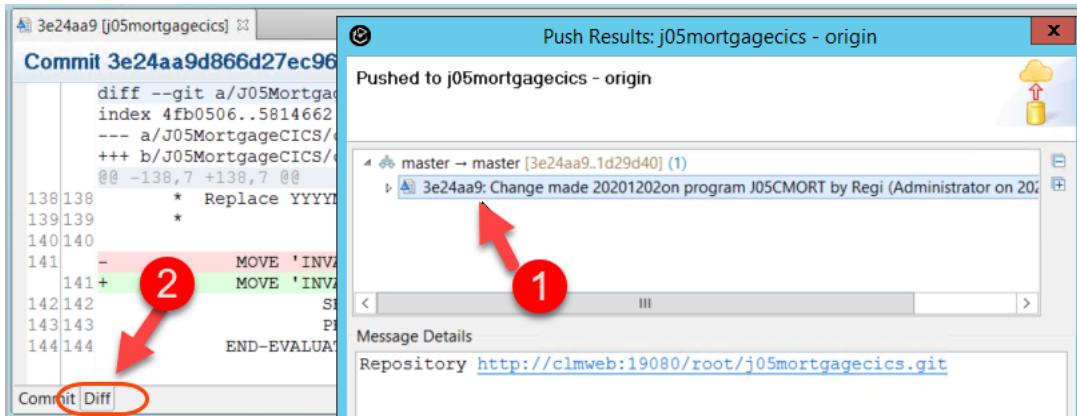


5.2.2 ► If it prompts for login credentials use **root** and password **zdtlinux** and click **Log in**



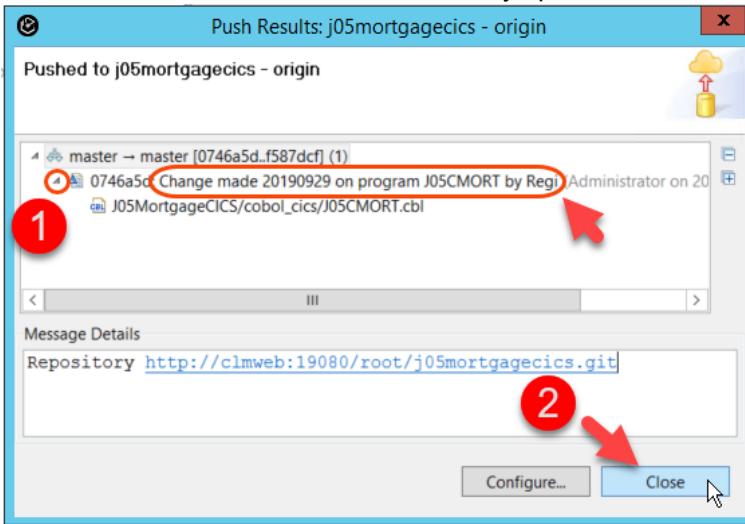
5.2.3 To see the changes you committed to Git:

- ① Double click on **Git hash string** and ② select **Diff tab**



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).

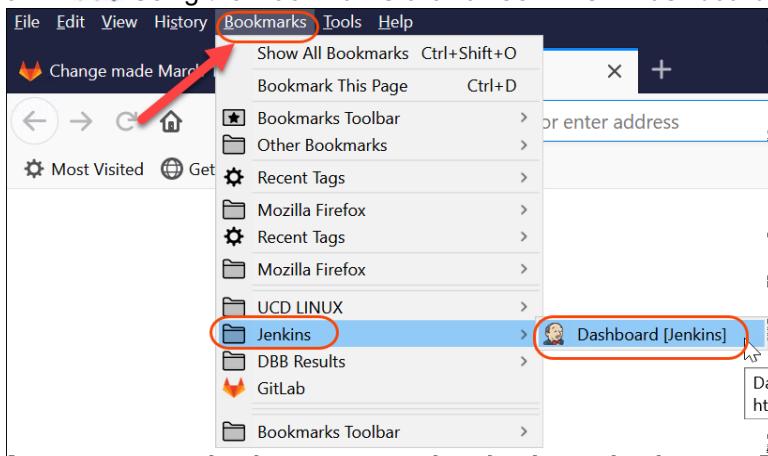
5.2.6 ► You also can click on **Change made yyyyymmdd on program J05CMORT by xxxxx** to see the changes

Section 6. Use Jenkins with Git plugin to build all the modified code committed to Git.

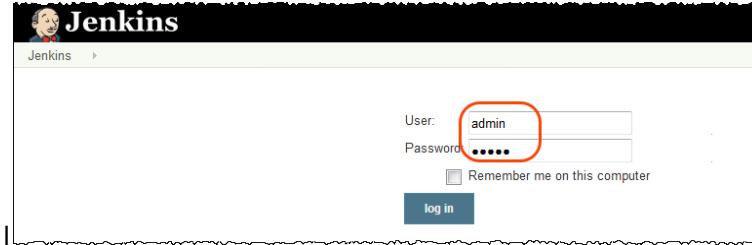
At this point the changed code is delivered into the Git repository that is on Linux. You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD. Note that this step makes a build of all code committed to Git, while on Section 4 only the selected COBOL program was built.

6.1 Logon to Jenkins using a Web Browser and start the z/OS agent

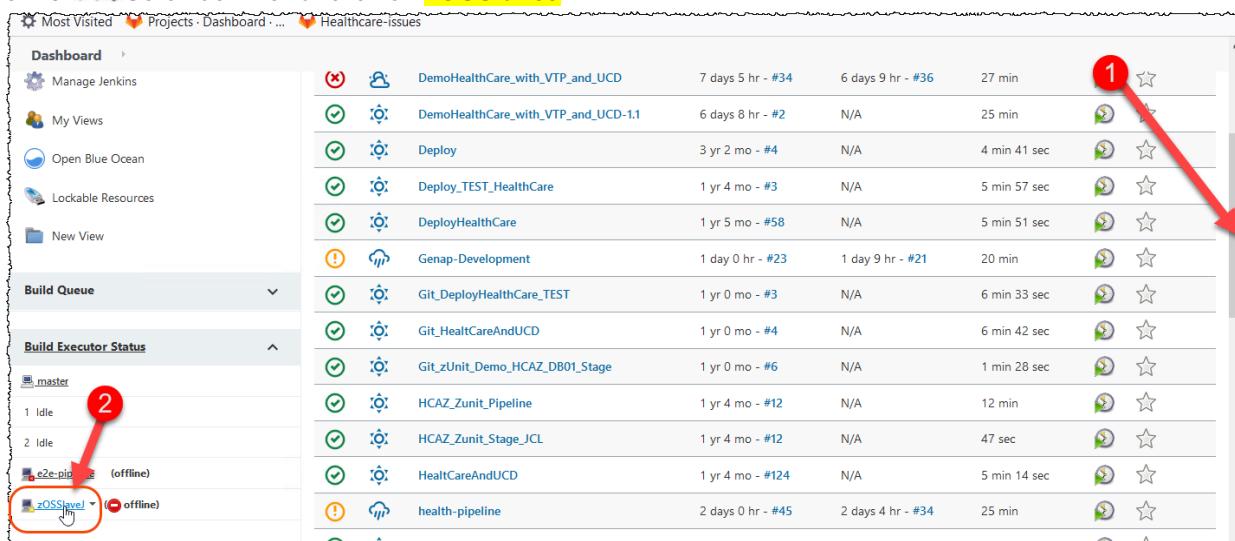
6.1.1 Using the Bookmarks click on Jenkins > Dashboard



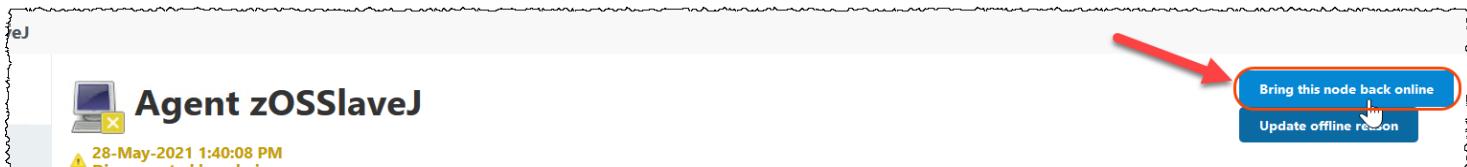
6.1.4 If required type the user admin and password admin (lowercase) and click log in.



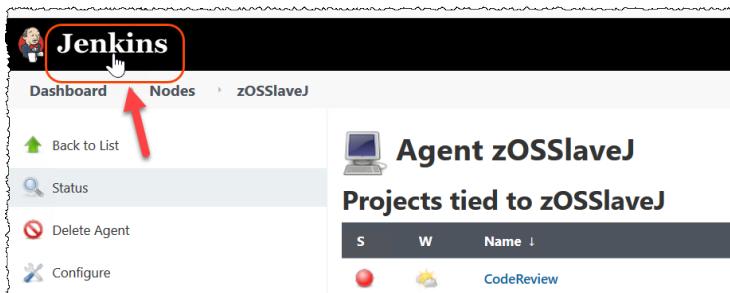
6.1.5 Scroll down and click on zOSSlaveJ



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.



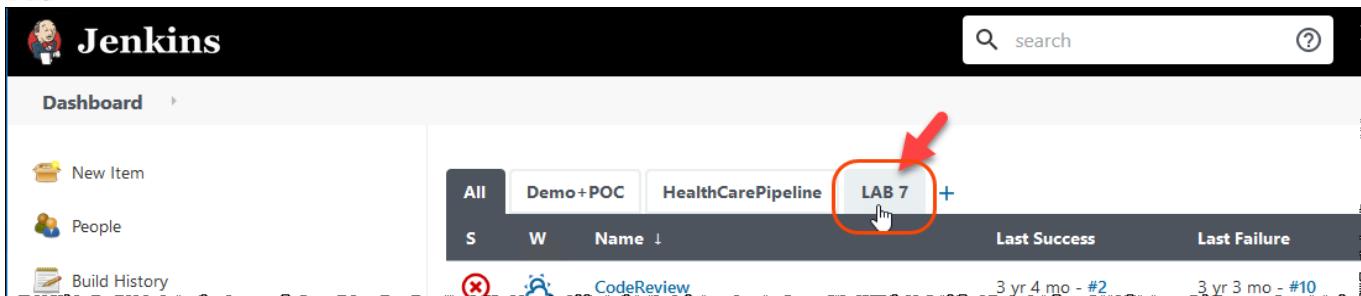
6.1.6 ► Click on Jenkins on top left:



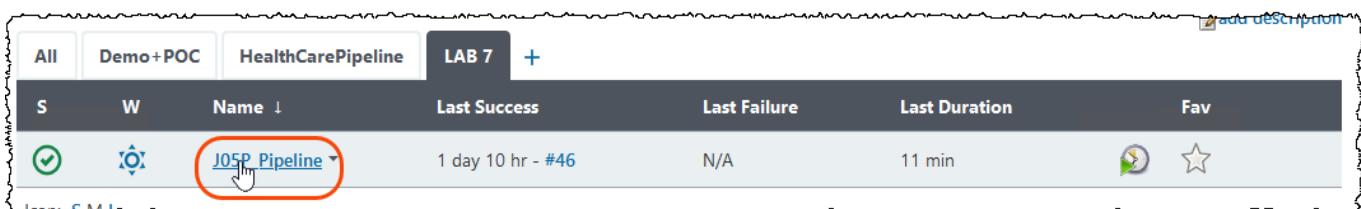
6.2 Starting the Jenkins Pipeline

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.

Jenkins > J05P_Pipeline >

Pipeline J05P_Pipeline
Pipeline to build and Deploy J05P - DevOps PoT

Back to Dashboard | Status | Changes | **Build Now** | Delete Pipeline | Configure | Full Stage View | Rename

Recent Changes

Stage View

- 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.

Jenkins > J05P_Pipeline >

Pipeline J05P_Pipeline
Pipeline to build and Deploy J05P - DevOps PoT

Back to Dashboard | Status | Changes | **Build Now** | Delete Pipeline | Configure | Full Stage View | Rename | Pipeline Syntax

1

Build History

#	Date
#7	May 29, 2019 2:15 PM
#6	Aug 29, 2018 3:00 AM

RSS for all RSS for failures

Stage View

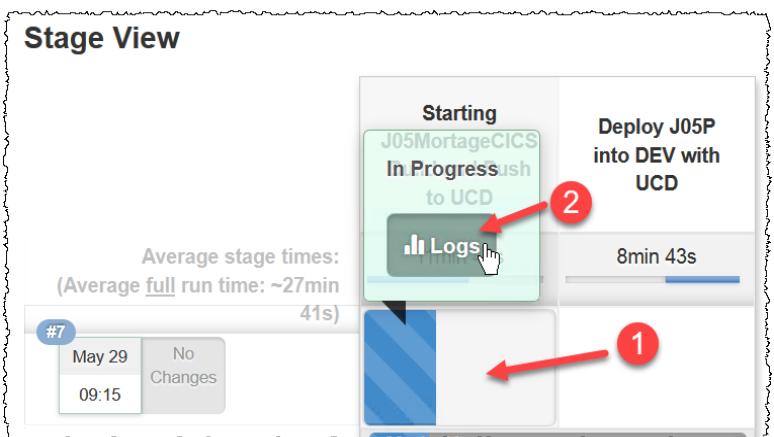
Starting J05MortageCICS Build and Push to UCD	Deploy J05P into DEV with UCD
Average stage times: (Average full run time: ~27min 41s)	10min 29s 8min 43s
41s)	
#7	May 29 09:15 No Changes
	25min 37s

2

6.3 Checking the results

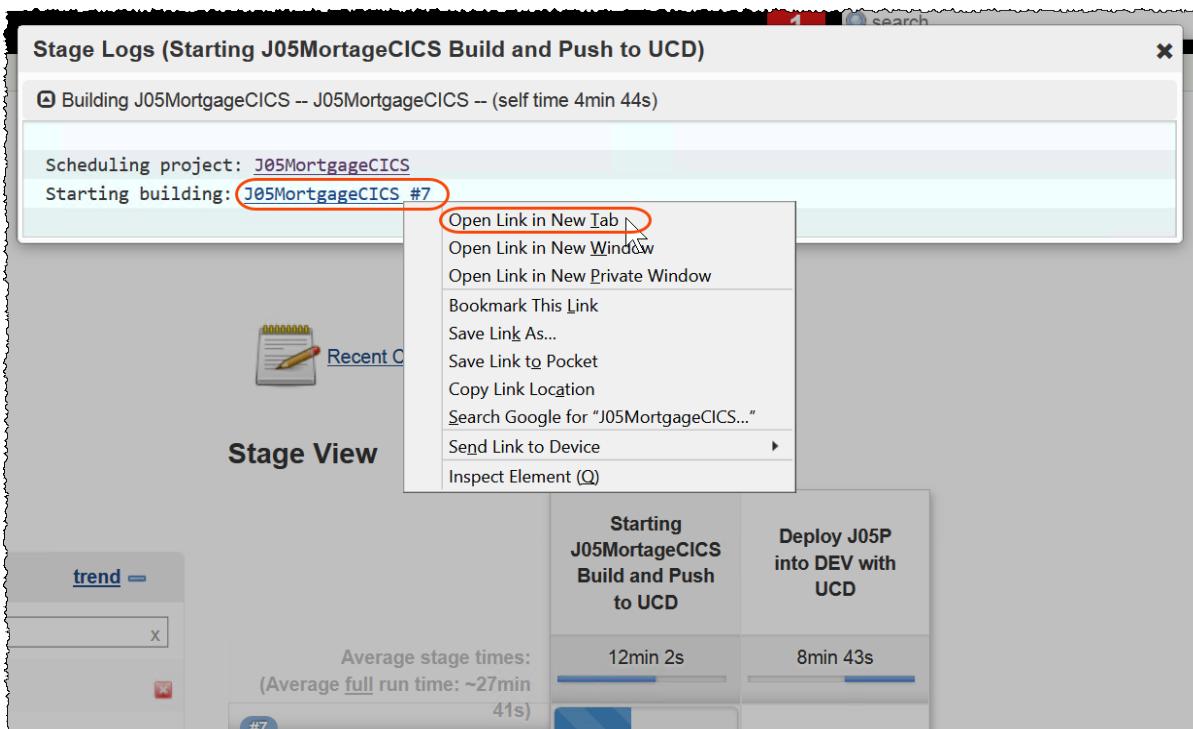
6.3.1 Since this z/OS is running in a small processor in the cloud, this activity may take 10 or more minutes, but you don't need to wait to be completed to check the results.
You can follow what is going on as described below ..

- Move the cursor and click on the **blue area** ① and then click on **Logs** ②.



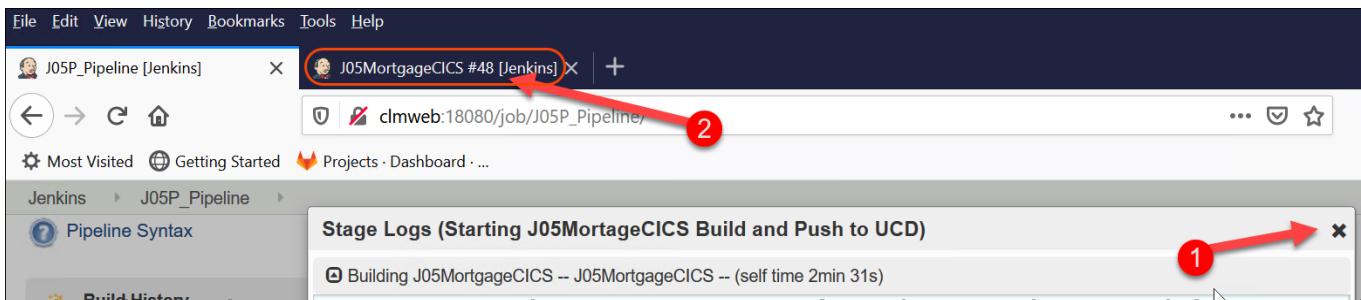
6.3.2 The dialog *Stage Logs* will open.

- Right click on **J05MortgageCICS #nn** and select **Open Link in New Tab**



6.3.3 This opens a new Jenkins tab as shown below.

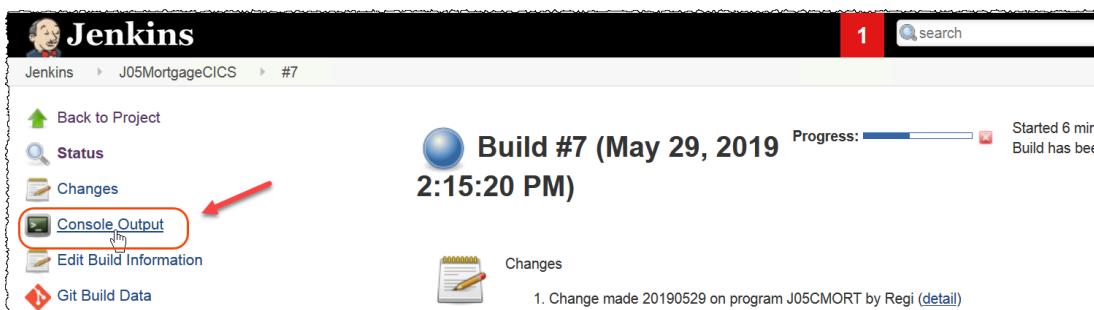
▶ Close Stage Logs dialog and click on the newly opened tab (J05MortgageCICS#nn [Jenkins])



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



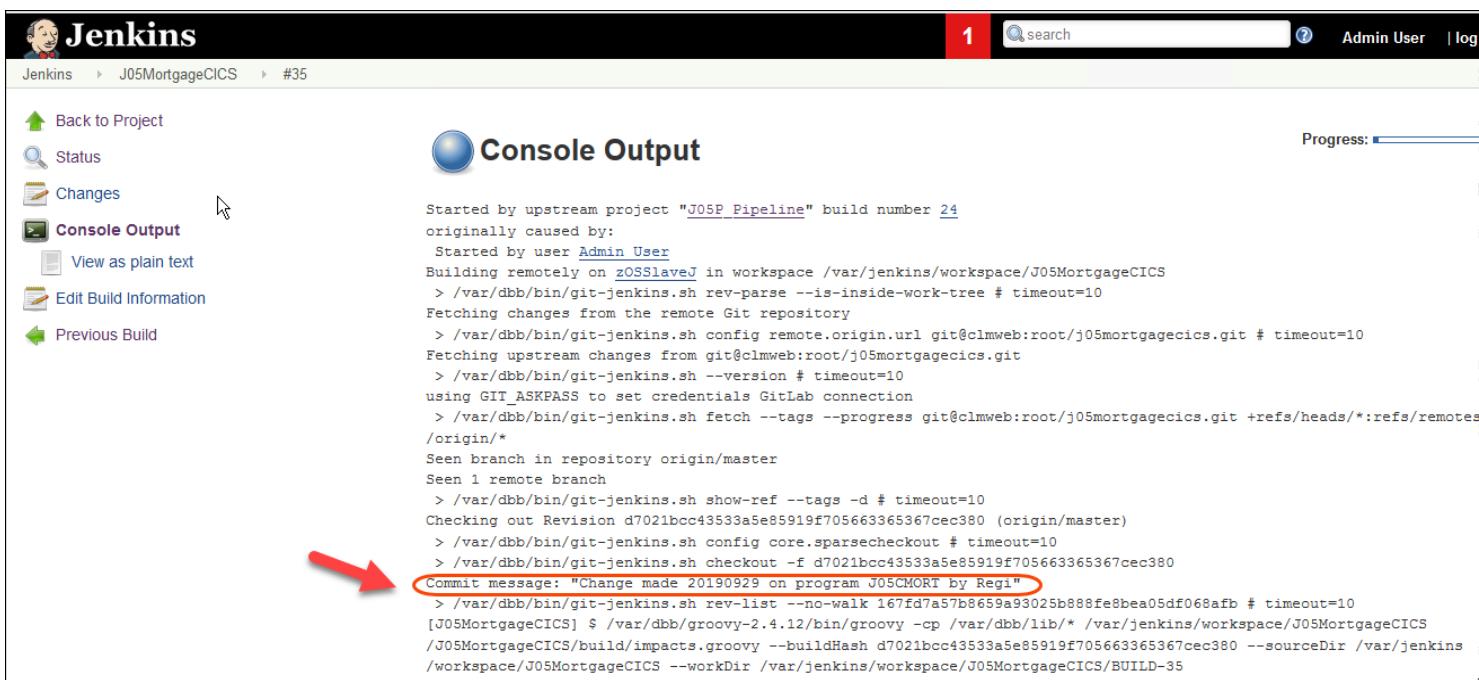
6.3.5 The Console Output log of first pipeline shows the various Groovy scripts being executed.

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 a incorrect value. Each zOS instance must have a unique value and you need to generate 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**.
- 6 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) 
> /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
Finished: SUCCESS
```




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) .

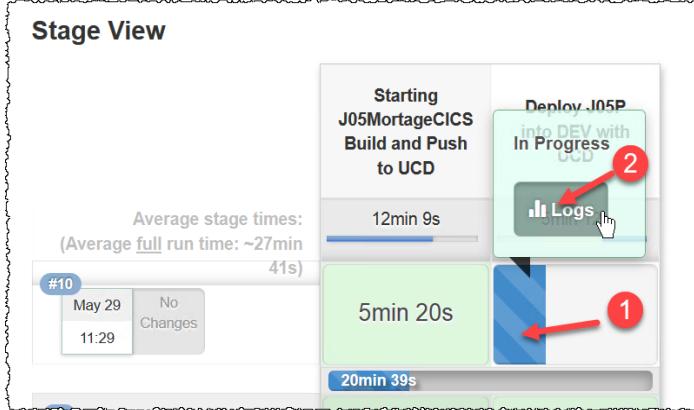
7.1 Checking the deploy results (second stage)

- 7.1.1 ► 1 Click on the previous browser tab **J05P_Pipeline[Jenkins]**

- 2 Click the Refresh icon 

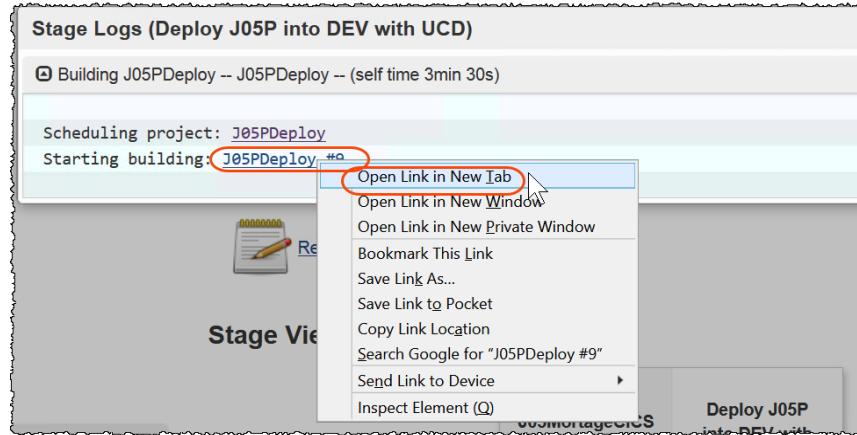


- 7.1.2 ► 1 Click as show in the **blue area** of the **Deploy stage** 1 and then click on **Logs** 2.

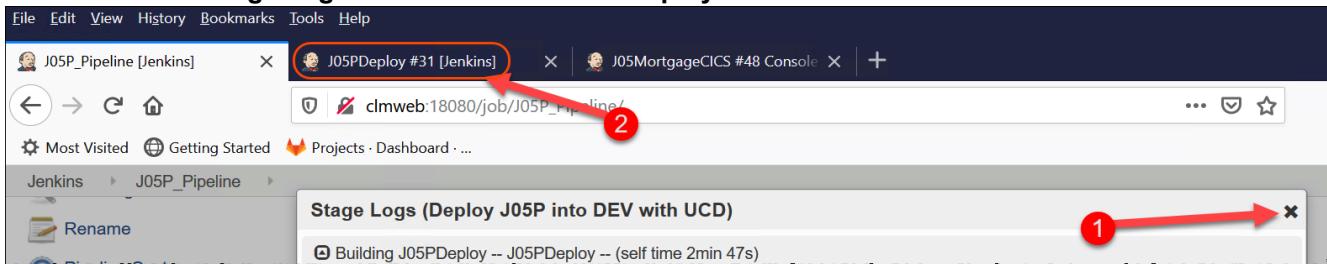


7.1.3 The dialog Stage Logs will open.

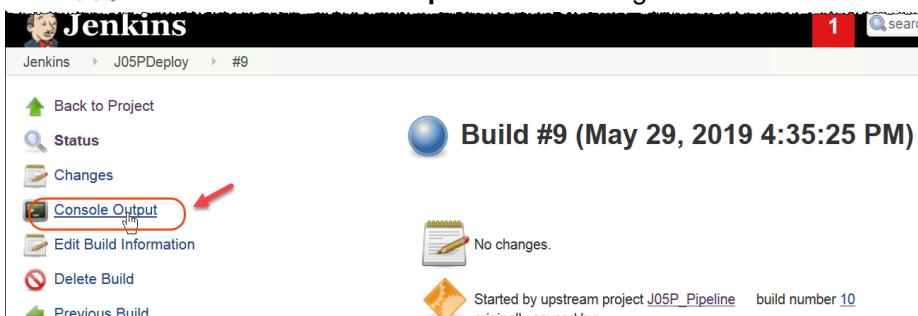
► Right click on **J05PDeploy #nn** and select **Open Link in New Tab**



7.1.4 ► Close Stage Logs and click on that J05PDeploy #nn

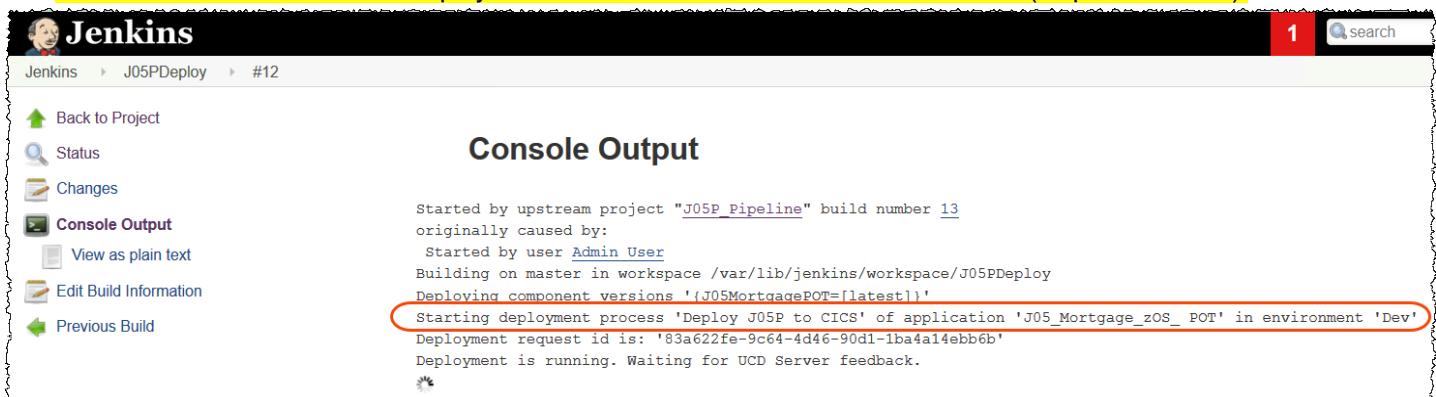


7.1.5 ► Click on the Console Output to view the logs.



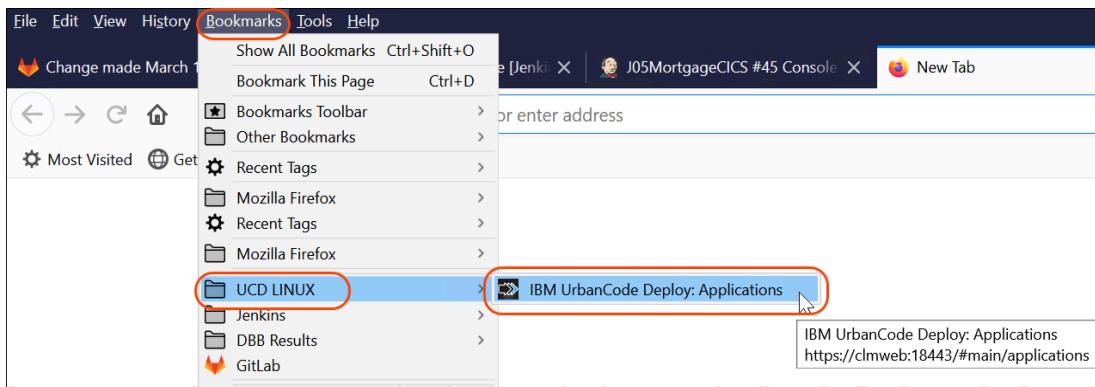
7.1.6 The Console Output log of second pipeline shows that the program that you delivered will be deployed to CICS Dev environment using UCD..

► 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)

Application: J05_Mortgage_zOS_POT

Environments History Configuration Components Blueprints Snapshots Processes Calendar Changes

Process	Environment	Snapshot	Scheduled For	By	Status	Actions
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.

Execution

Success ▾

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	1 / 1	5:25:24 PM	0:00:53	Success
Total Execution	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-a7ff-310c09585fc1
35 WE_ACTIVITY_ID=19982f24-88a5-4b63-alab-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 in orange. Below the tabs, the breadcrumb navigation shows 'Home / Components'. Under 'Components', there are two sub-options: 'Components' and 'Templates'.

7.2.10 ► Scroll down and click on J05MortgagePOT

This screenshot shows the details for the 'J05MortgagePOT' component. It includes fields for 'Name' (J05MortgagePOT), 'Last Modified' (20201202-200731), 'Used in DevOps Pot and Jenkins Lab' (true), and 'Timestamp' (7/17/2018, 4:3 PM).

7.2.11 ► Click on Versions and the version created by your Jenkins run (it is a timestamp value)

This screenshot shows the 'Versions' tab of the component details 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 Importing
Filter	Statuses	Any			
20201202-200731		Incremental	admin	12/2/2020, 3:07 PM	false

7.2.12 ► Expand EMPOT05.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.

This screenshot shows the 'Artifacts' section of the component details page. A red circle labeled '1' highlights the 'Expand All' link. A red circle labeled '2' highlights the expanded entry for 'EMPOT05.DEMO.DEV.LOAD [PDS,ADD]'. The entry shows 'J05CMORT' as the artifact type, 'LOAD' as the deploy type, and 'J05MortgageCICS/cobol_cics /J05CMORT.cbl' as the inputs. Properties include '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.

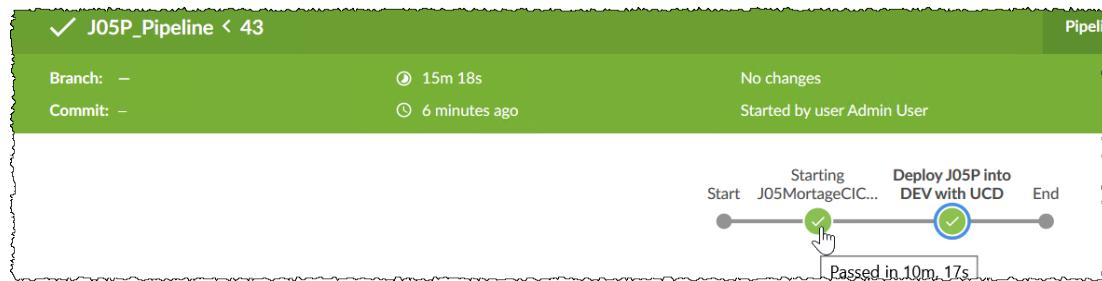
7.3 Using Jenkins Blue Ocean Plugin

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"

This 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 list.

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
File Copy Paste Send Recv Display Color Map Record Stop Play Record Support Index
z/OS V2R2 PUT1606 / RSU1607 IP Address = 10.148.150.8
VTAM Terminal = SC0TCP03
Application Developer System
      // 0000000 SSSSS
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"
I cicsts53
24/01/11
Connected to remote server/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 . . . 
Language . . .
New Password . . .

```

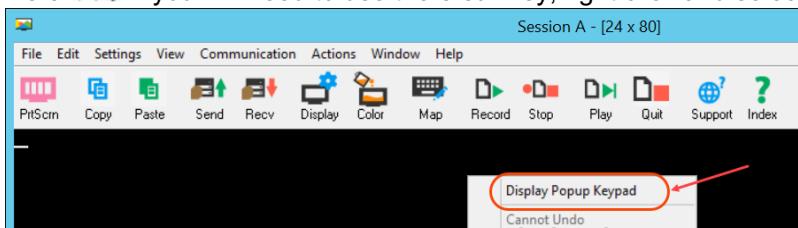
7.3.4 The sign-on message is displayed

```

DFHCE3549 Sign-on is complete (Language ENU).
MF a

```

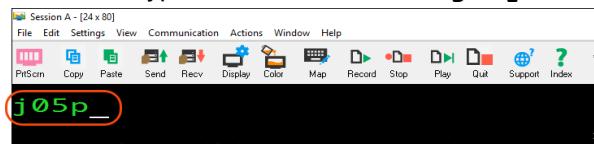
7.3.5 ► If you will need to use the **clear** key, right click and select **Display Popup Keypad**



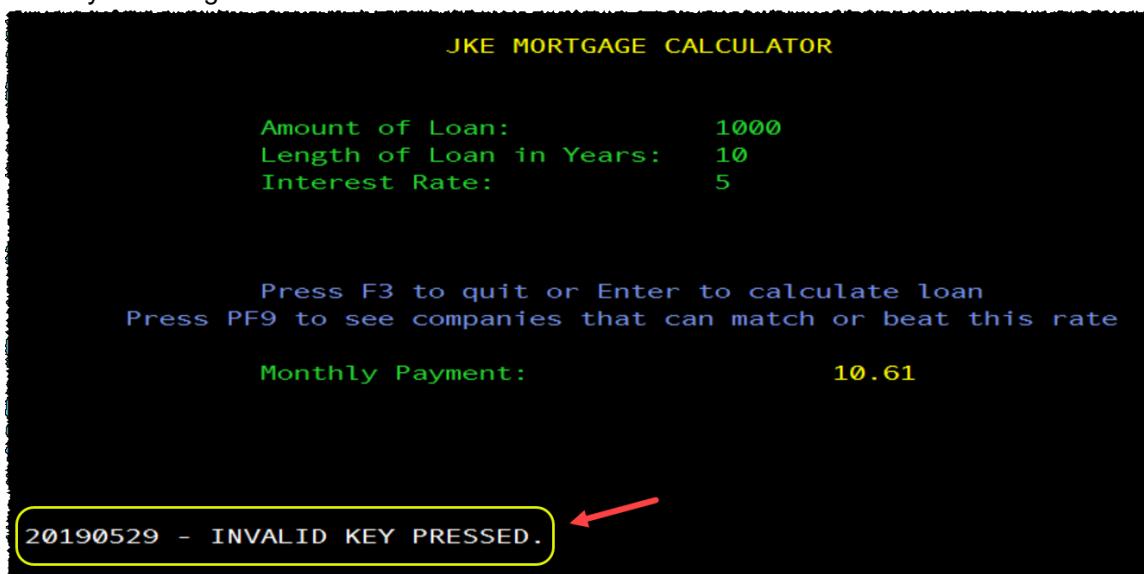
► and then select **Clear**



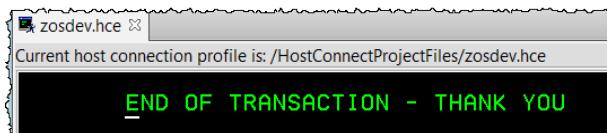
7.3.6 ► Type the CICS transaction **j05p** and press the **Enter** key.



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.



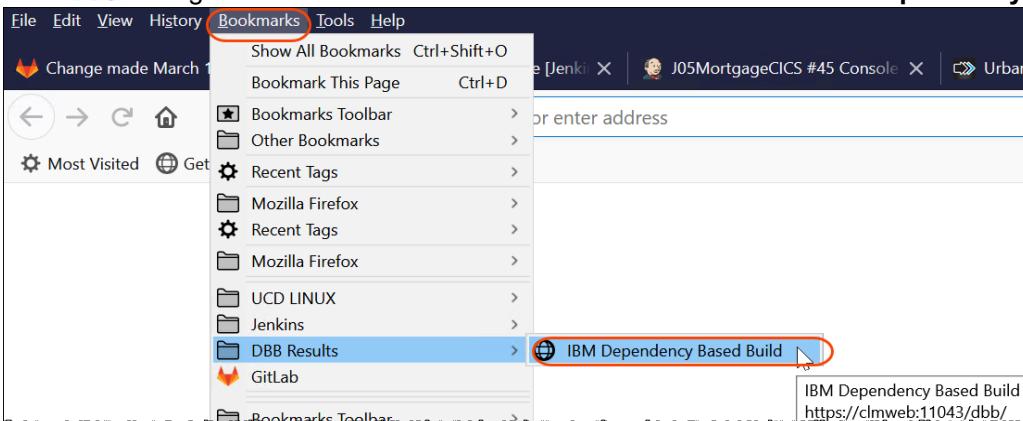
7.3.9 ► Close the terminal emulator.

Section 8.(Optional) Understanding DBB Build Reports

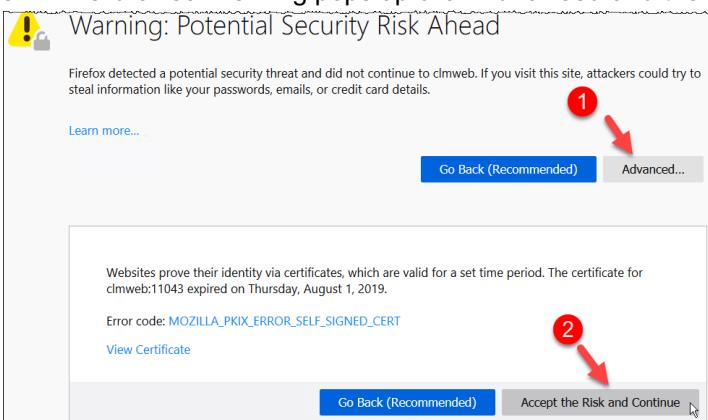
DBB has an application server (our is running on Linux) that capture the various build events. Here you will take a quick look of those reports.

8.1 Login to the DBB server using the browser

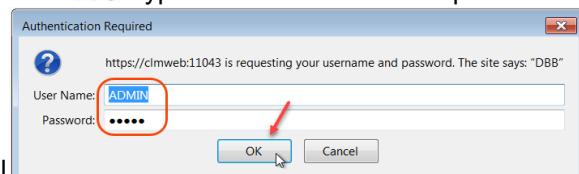
8.1.1 Using the Browser Bookmarks click on DBB Results > IBM Dependency Based Build



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 it 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.

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)

File Edit View History Bookmarks Tools Help

IBM Dependency Based Build × +

IBM Collections **Build Results**

https://clmweb:11043/dbb/

DBB Build Results

- ▶ J01MortgageCICS
- ▶ HealthCareApp
- ▶ J03MortgageCICS
- ▶ J02MortgageCICS
- ▶ J04MortgageCICS
- ▶ **J05MortgageCICS** 2
 - [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](#)** 3
- ▶ J06MortgageCICS

8.3.2 ► This opens the *DBB Build Results..* Click the Build Report **view** to get details..

The screenshot shows a web browser window titled "IBM Dependency Based Build". The URL is https://clmweb:11043/dbb/. The page displays a table of build results. One row, specifically the "Build Report" row, contains a "view" link which is highlighted with a red oval and a red arrow pointing to it.

Field	Value
id	899
group	J05MortgageCICS
label	build.20190529.043258.032
Build Report	view
buildReportData	view
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 IEWLINK	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 IEWLINK	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 5 – (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**.

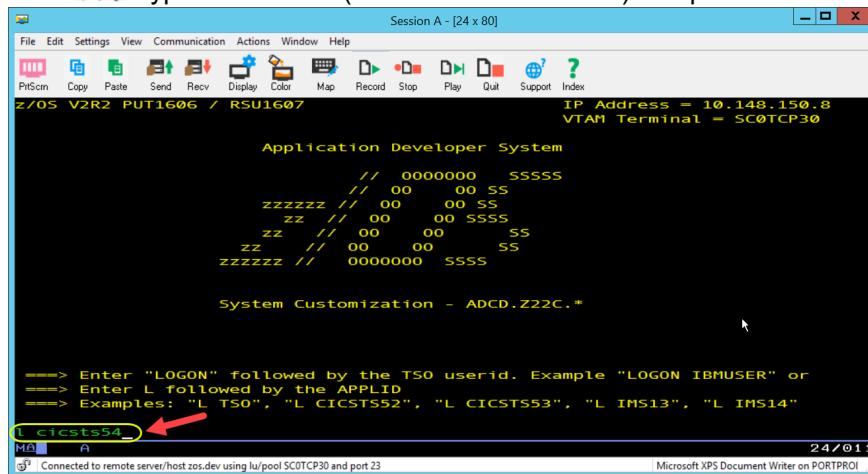
1.1 Emulate a 3270 terminal and Connect to CICS version 5.4

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

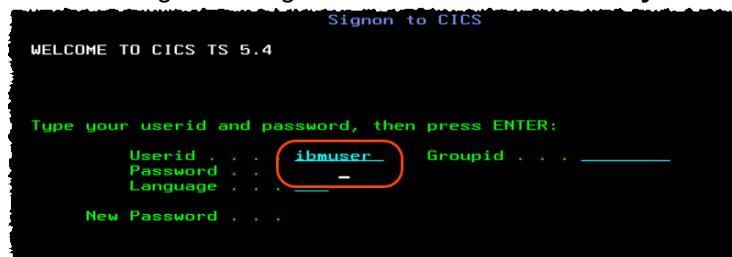


This opens the host emulator.

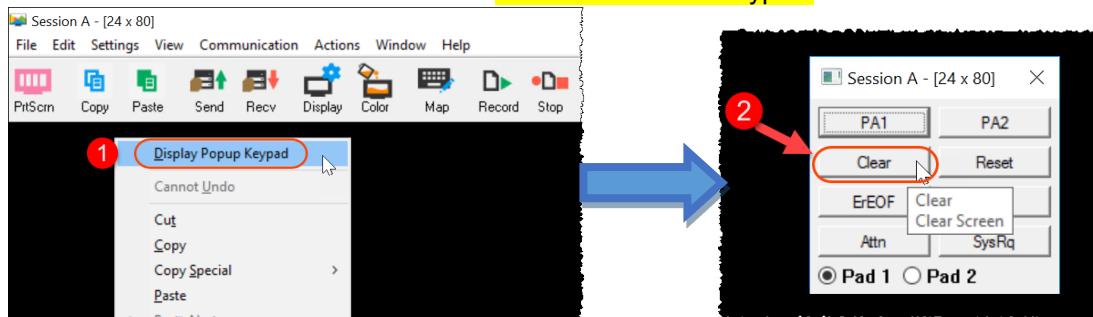
- 1.1.2 Type **I cicsts54.** (where I is L lower case) and press **Enter key**.



- 1.1.3 Sign on using **ibmuser** as the userid and **sys1** as the password.

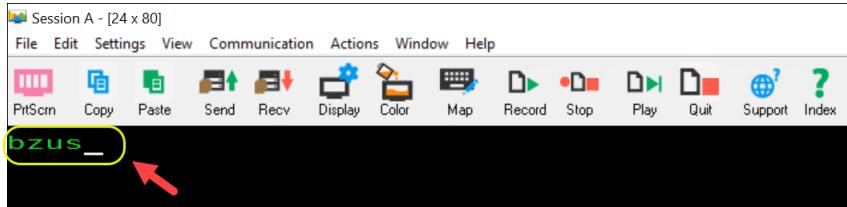


- 1.1.4 To clear the screen, **right click** on the dark space and select **Display Popup Keypad** and click the **Clear** button. **DO NOT close the keypad.**

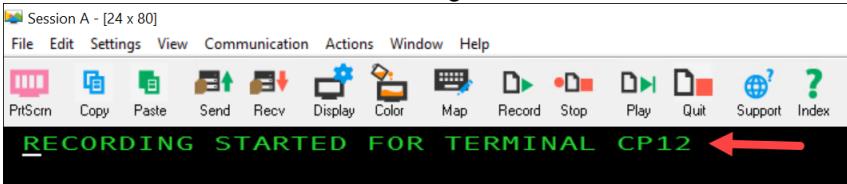


1.2 Using VTP to record the Health Care Application transaction sequence

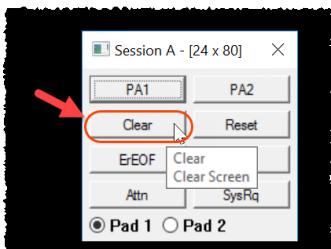
1.2.1 ► Click on the black area to type the transaction **bzus** and press **Enter key**



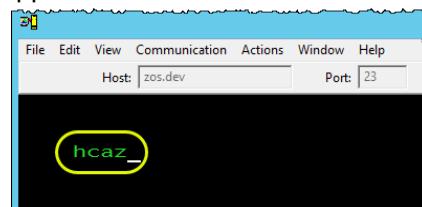
1.2.2 This invokes the Z VTP recording in the CICS terminal..



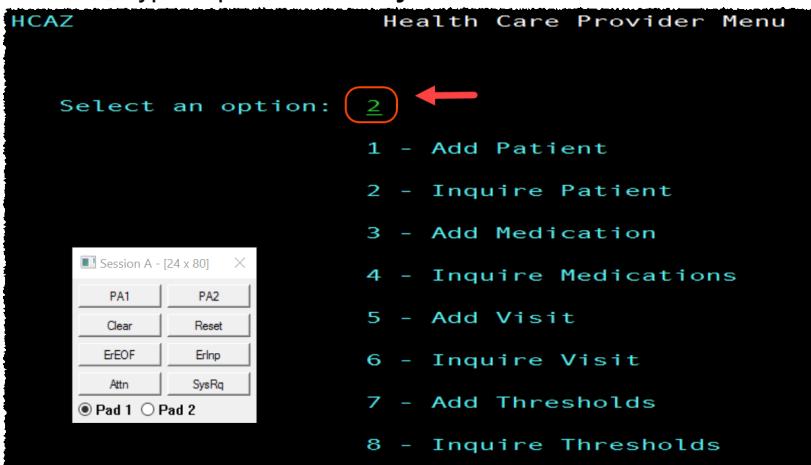
1.2.3 ► Clear the screen again clicking the **Clear** button.



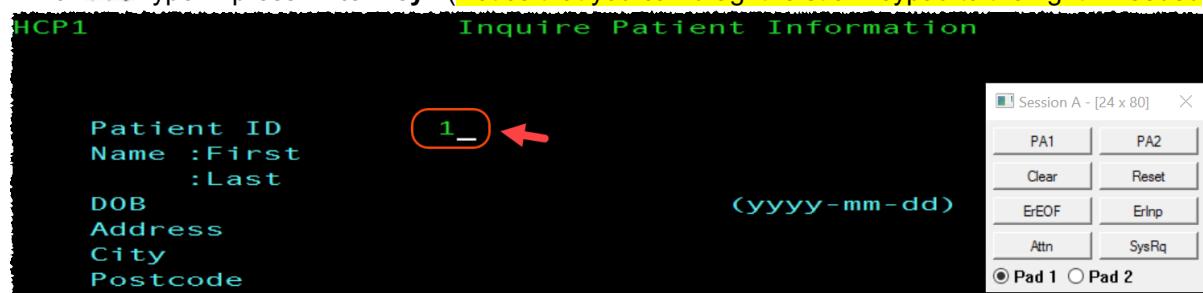
1.2.4 ► On CICS main terminal type **hcaz** and press **Enter key** to invoke the CICS transaction application



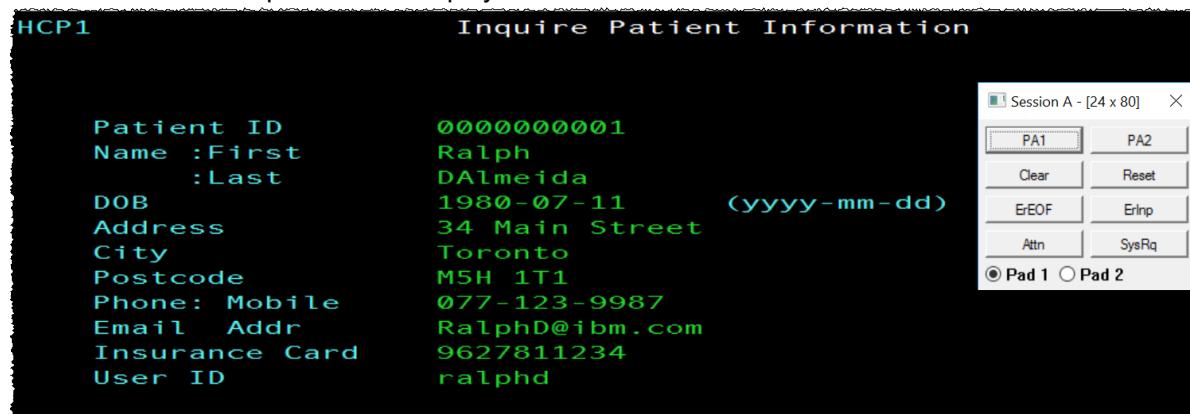
1.2.5 ► Type 2 press **Enter key** .



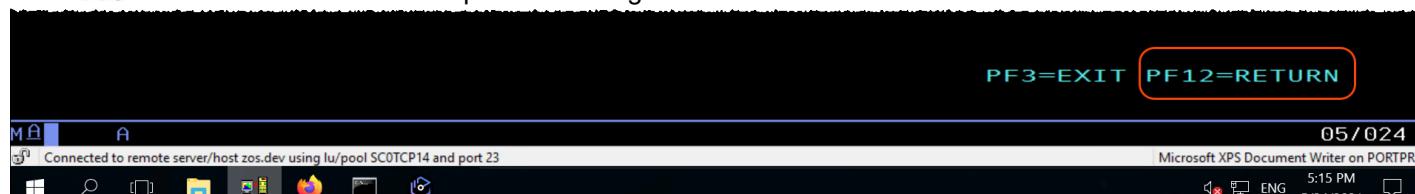
1.2.6 ► Type 1 press **Enter key** . (Notice that you can drag the stick keypad to the right if needed)



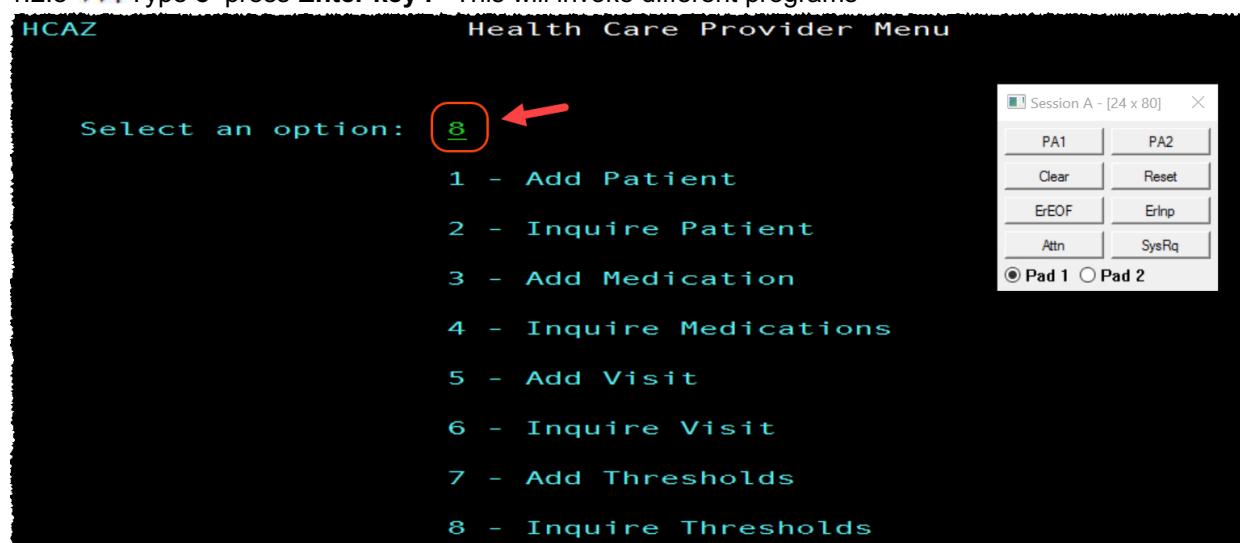
You should have the patient data displayed.



1.2.7  Press **PF12** to return to the previous dialog



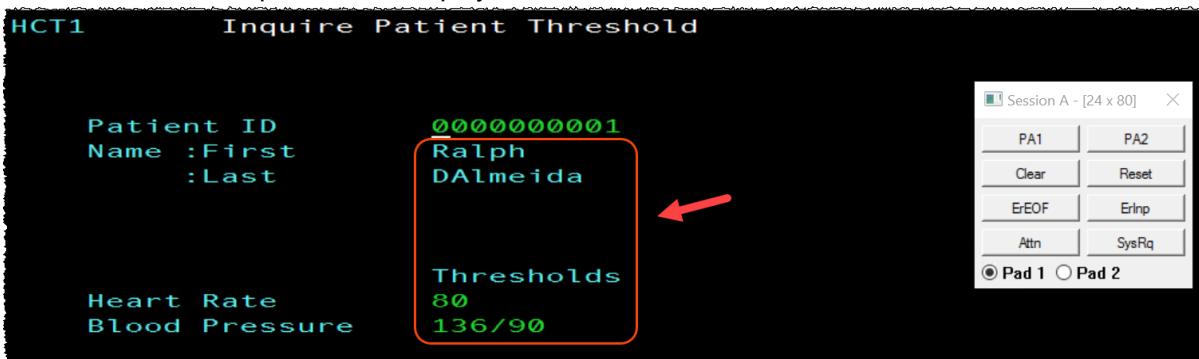
1.2.8 ► Type 8 press **Enter key** . This will invoke different programs



1.2.9 ➡ Type 1 press **Enter key**. (Notice that you can move the stick keypad to the right if needed)

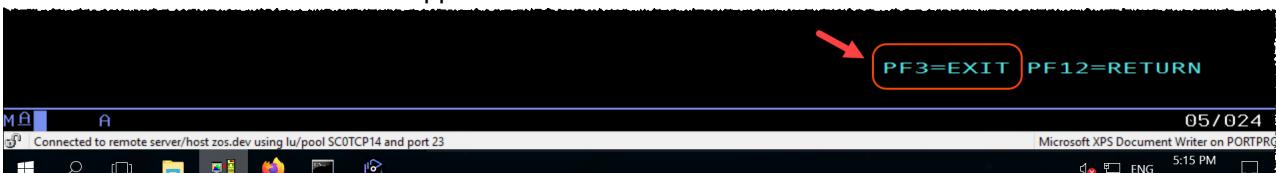


You should have the patient data displayed.

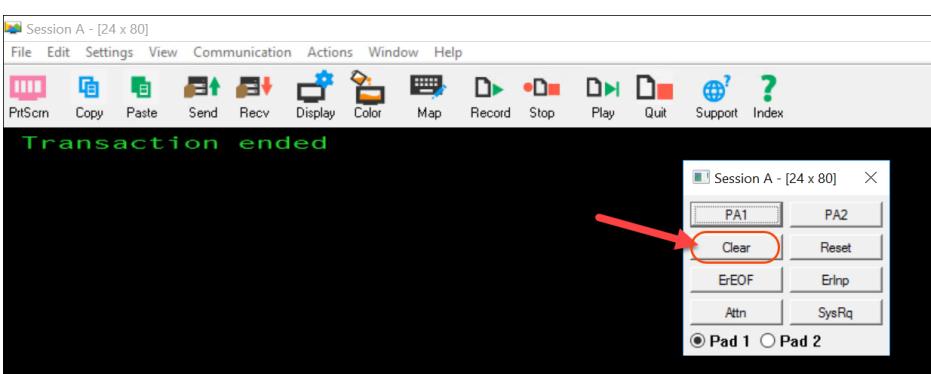


IMPORTANT: Remember the *first name* of this patient (Ralph). Later you will introduce a bug.. Instead of **Ralph** the COBOL program will display “**BAD NAME**” as first name.

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..



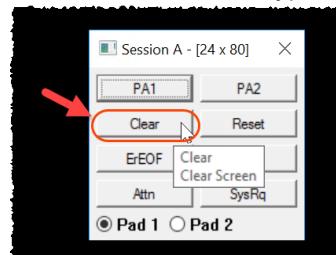
1.2.12 ➡ Type the transaction **bzue** and press **Enter key**



1.2.13 ► The following message appears in the screen.

```
RECORDING STOPPED FOR TERMINAL CP14
```

1.2.14 ► Use the Keypad to click on the **Clear** button and return to the main CICS terminal.



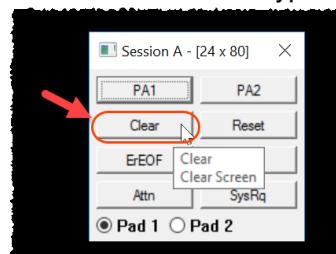
1.2.15 ► Type the transaction **bzuw** and press **Enter key**

```
bzuw
```

1.2.16 This captures the recording into a CICS Transient Data Queue (TDQ) called **BZUQ** as displayed in the message.

```
92 RECORDS WRITTEN TO QUEUE BZUQ
```

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**.

```
cemt inq td
```

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  Dat
                         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
Tdqqueue(BZUQ)
Type(Extra)
Nameind()
Triggerlevel(      )
Enablestatus( Enabled )
Openstatus( Closed )
Termid()
Tranid()
Userid()
Disposition(Old)
Iotype(Output)
Indoubt()
Indoubtwait()
Databuffers(001)
Ddname(BZUBZUQ)
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**

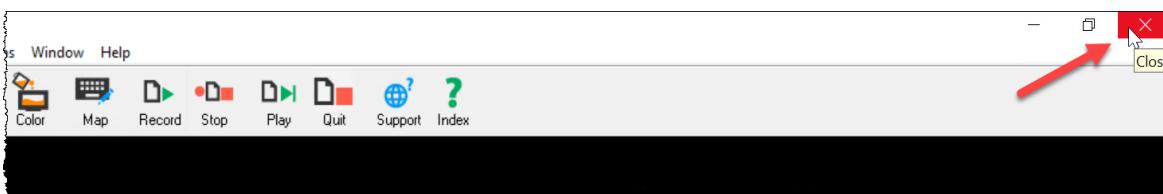
1.2.24 ►| Press PF3 to end this dialog



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

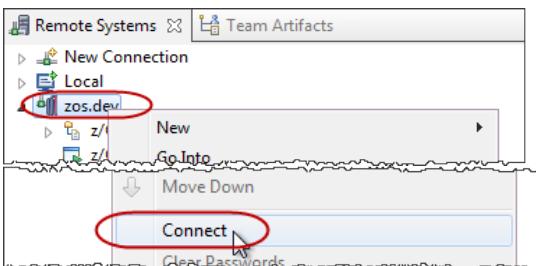
- 2.2.1 ►| Open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**

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.

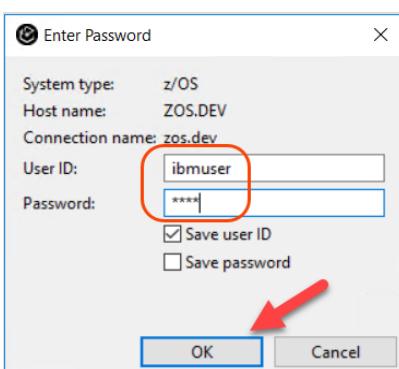
- | Using **Remote Systems** view, right click on **zos.dev** and select **Disconnect** and then **Connect**



- 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
  jcl
    HCAZPLA2.jcl
    HCAZPLA3.jcl
    HCAZPLAY.jcl
    HCAZPLCC.jcl
    HCIPCC.jcl
    HCIPDBG.jcl
    HCIPZRUN.jcl
    VTP#HCAZ.jcl
    VTP#TES1.jcl
    VTP#TES2.jcl
    ZUDBDB01.jcl
    ZUDBDB02.jcl
    ZUNITCC.jcl

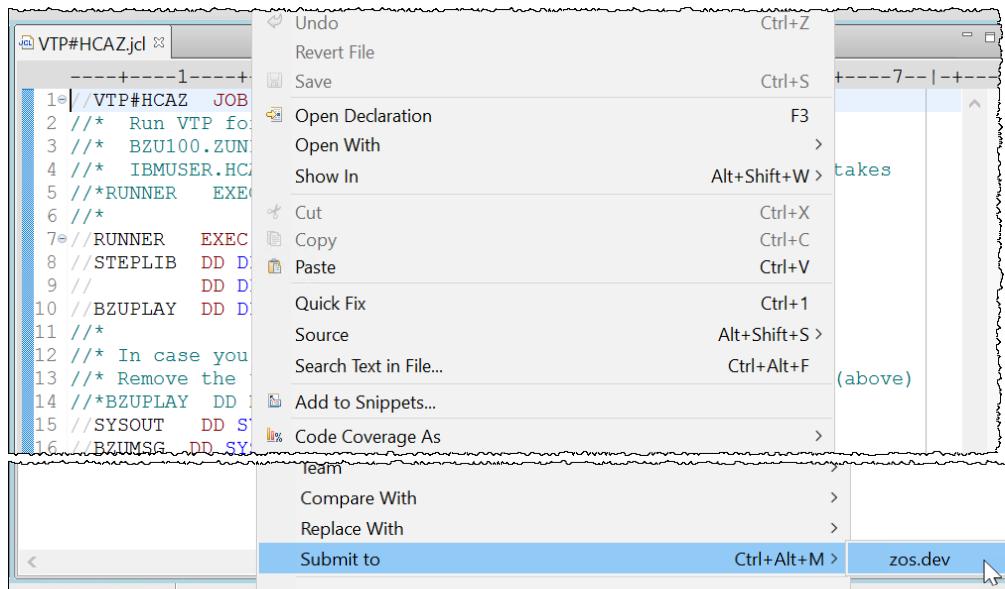
Properties Outline
VTP#HCAZ.jcl

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 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 /

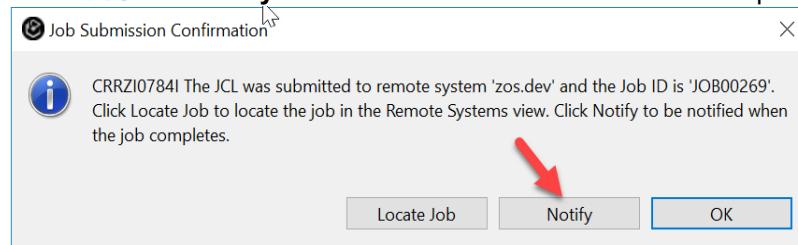
```

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



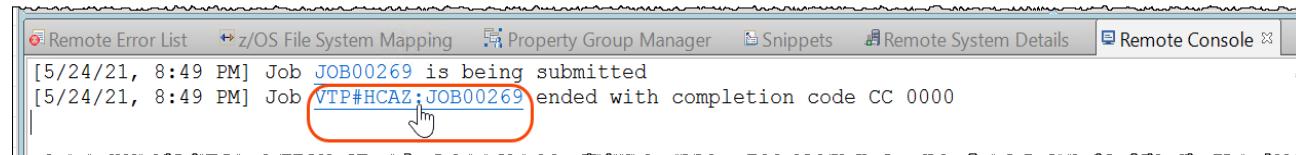
2.2.6 ► Click **Notify** to be notified when the execution is complete.



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.

The screenshot shows the 'Remote Systems' window with the 'Team Artifacts' pane open. The 'Retrieved Jobs' section shows the expanded log for 'VTP#HCAZ:JOB00269 [CC 0000]'. A red arrow points from the log to the 'RUNNER:BZUMSG' step in the artifacts pane.

2.2.9 ▶ Scroll down the report displayed. In the line 28 you will see that the program **HCP1BI01** has **DB2 CALL=SELECT** and is invoked by program **HCP1PL01**

The screenshot shows the 'Remote Systems' window with the 'Team Artifacts' pane open. The 'Retrieved Jobs' section shows the expanded log for 'VTP#HCAZ:JOB00269 [CC 0000]'. Line 28 of the log is highlighted with a red box, showing 'RECEIVED DB2 CALL=SELECT PROGRAM=HCP1BI01 LINE=247'. A red arrow points from this line in the log to the 'RUNNER:BZUMSG' step in the artifacts pane.

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 spool file titled 'IBMUSER.VTP#HCAZ.JOB00269.D0000102.?spool' is displayed, showing CICS transaction logs. Line 103 contains the message 'BZUP002I FINISHED EXECUTION RC=00'. On the right, the 'Remote Systems' window shows a tree view of system artifacts, including 'IBMUSER.HCAZ.VTP.*', 'IBMUSER.HCPA.VTP*', 'IBMUSER.POT.*', 'My Data Sets (IBMUSER.*)', 'My Favorites', 'TSO Commands', 'JES', and 'Retrieved Jobs'. Under 'Retrieved Jobs', several entries are listed, with 'RUNNER:BZUMSG' highlighted by a red circle and a red arrow pointing to it from the left.

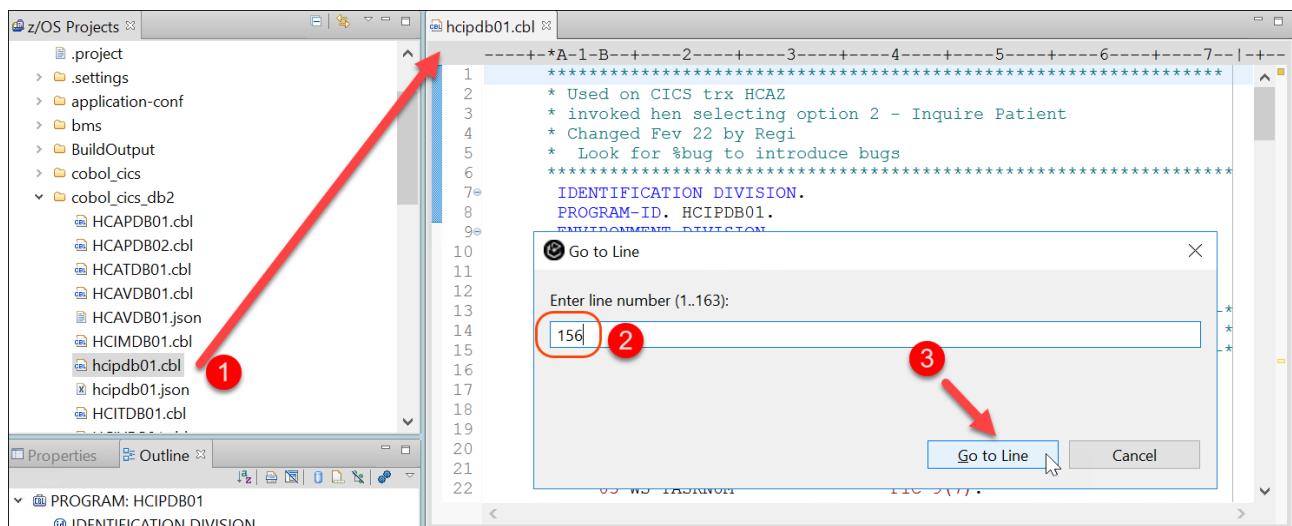
2.2.11 ►| Use **Ctrl + Shift + F4** to close all opened editors.

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.

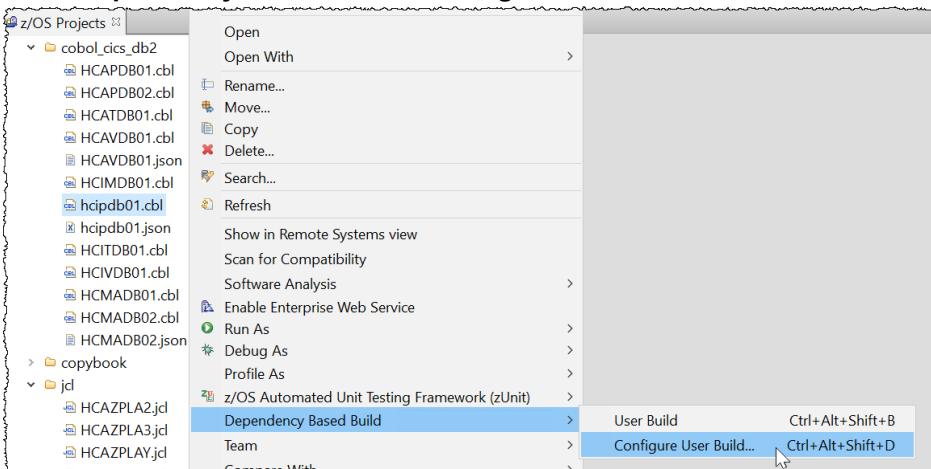
```

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      *
160      MOVE "02" to CA-NEWFIELD
161      *
162      EXIT.
163      *
164      COPY HCERRSPD.

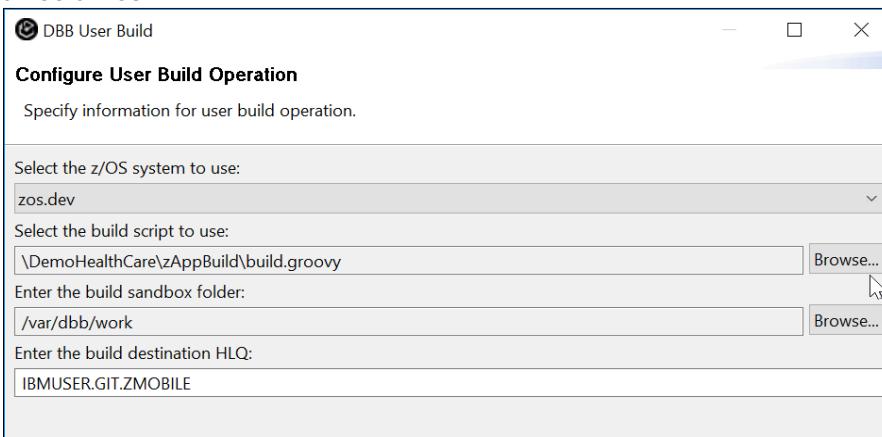
```

3.2 Building the modified program using DBB

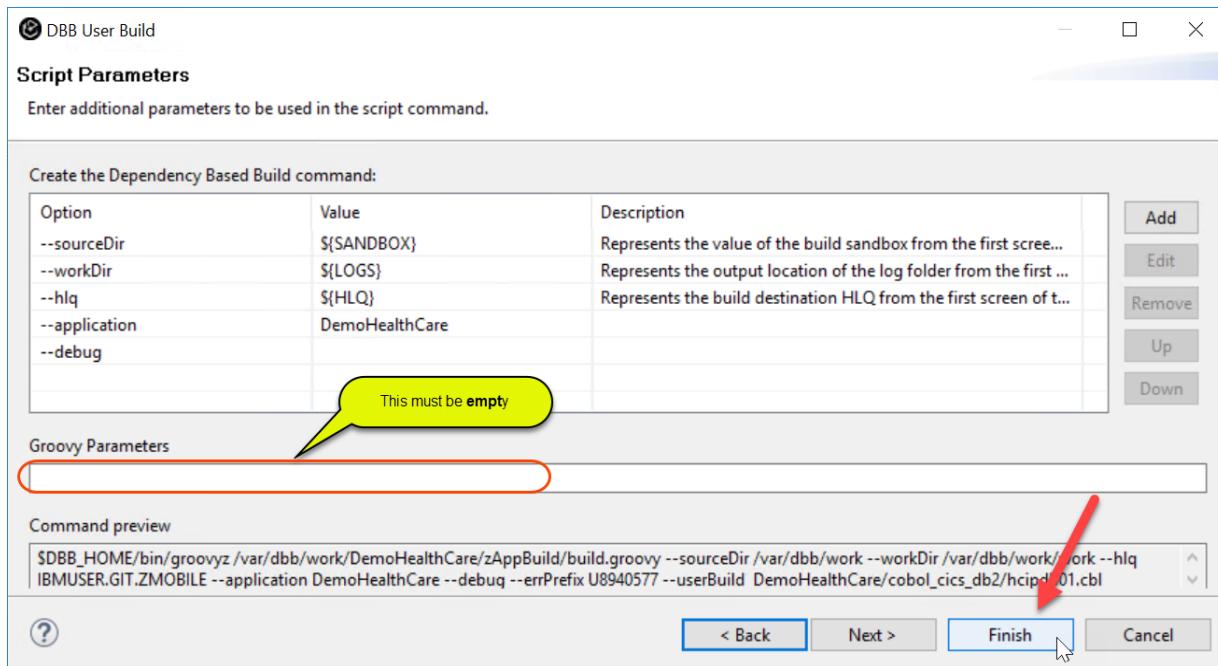
3.2.1 ► On the z/OS Projects view, right click on **hcpdb01.cbl** and select **Dependency Based Build > Configure User Build...**



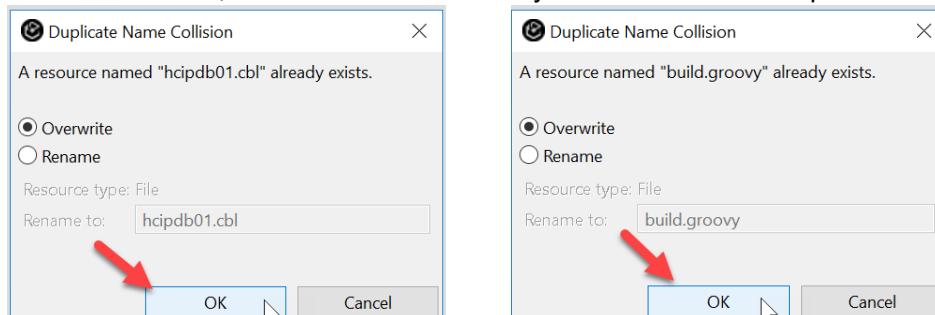
3.2.2 ► If the values are not populated, use the Browse button, specify the values below and **click Next three times**



3.2.3 ► Be sure that the field **Groovy Parameters** is empty and click **Finish**

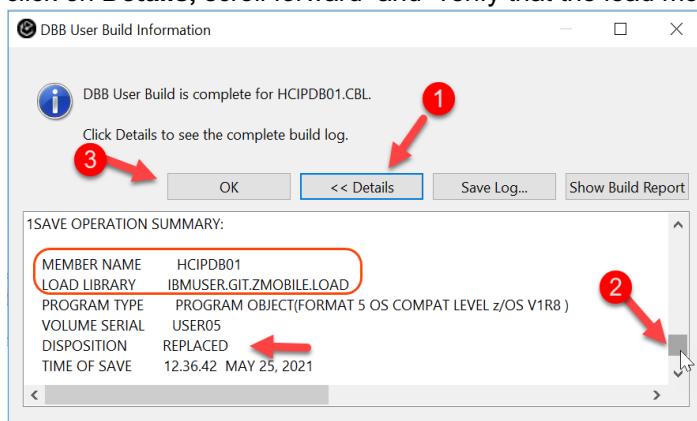


3.2.4 ► If asked, overwrite the code already on z/OS as the example below



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>

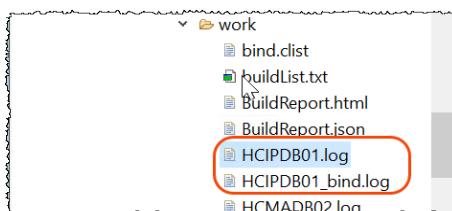
** Build ended at Tue May 25 13:37:41 CDT 2021
** Build State : CLEAN
** Total files processed : 1
** Total build time : 2 minutes, 51.887 seconds

** 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.

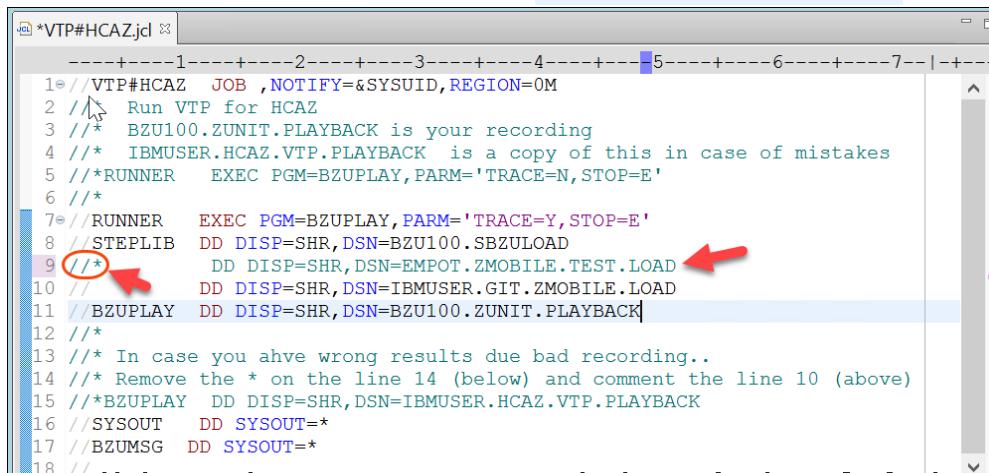
```

-----+---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 c
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=EMPOT.ZMOBILE.TEST.LOAD
11 // DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD
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**



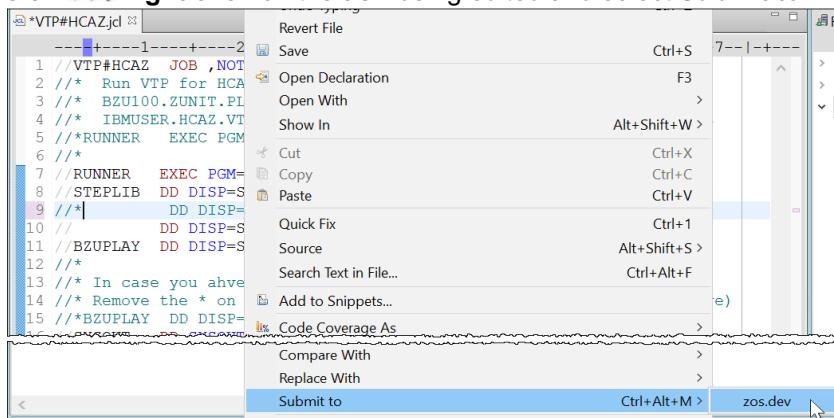
```

*VTP#HCAZ.jcl
1 //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 * (arrow)
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 //

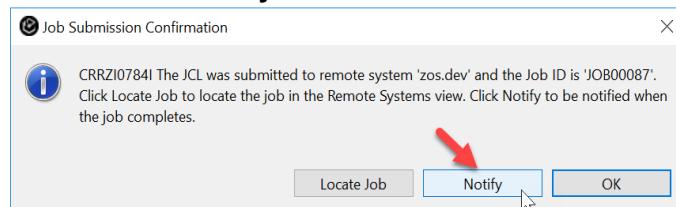
```

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



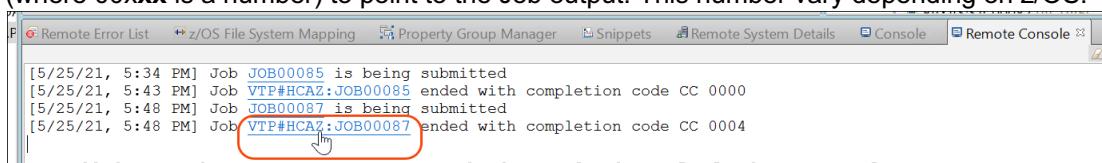
3.3.5 ► Click **Notify** to be notified when the execution is complete.



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**

The screenshot shows a terminal window on the left displaying a log of IBM VTP (Virtual Telecommunications Processor) activity. The log entries are color-coded: blue for received calls, red for recorded calls, and green for requests. A red box highlights several entries, specifically lines 35 through 40, which show a program being recorded. Red arrows point from these highlighted lines to a file browser interface on the right. The file browser, titled 'Remote Systems' and 'Team Artifacts', lists various resources under 'zos.dev'. Under 'JES', it shows 'Retrieved Jobs' for 'VTP#HCAZ:JOB00087 [CC 0004]' and 'VTP#HCAZ:JOB00085 [CC 0000]'. It also lists 'My Jobs' and 'Active Jobs'.

```
VTP#HCAZ.jcl IBMUSER.VTP#HCAZ.JOB00087.D0000102.?spool
20 BZUP401I RECEIVED CICS CALL=X'OE02' -LINK TROUBLE-HC1BI01 LINE=220
21 BZUP402I RECORDED CICS CALL=X'OE02' -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'OE08' -RETURN PROGRAM=HCIPDB01 LINE=230
25 BZUP402I RECORDED CICS CALL=X'OE08' -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=FFFFCCCC4CDC44CC9988884444444444FFFF6FF6FFFF4D889
31 PROGRAM DATA=123421405145004134594100000000000001980070113404195
32 BZUP401I RECEIVED CICS CALL=X'OE08' -RETURN PROGRAM=HC1BI01 LINE=212
33 BZUP402I RECORDED CICS CALL=X'OE08' -RETURN RECORD=28 ORIGINAL LINE=212
34 BZUP202W REQUEST=RETURN ARG=CMA RECORD=28 ADDRESS=20757100 OFFSET=28
35 PROGRAM COLS=-+---3---+---4---+---5---+---6---+---7---|+-
36 PROGRAM DATA=1234BAD NAME Dalmeida 1980-07-1134 Main
37 PROGRAM DATA=FFFFCCCC4CDC44CC9988884444444444FFFF6FF6FFFF4D889
38 PROGRAM DATA=12342140514500413459410000000000001980070113404195
39 BZUP401I RECEIVED CICS CALL=X'OE08' -RETURN PROGRAM=HC1BI01 LINE=212
40 BZUP402I RECORDED CICS CALL=X'OE08' -RETURN RECORD=28 ORIGINAL LINE=212
41 BZUP202W REQUEST=RETURN ARG=CMA RECORD=28 ADDRESS=20757100 OFFSET=28
42 PROGRAM=HC1BI01 LINE=212 MODE=INPT
43 PROGRAM COLS=-+---3---+---4---+---5---+---6---+---7---|+-
44 PROGRAM DATA=1234BAD NAME Dalmeida 1980-07-1134 Main
45 PROGRAM DATA=FFFFCCCC4CDC44CC9988884444444444FFFF6FF6FFFF4D889
46 PROGRAM DATA=1234214051450041345941000000000001980070113404195
47 BZUP401I RECEIVED CICS CALL=X'OE08' -RETURN PROGRAM=HC1BI01 LINE=212
48 BZUP402I RECORDED CICS CALL=X'OE08' -RETURN RECORD=28 ORIGINAL LINE=212
```

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.

The screenshot shows the IBM i Navigator interface. On the left, a terminal window displays a JCL job named VTP#HCAZ.jcl. The job contains several BZUP401I messages indicating CICS calls and a BZUP002I message indicating the finished execution with RC=04. It also includes sections for DB2 statistics and CICS statistics. On the right, a 'Remote Systems' tree view is shown, with 'zos.dev' expanded to show z/OS UNIX Files, z/OS UNIX Shells, MVS Files, TSO Commands, and JES, which has 'Retrieved Jobs' expanded to show entries for VTP#HCAZ:JOB00119 through JOB00085.

```
*VTP#HCAZ.jcl IBMUSER.VTP#HCAZ.JOB00119.D0000102?.spool
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**.

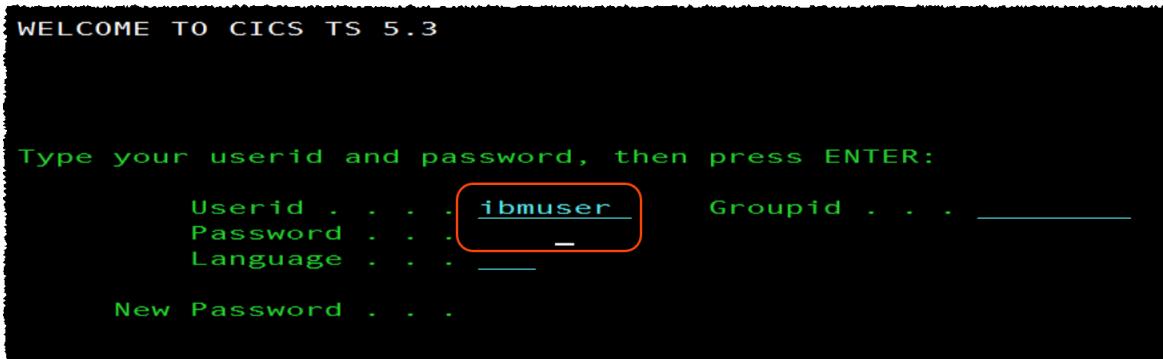
```
====> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
====> Enter L followed by the APPLID
====> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"
```

```
l cicsts53 _
```

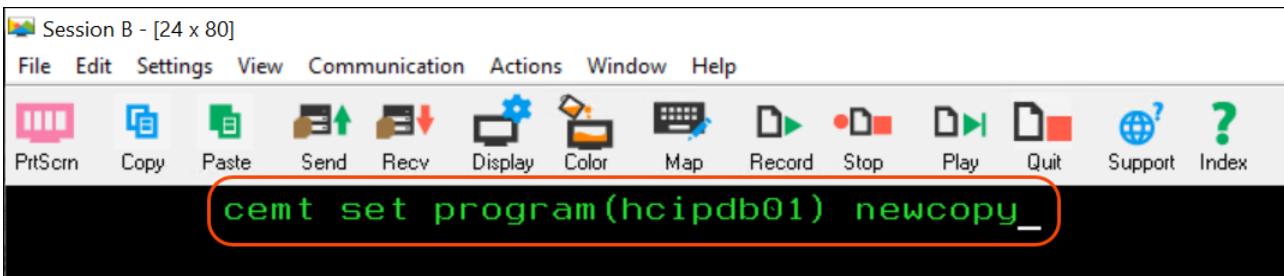
B

24/011

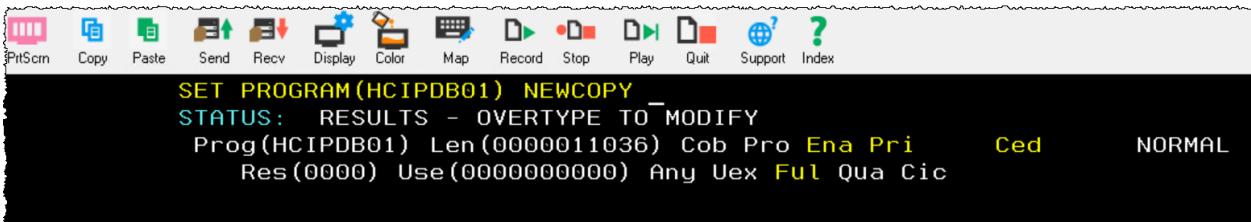
4.1.3 ► Sign on using **ibmuser** as the userid and **sys1** as the password and press **Enter key**.



4.1.4 ► Type **cemt set program(hcipdb01) newcopy** and press **Enter key**.



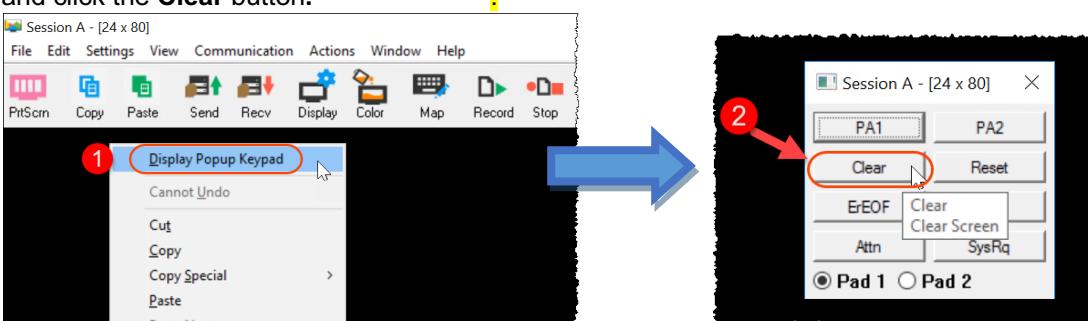
4.1.5 This below indicates the **NEWCOPY** was successful:



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.

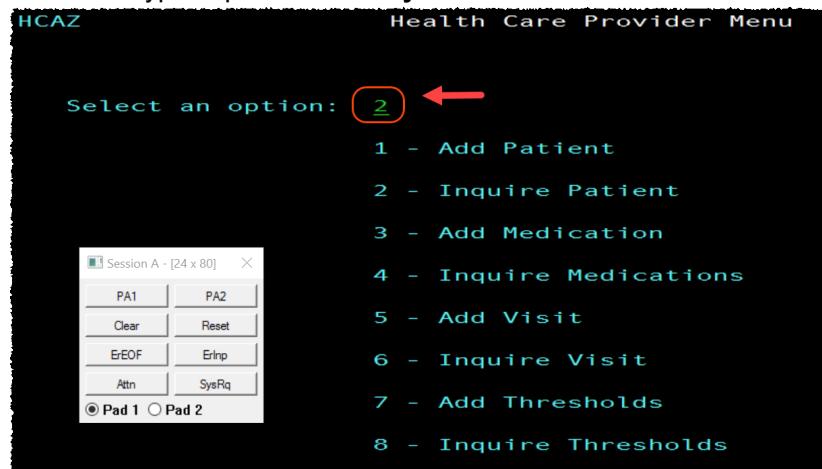


4.2 Executing HCAZ transaction

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**.



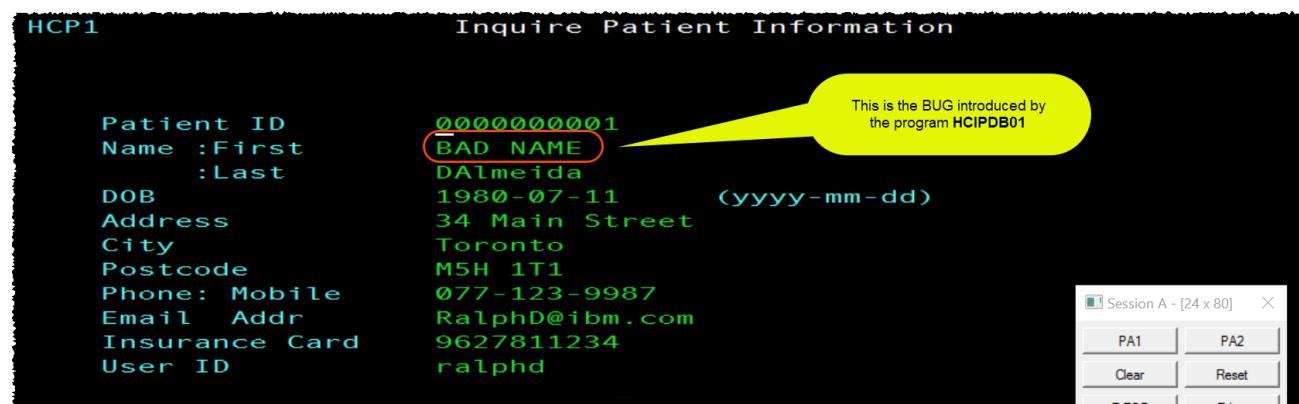
4.2.3 ► Type 1 press **Enter key**. (You can close the stick keypad if you want)



You should have the patient data displayed.

Notice that instead of **Ralph** the **BAD NAME** is displayed

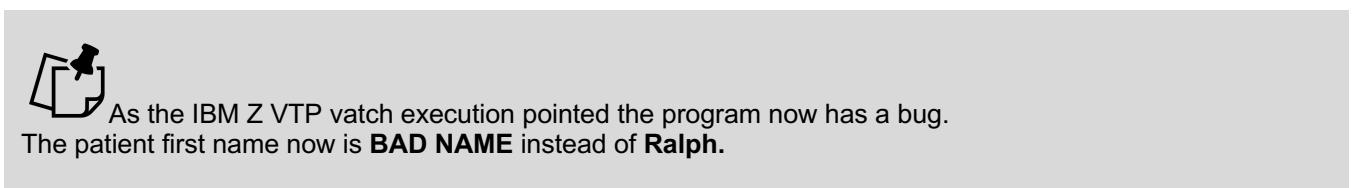
This confirm the report provided by the batch JCL executed before.



4.2.4 ► Press PF3 to exit the program



4.2.5 ► Close the terminal emulation



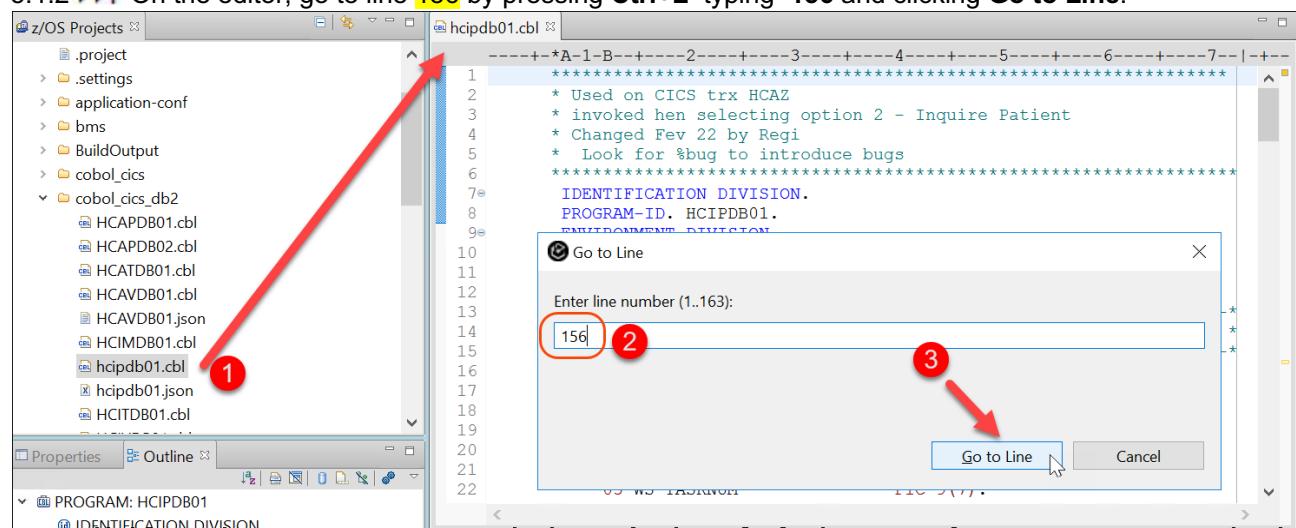
Section 5 Use IDz to fix the bug and recompile/bind the program.

Using IDz again you will remove the bug introduced Below the details

5.1 Modifying one COBOL program again to remove the bug

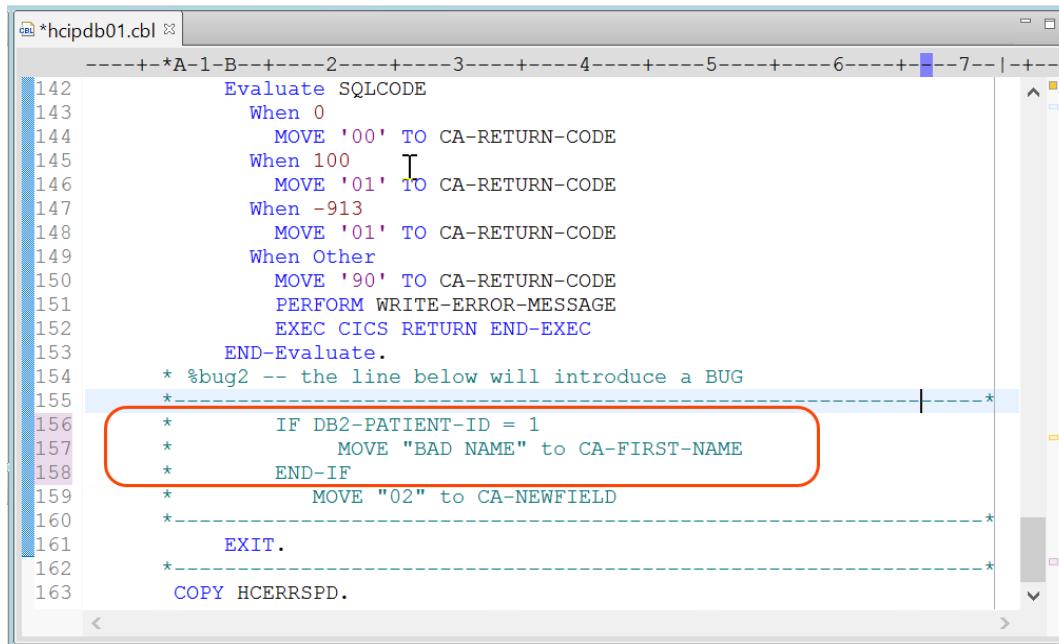
5.1.1 ► Using IDz and the z/OS Projects view, open **hcipdb01.cbl** under **cobol_cics_db2** by double clicking on it

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.



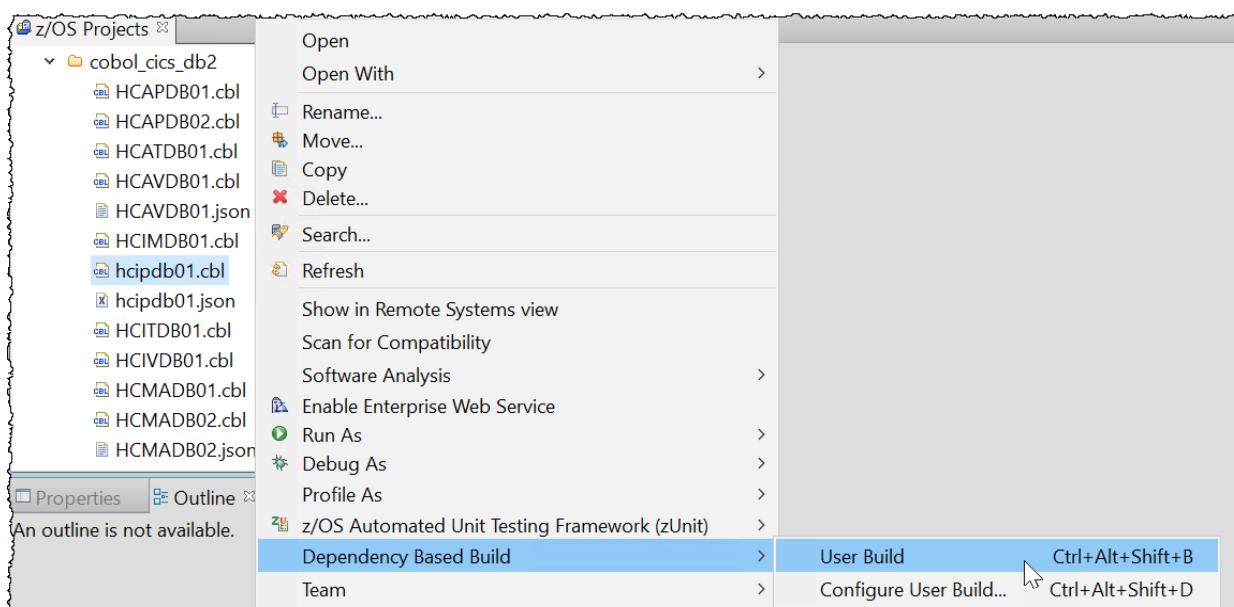
```

* *-----*A-1-B-----2-----3-----4-----5-----6-----7-----I-----*
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.

```

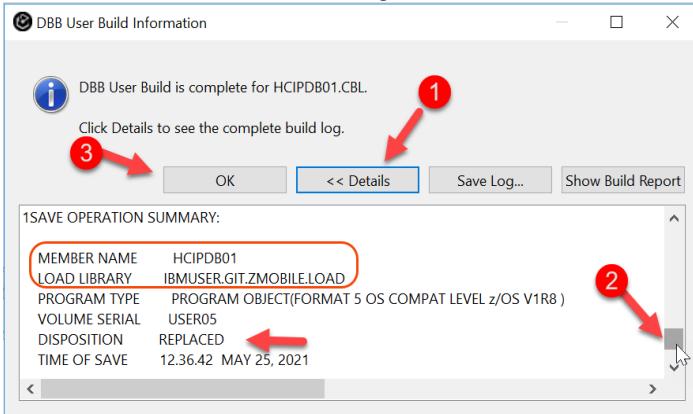
5.2 Rebuilding the fixed program using DBB

5.2.1 ► On the z/OS Projects view, right click on **hcipdb01.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
/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>
```

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

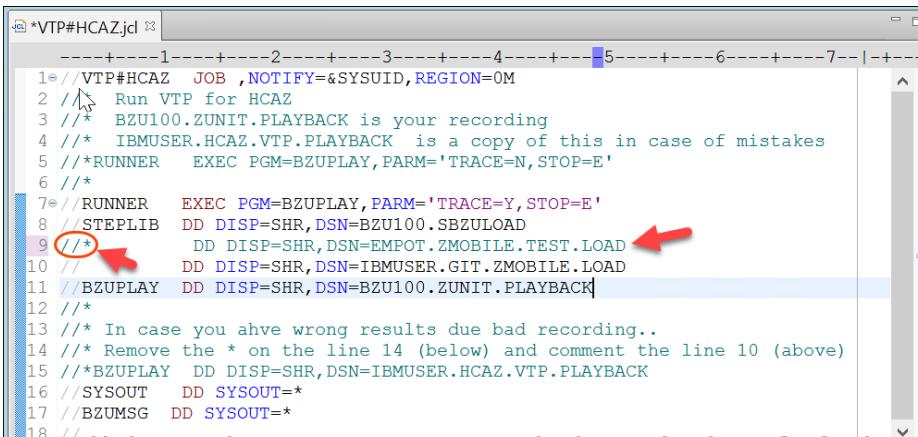
Since we fixed the bug , now the VTP test should execute with Return Code 00.

6.1.1 ► Using the project **DemoHealthCare** expand **jcl** and double click on **VTP#HCAZ.jcl** to edit the JCL that runs VTP.

```
-----1-----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
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.SBZUL0AD
9 //          DD DISP=SHR,DSN=EMPT.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=*
```

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**

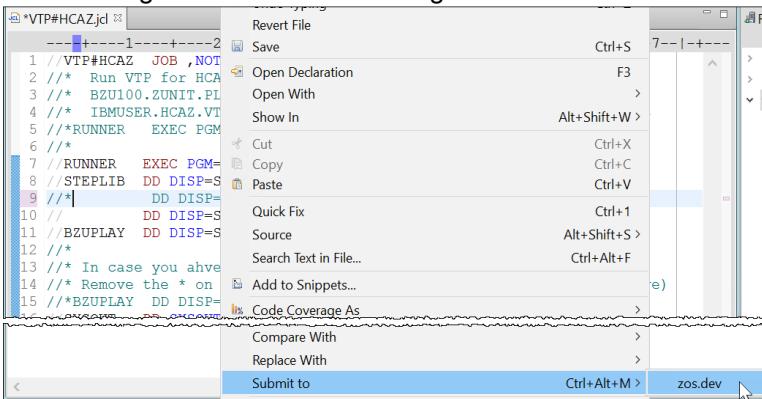


```

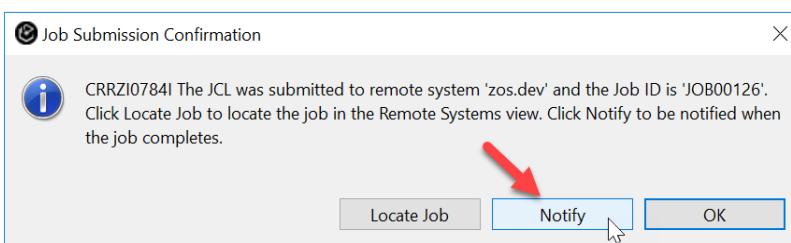
*VTP#HCAZ.jcl
1 //> VTP#HCAZ JOB ,NOTIFY=&SYSUID,REGION=OM
2 //> Run VTP for HCAZ
3 //** BZU100.ZUNIT.PLAYBACK is your recording
4 //** IBMUSER.HCAZ.VTP.PLAYBACK is a copy of this in case of mistakes
5 //**RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=N,STOP=E'
6 /**
7 //>RUNNER EXEC PGM=BZUPLAY,PARM='TRACE=Y,STOP=E'
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 //

```

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



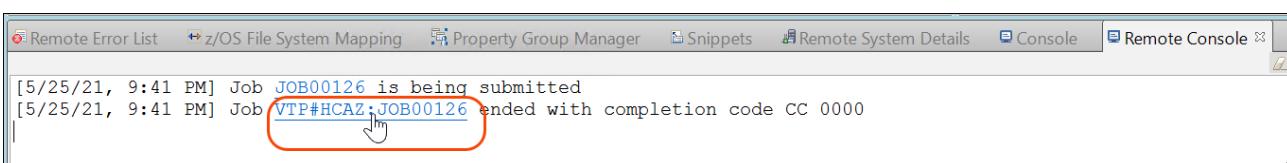
6.1.4 ► Click **Notify** to be notified when the execution is complete.



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 vary 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 "IBMUSER.VTP#HCAZ.JOB00126.D0000102.spool" containing a log of CICS transactions. A red box highlights the "RUNNER:BZUMSG" step at the bottom of the log. On the right is the "Remote Systems" tree view under "Team Artifacts". It shows expanded nodes for "VTP#HCAZ:JOB00126" and "RUNNER:BZUMSG". A red arrow points from the "RUNNER:BZUMSG" node in the tree to the "RUNNER:BZUMSG" step in the log editor.

```

1 22 BZUP401I RECEIVED CICS CALL=X'1802'-RECEIVE_MAP PROGRAM=HCP1PL01 LINE=658
2 23 BZUP402I RECORDED CICS CALL=X'1802'-RECEIVE_MAP RECORD=19 ORIGINAL LINE=65
3 24 BZUP401I RECEIVED CICS CALL=X'OE02'-LINK PROGRAM=HCP1PL01 LINE=669
4 25 BZUP402I RECORDED CICS CALL=X'OE02'-LINK RECORD=20 ORIGINAL LINE=669
5 26 BZUP401I RECEIVED CICS CALL=X'OE02'-LINK PROGRAM=HCIPBI01 LINE=220
6 27 BZUP402I RECORDED CICS CALL=X'OE02'-LINK RECORD=22 ORIGINAL LINE=220
7 28 BZUP700I RECEIVED DB2 CALL=SELECT PROGRAM=HCIPDB01 LINE=235
8 29 BZUP701I RECORDED DB2 CALL=SELECT RECORD=25 ORIGINAL LINE=247
9 30 BZUP401I RECEIVED CICS CALL=X'OE08'-RETURN PROGRAM=HCIPDB01 LINE=230
10 31 BZUP402I RECORDED CICS CALL=X'OE08'-RETURN RECORD=26 ORIGINAL LINE=241
11 32 BZUP401I RECEIVED CICS CALL=X'OE08'-RETURN PROGRAM=HCIPBI01 LINE=212
12 33 BZUP402I RECORDED CICS CALL=X'OE08'-RETURN RECORD=28 ORIGINAL LINE=212
13 34 BZUP401I RECEIVED CICS CALL=X'1804'-SEND_MAP PROGRAM=HCP1PL01 LINE=688
14 35 BZUP402I RECORDED CICS CALL=X'1804'-SEND_MAP RECORD=31 ORIGINAL LINE=688
15 36 BZUP401I RECEIVED CICS CALL=X'OE08'-RETURN PROGRAM=HCP1PL01 LINE=747
16 37 BZUP402I RECORDED CICS CALL=X'OE08'-RETURN RECORD=32 ORIGINAL LINE=747
17 38 BZUP400I STARTING TRANSACTION=HCP1 USING PROGRAM=HCP1PL01
18 39 BZUP401I RECEIVED CICS CALL=X'0206'-HANDLE_AID PROGRAM=HCP1PL01 LINE=649
19 40 RZUP402T RECORDED CICS CALL=X'0206'-HANDLE_AID RECORD=35 ORTGTMAT T.TNE=649

```

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.

The screenshot shows the IBM Z VTP interface. On the left is a code editor window titled "IBMUSER.VTP#HCAZ.JOB00269.D0000102.spool" containing a log of CICS transactions. A red box highlights the "BZUP002I FINISHED EXECUTION RC=00" message at the bottom of the log. On the right is the "Remote Systems" tree view under "Team Artifacts". It shows expanded nodes for "VTP#HCAZ:JOB00269" and "RUNNER:BZUMSG". A red arrow points from the "RUNNER:BZUMSG" node in the tree to the "BZUP002I FINISHED EXECUTION RC=00" message in the log editor.

```

1 96 BZUP401I RECEIVED CICS CALL=X'1802'-RECEIVE_MAP PROGRAM=HCT1PL01 LINE=653
2 97 BZUP402I RECORDED CICS CALL=X'1802'-RECEIVE_MAP RECORD=89 ORIGINAL LINE=65
3 98 BZUP401I RECEIVED CICS CALL=X'1806'-SEND_TEXT PROGRAM=HCT1PL01 LINE=734
4 99 BZUP402I RECORDED CICS CALL=X'1806'-SEND_TEXT RECORD=91 ORIGINAL LINE=734
5 100 BZUP401I RECEIVED CICS CALL=X'OE08'-RETURN PROGRAM=HCT1PL01 LINE=740
6 101 BZUP402I RECORDED CICS CALL=X'OE08'-RETURN RECORD=92 ORIGINAL LINE=740
7 102 ****
8 103 BZUP002I FINISHED EXECUTION RC=00
9 104 ****
10 105 BZUP300I DB2 STATISTICS
11 106 SELECT NORMAL=3
12 107 ****
13 108 BZUP300I CICS STATISTICS
14 109 HANDLE_CONDITION NORMAL=6
15 110 HANDLE_AID NORMAL=6
16 111 LINK NORMAL=6
17 112 RETURN NORMAL=12
18 113 RECEIVE_MAP NORMAL=6
19 114 SEND_MAP NORMAL=6
20 115 SEND_TEXT NORMAL=1
21 116 ****

```

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 6 – (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.

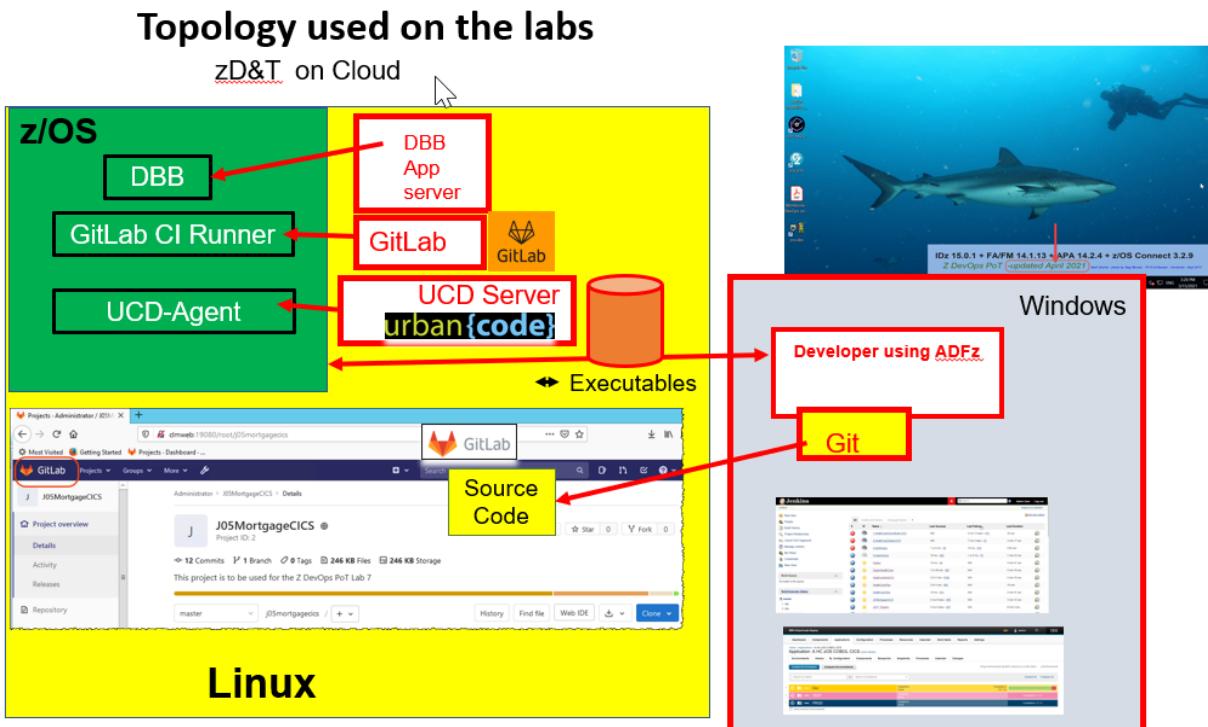
The updated code will be deployed to CICS using [UrbanCode Deploy](#) (UCD)

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.



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

The screenshot shows a web browser window with the following details:

- File Edit View History Bookmarks Tools Help
- New Tab x +
- Search with Google or enter address
- Most Visited Projects - Dashboard · ... Healthcare-issues Genapp - Wrong DOB...
- Genapp - Wrong DOB on SSC1 (#1) · Issues · Administrator / dbb-pipeline · GitLab http://clmweb:19080/root/dbb-pipeline/-/issues/1

This explains the current bug on the application

The screenshot shows the following components:

- Administrator > dbb-pipeline > Issues > #1
- Opened 28 minutes ago by Administrator Maintainer Close issue New issue
- Genapp - Wrong DOB on SSC1**
- When running **SSC1** transaction on **CICSTS5.3** the Date of Birth (DOB) is **0000-00-00**. Look at program **LGICDB01** that retrieves this value from the DB2 Table.
- SSC1 General Insurance Customer Menu**
- 1. Cust Inquiry Cust Number 0000000001
2. Cust Add Cust Name :First Andrew
4. Cust Update :Last Pandy
DOB House Name 0000-00-00 (yyyy-mm-dd)
House Number 34
Postcode PI10100
Phone: Home 01962 811234
Phone: Mob 07799 123456
Email Addr A.Pandy@beebhouse.com

- 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.

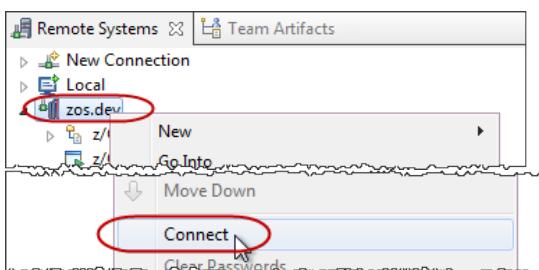
► Verify that the message indicates that it is Version **15.0.1**

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**.

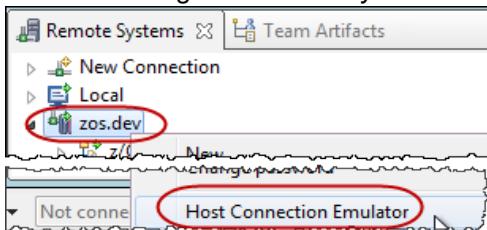


1.1.4 ► Type **ibmuser** as userid and **sys1** as password. Click **OK** to connect to z/OS.

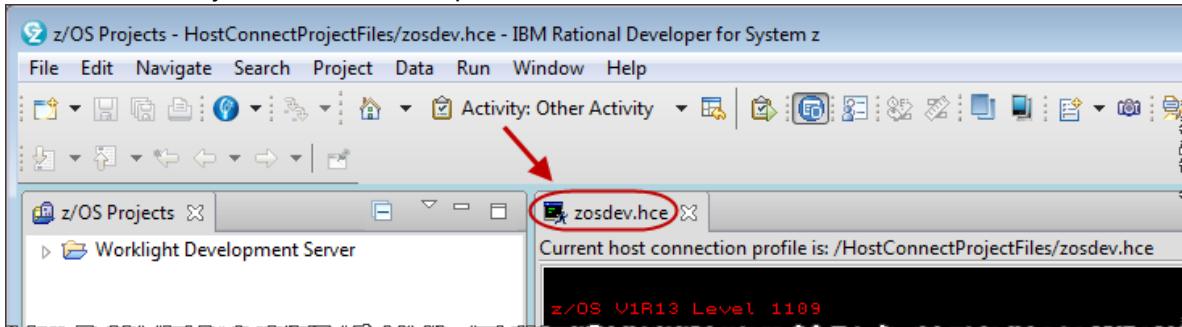


1.1.5 ► Wait until connection is complete (look at the bottom and left until the green bar disappears)

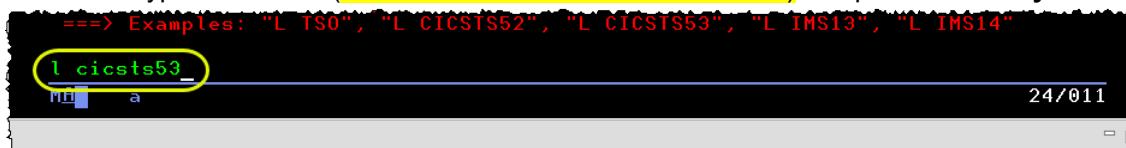
1.1.6 ► Using the **Remote Systems** view, right click on **zos.dev** and select **Host Connection Emulator**.



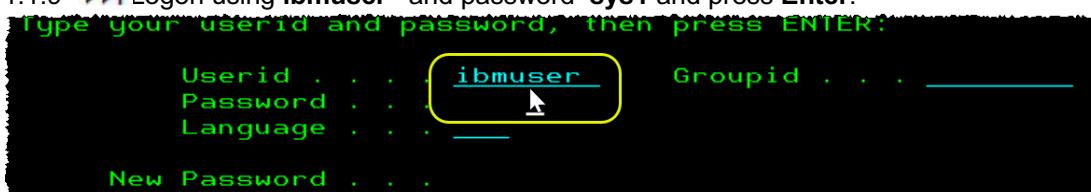
1.1.7 ► Since you will need more space, **double-click** on the **zosdev.hce** title



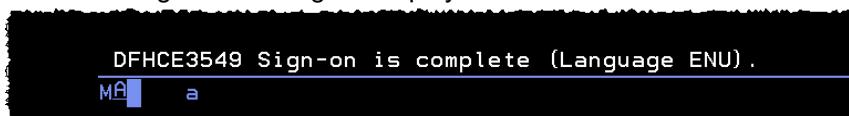
1.1.8 ► Type **I cicsts53**. (where "I" is the lower case of letter "L") and press **Enter key**.



1.1.9 ► Logon using **ibmuser** and password **sys1** and press **Enter**.



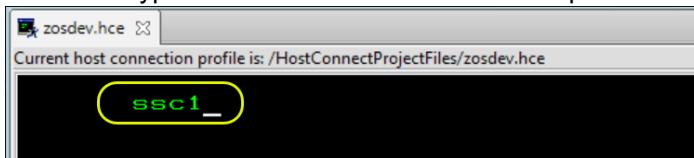
1.1.10 The sign-on message is displayed



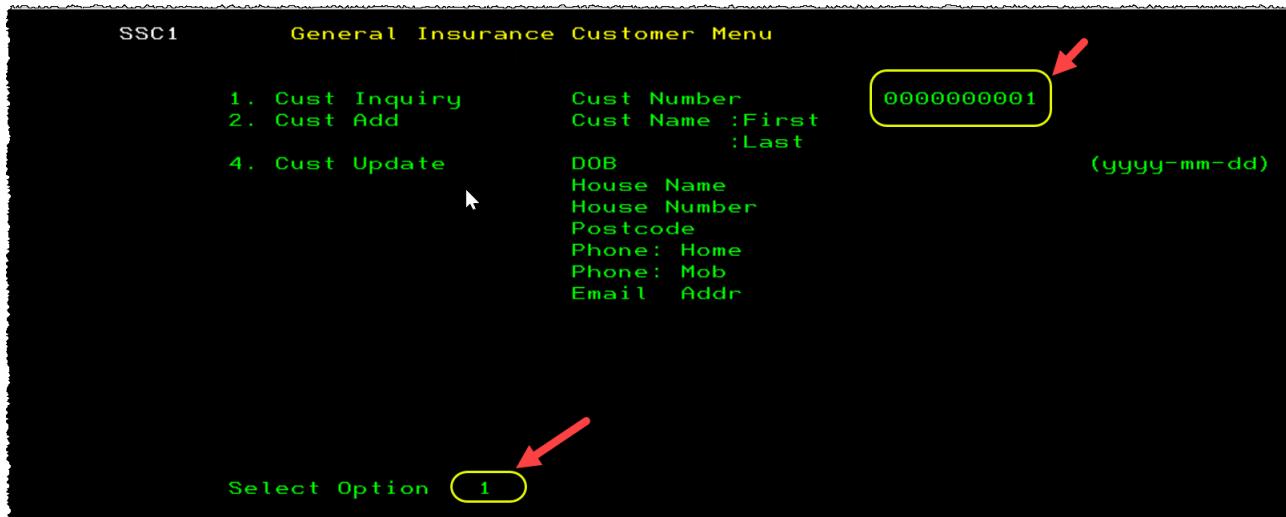
1.2 Run CICS transaction SSC1

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.

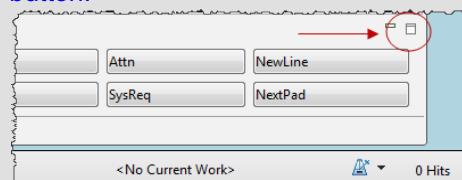
1.2.1 ► Type the CICS transaction **ssc1** and press the **Enter key**.



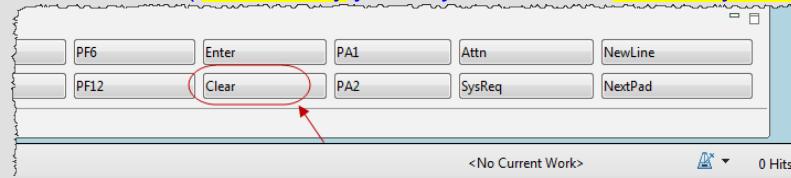
1.2.2 ► Move the mouse to last 0 and type 1, use the tab key and type 1 as **Select option** and press **Enter**



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)

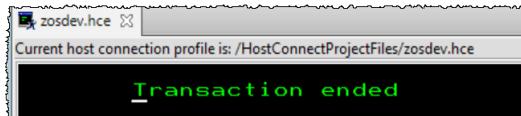


1.2.3 ► The data below is retrieved from a DB2 table .

Notice that **DOB** (*Date Of Birth*) 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

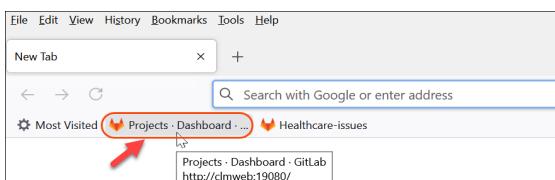
The Genapp Application used in this lab has its source code in the *GitLab Repository* that is on a Linux system. You must connect to the GitLab Repository and clone the Git project into the IDz. This brings the source code into your IDz workspace for edits and build.

2.1.1 Before cloning the Git repository, you should visualize the code stored at GitLab.

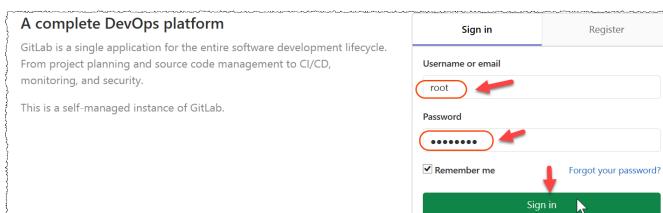
► Go back to the web browser clicking in the icon in the bottom of your screen



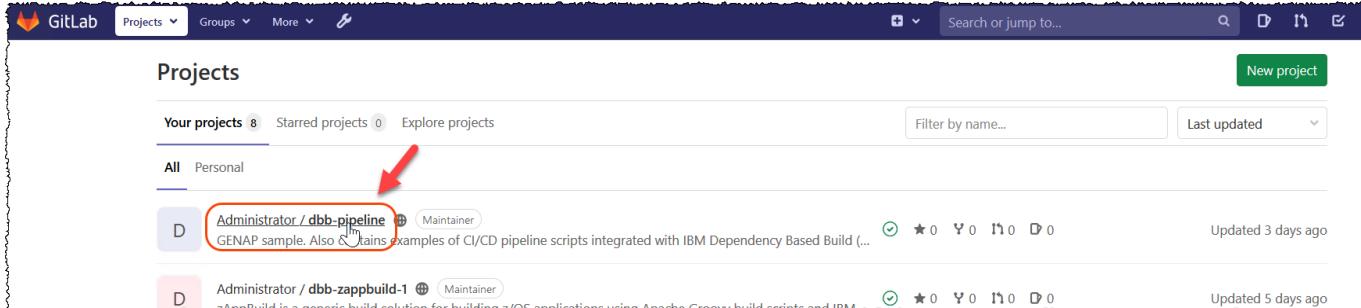
2.1.2 ► Click on this bookmark below to access **GitLab**. The URL used is <http://clmweb:19080/>



► If asked for credentials use **root** and password: **zdtlinux**



2.1.3  Click on the hyperlink below to see the **dbb-pipeline** repository.

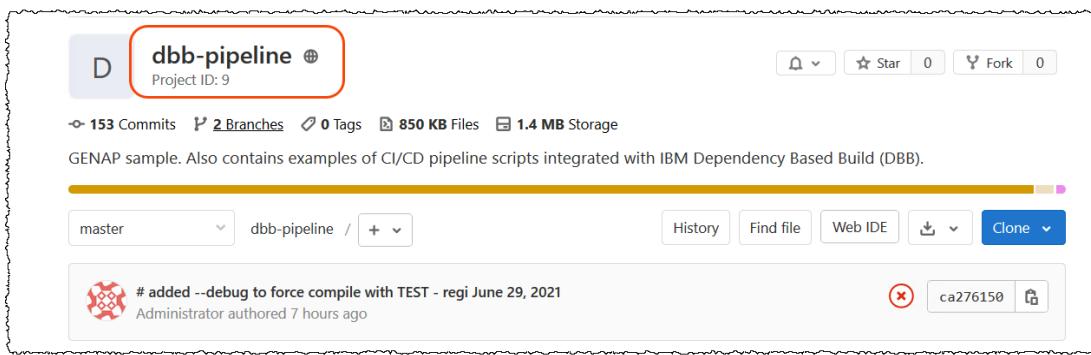


The screenshot shows the GitLab interface with the 'Projects' tab selected. There are two projects listed:

- Administrator / dbb-pipeline**: Maintainer, GENAP sample. Also contains examples of CI/CD pipeline scripts integrated with IBM Dependency Based Build (DBB). Updated 3 days ago.
- Administrator / dbb-zappbuild-1**: Maintainer, Updated 5 days ago.

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



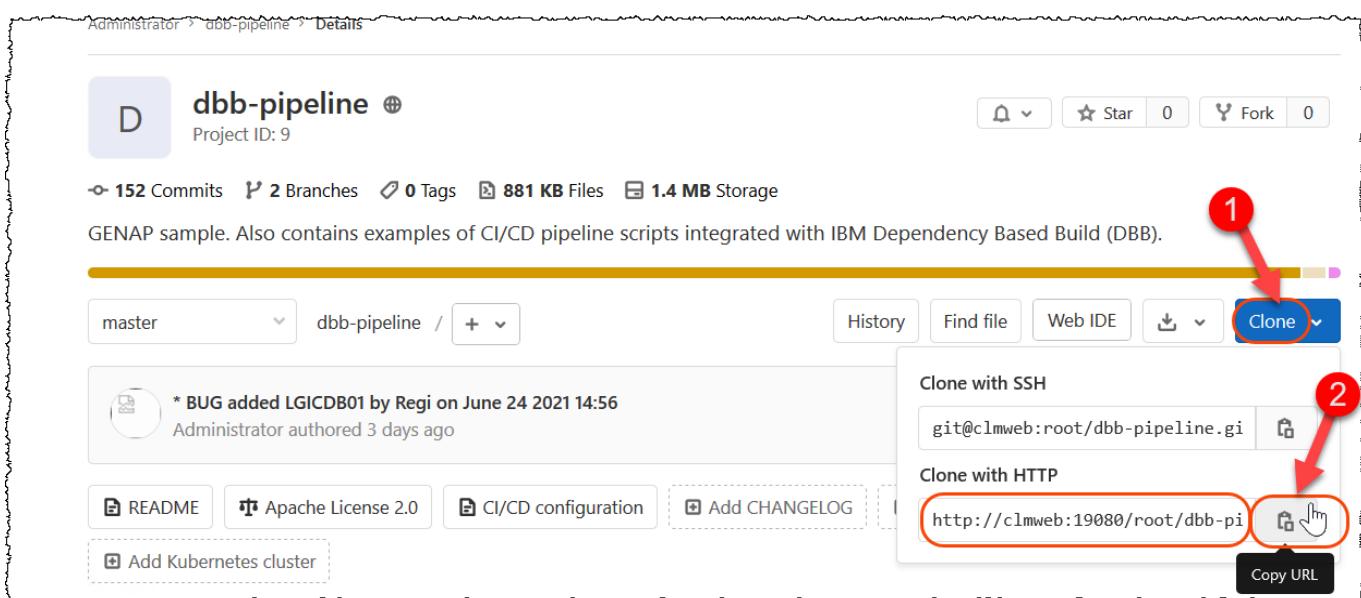
The screenshot shows the repository details for 'dbb-pipeline'. It includes the following information:

- 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
- Breadcrumbs: dbb-pipeline /
- Actions: History, Find file, Web IDE, Download, Clone (blue button), Copy (link icon)
- Commit: # added --debug to force compile with TEST - regi June 29, 2021 (ca276150)

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.



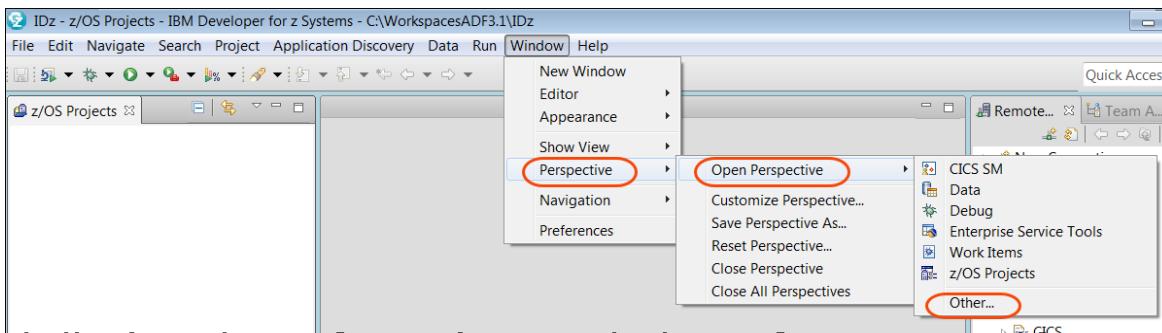
The screenshot shows the repository details for 'dbb-pipeline' with the following focus on cloning:

- Clone** button (blue button with a dropdown arrow)
- Clone with SSH**: git@clmweb:root/dbb-pipeline.git (link icon)
- Clone with HTTP**: http://clmweb:19080/root/dbb-pipeline.git (link icon)
- Copy URL** button (black button with a copy icon)

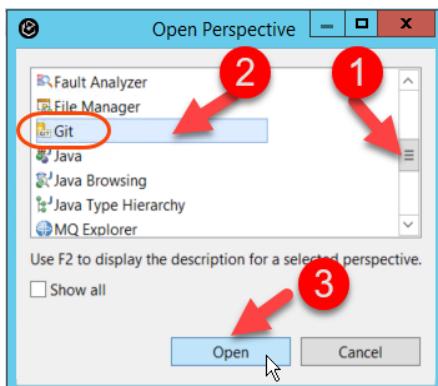
2.1.6 ► Go back to IDz using the icon that is on the base of your screen:



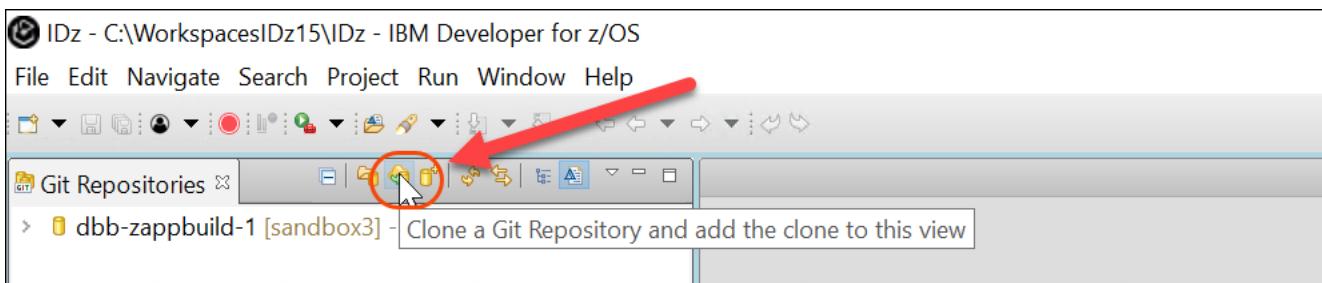
2.1.7 ► Open the **Git** perspective by selecting **Window > Perspective > Open Perspective > Other...**



2.1.8 ► Select **Git** and click **Open**

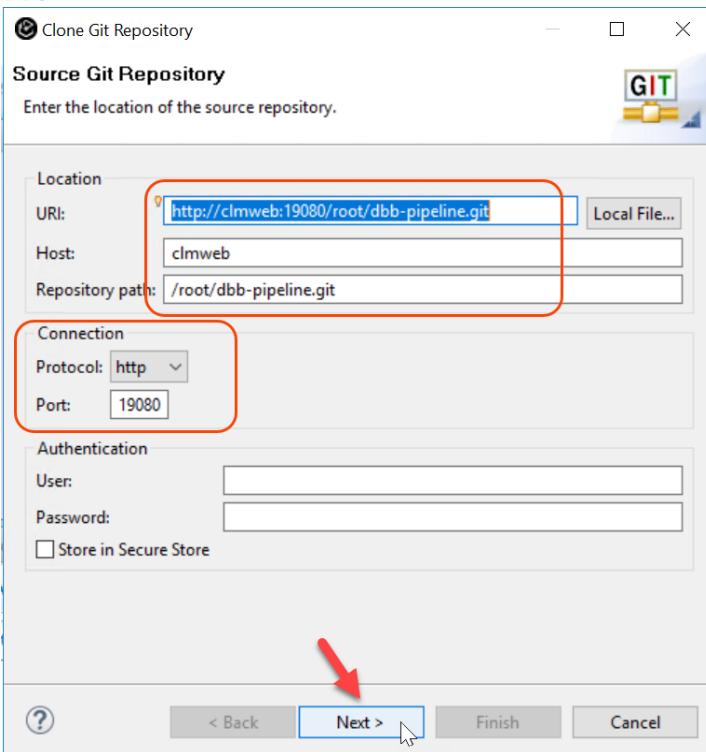


2.1.9 ► In the **Git Repositories** tab, click on the icon to 'Clone a Git repository'.



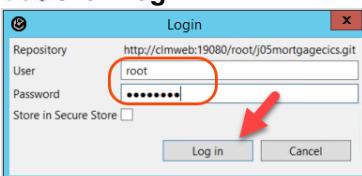
2.1.9 ► The values copied from the web page (copy URL) will be shown in the Clone Git Repository.
 Tip: In case you don't see it, got back to the page and copy it again (steps 2.1.1-2.1.5 above).

► Click Next

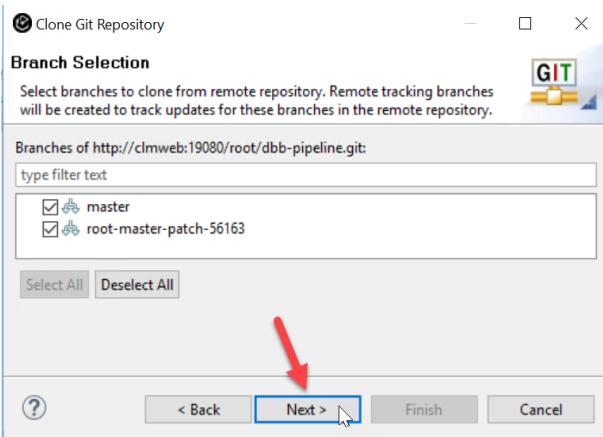


2.1.10 If a dialog asks for login the credentials are a user user: **root** and password **zdtlinux**

► Click Log in

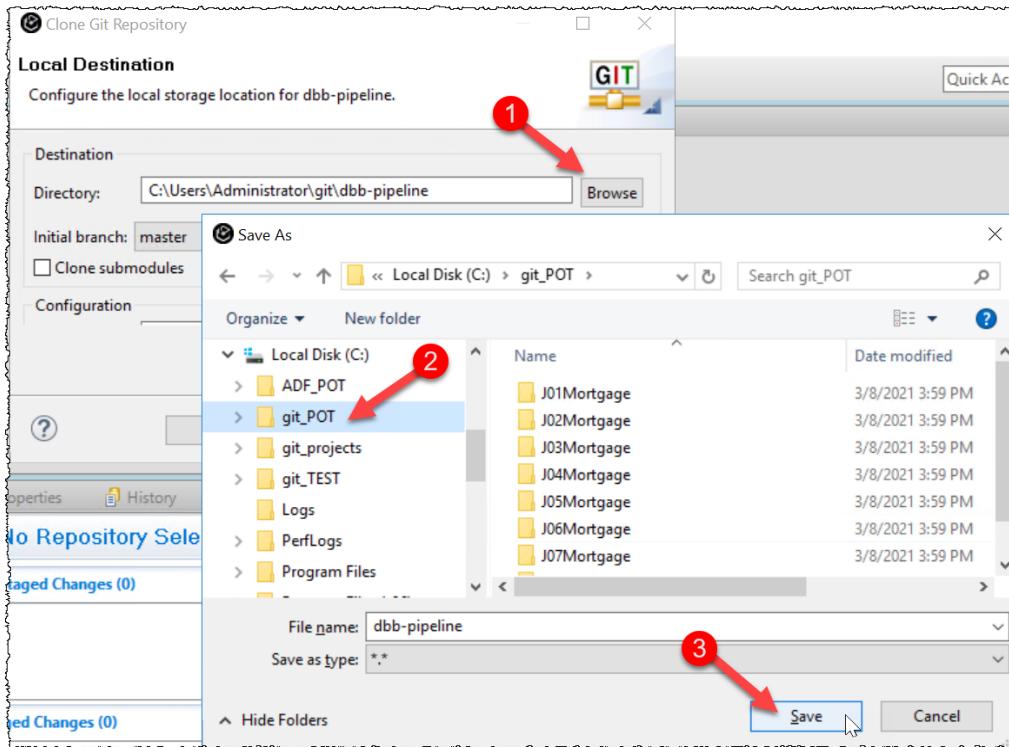


2.1.11 We will clone all branches. ► Click Next



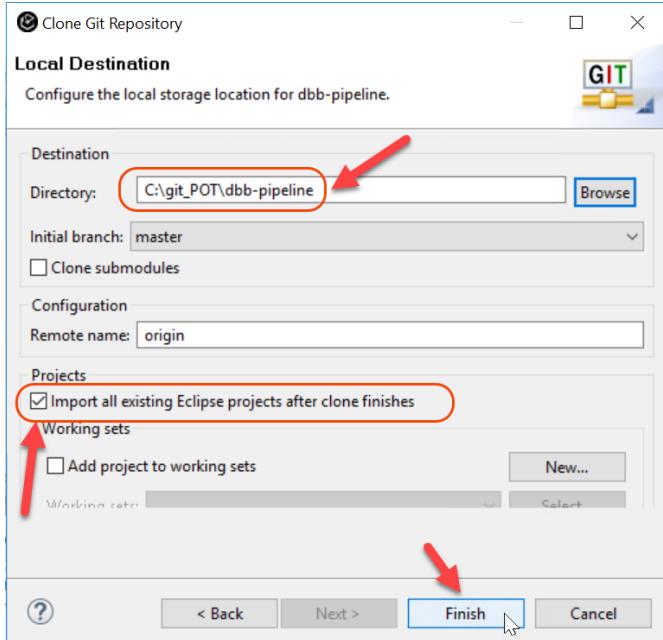
2.1.12 We will clone the repository on your local windows and import to be shown on IDz

- 1 ►► Use **Browse** button to select the directory **C:\git_POT** (on left).
- 2 ►► Click folder **git_POT** on left
- 3 ►► Click **Save**

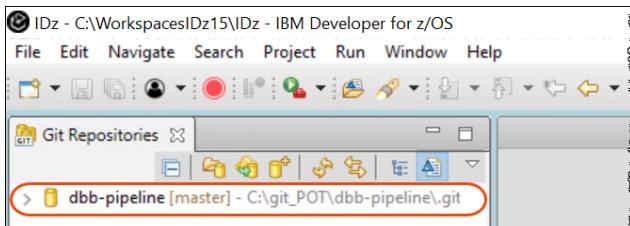


2.1.13 The *Directory* now should point to **C:\git_POT\dbb-pipeline**

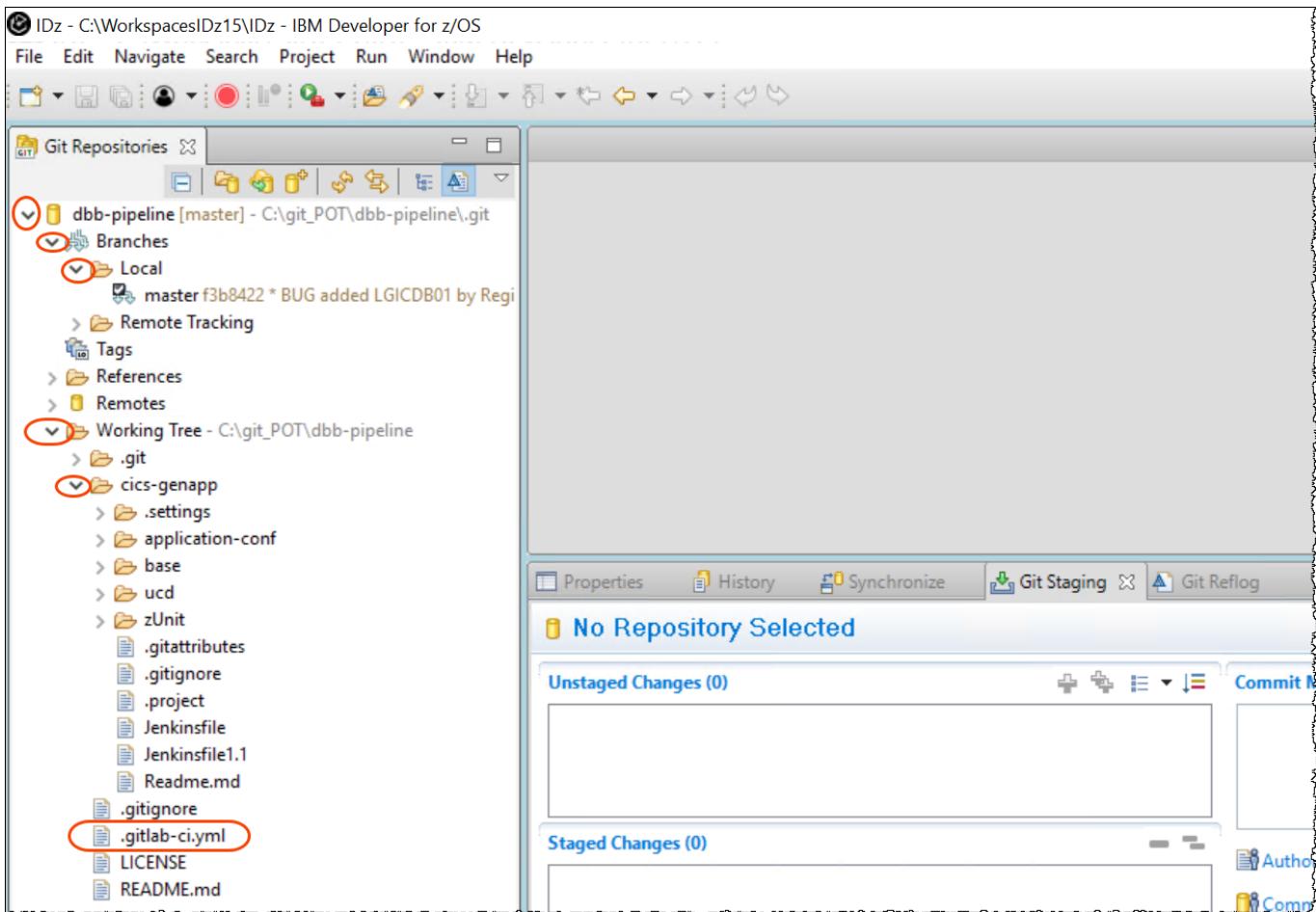
- Be sure that you selected **Import all existing Eclipse projects after clone finishes**
- Click **Finish**



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

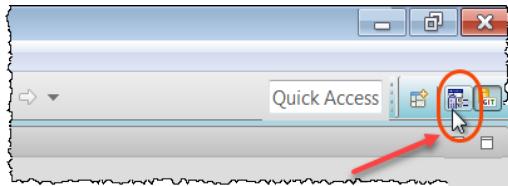


2.2 Verify the code cloned using z/OS projects perspective

The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

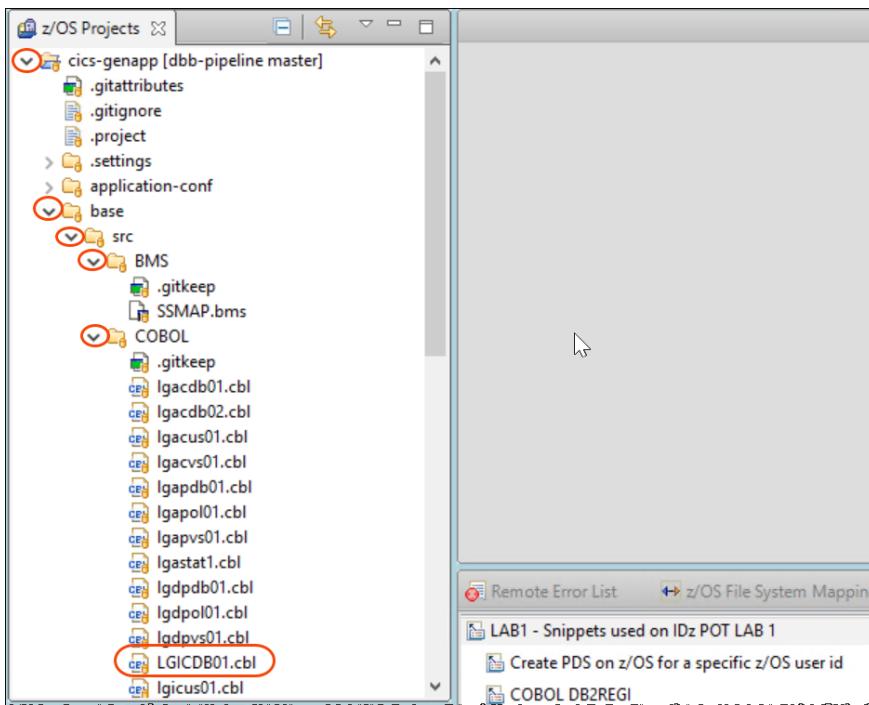
Notice that you have cloned the project at your local workstation but later you will need to connect to z/OS to be able to compile and build your programs.

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

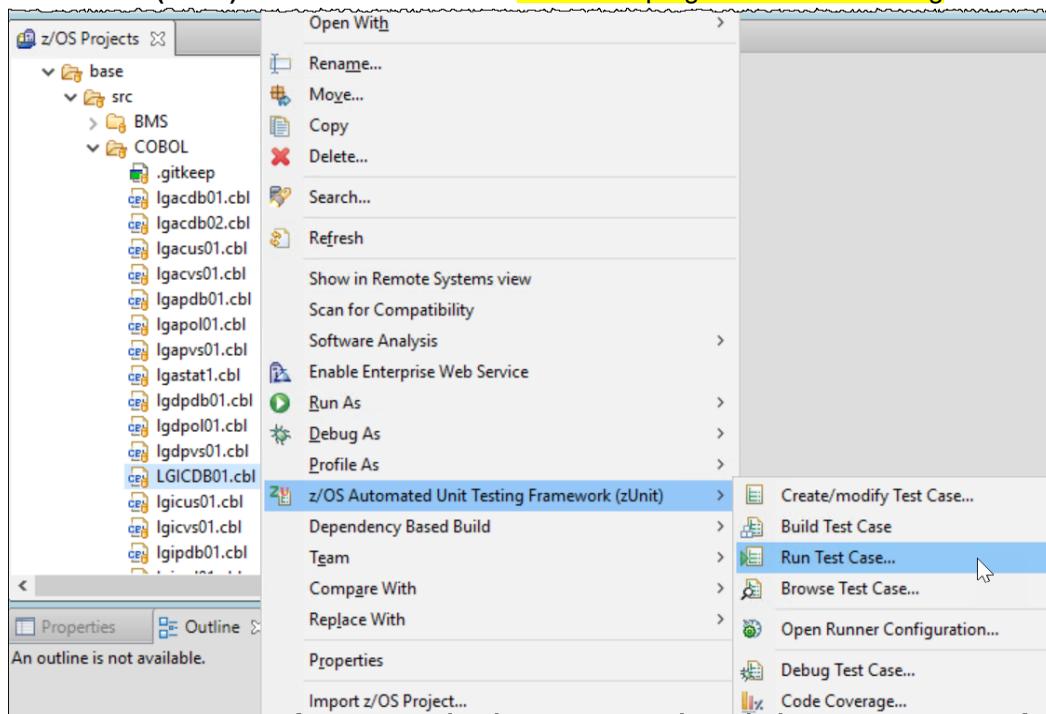
Running zUnit you will verify that the DOB is incorrect . A zUnit test case was created by other developer when the dialog was correct and the DOB date was displayed correct, For details how to create zUnit test cases you can see the zUnit labs provided.

3.1 Running the zUnit in batch

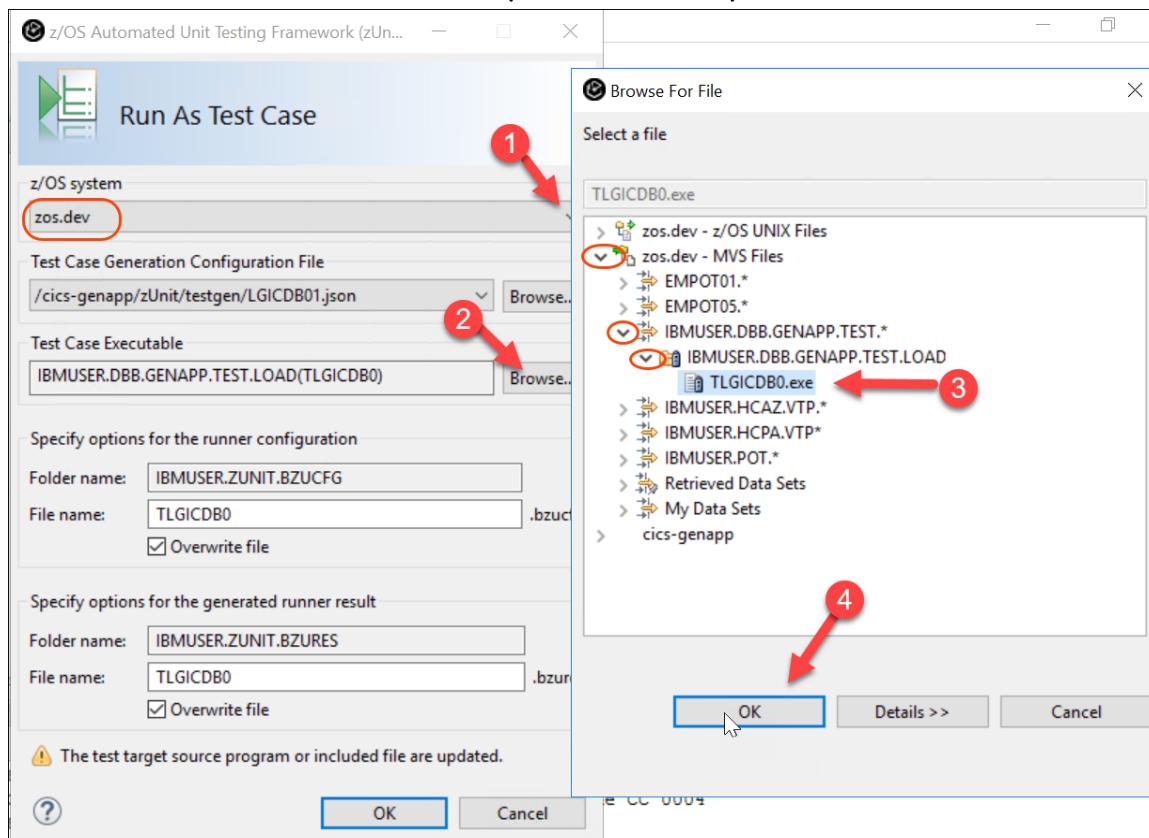
A zUnit test case is already available, and the developer will run it.

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**.

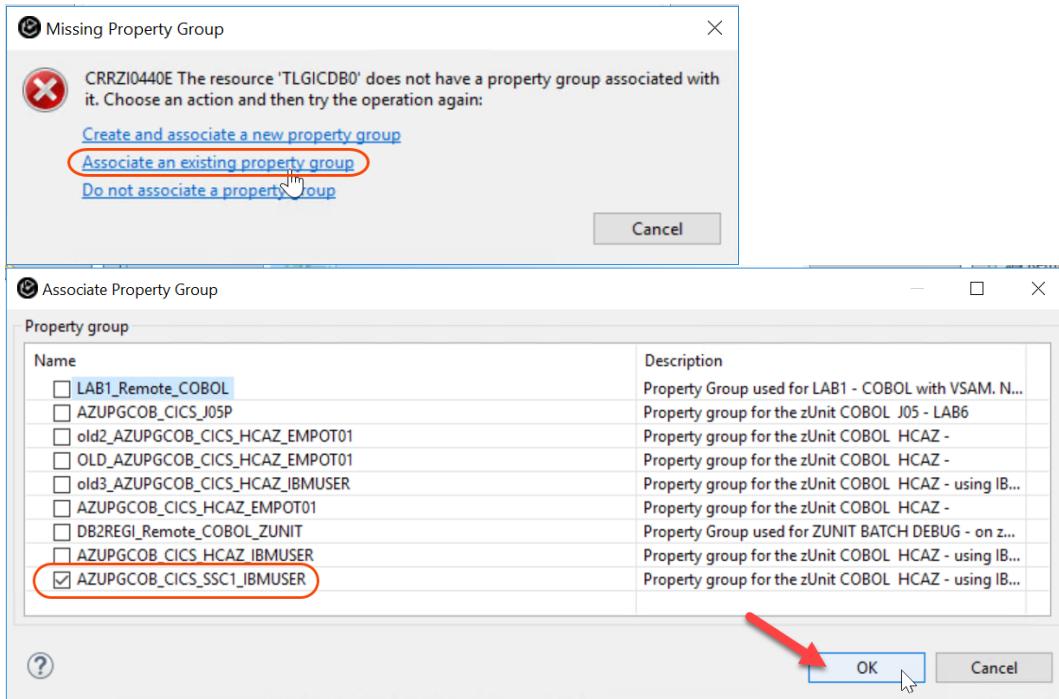
3.1.1 ➡ Using z/OS Projects view right click **LGICDB01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)** and **Run Test Case ...**This is the program that has the bug



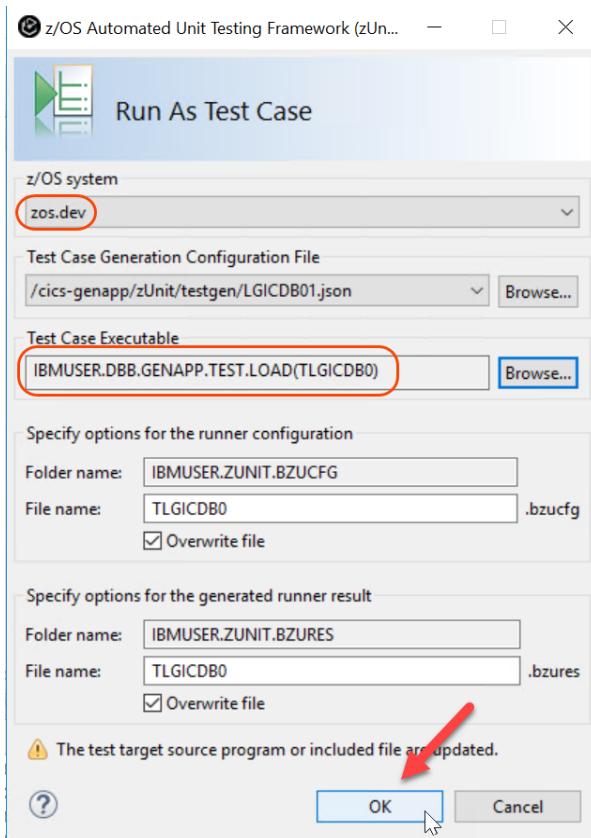
3.1.2 ➡ Select **zos.dev** for z/OS system, for **Test Case Executable** use the **Browse...** button and choose **IBMUSER.DBB.GENAPP.TEST.LOAD (TLGICDB0.exe)** and Click **OK**.



3.1.3 ► If the dialog below appears select **Associate an existing property group** point to **AZUPGCOB_CICS_SSC1_IBMUSER** and click **OK**

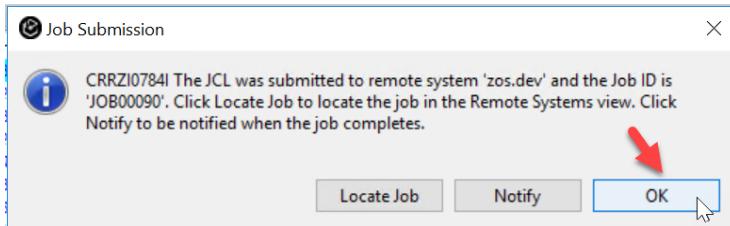


3.1.3 ► Be sure that your dialog values below match the screen below and click **OK**.



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	

A red circle highlights the **--debug** option in the list.

3.2 Verifying the zUnit Results

This Batch job can be seen at 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.

Exception details

If the result of the test is "fail" or "error", this section contains the details of the exception.

Test cases and test results

type filter text

- zUnit runner results
 - Test case TLGICDB0
 - Test TEST2 (fail)
 - Exception message number (3000), Severity (4), Component zos

Add... Remove Up Down

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
BZUPT001 FINISHED EXECUTION RC=04

A red circle highlights the 'Exception message number (3000), Severity (4), Component zos' entry, and a red arrow points to the detailed exception message on the right.

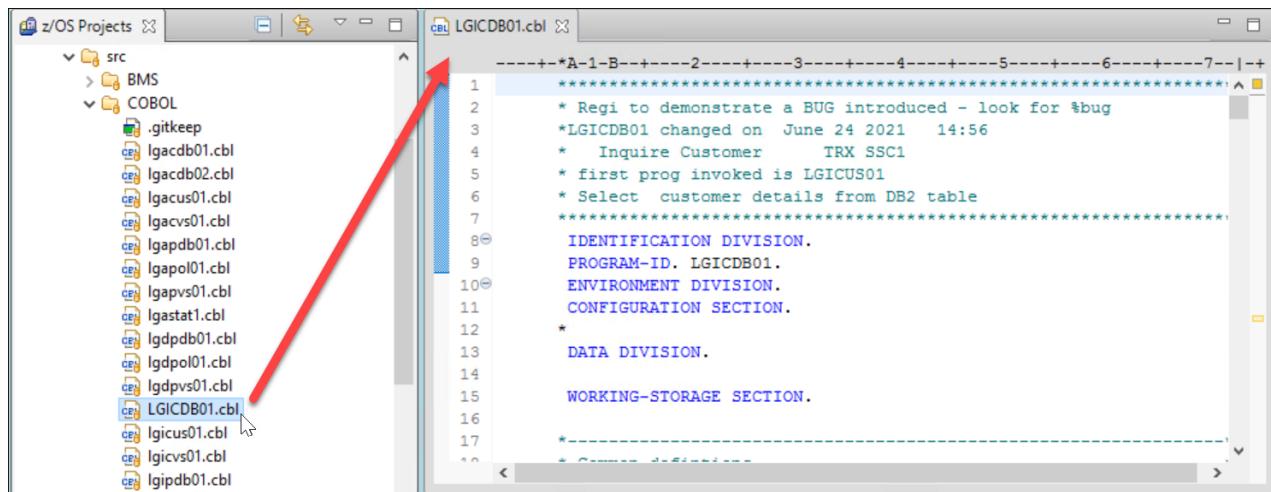
3.2.3. ► Use **Ctrl + Shift + F4** to close all opened editors.

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..

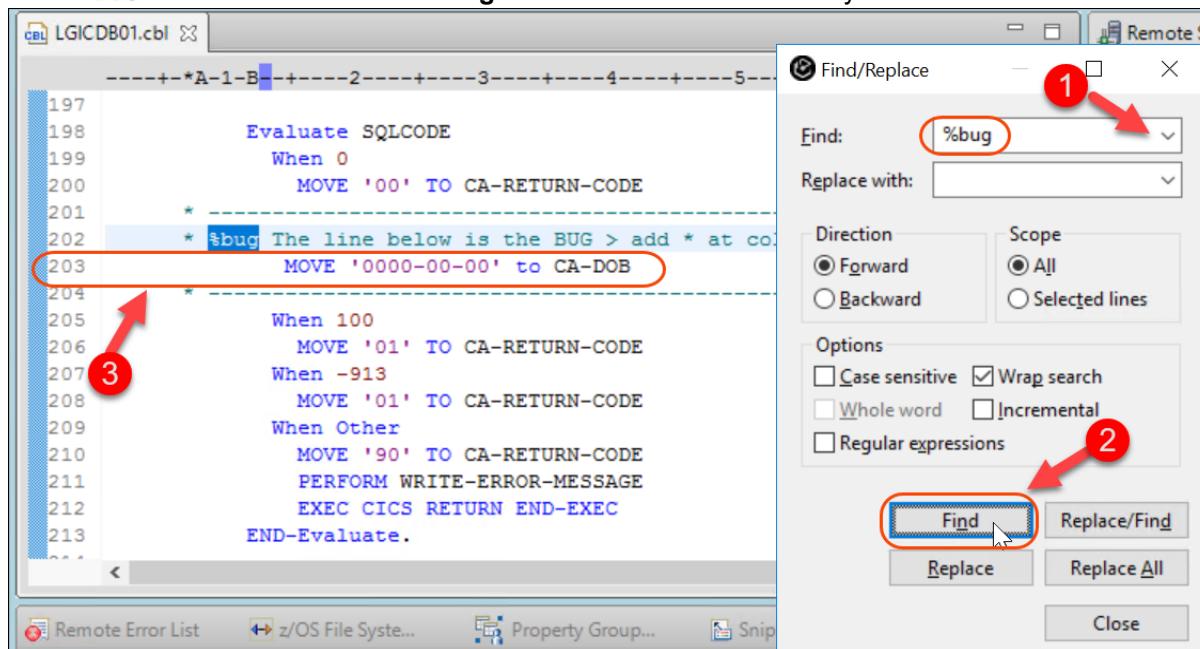
4.1 Edit and modify the code that send the message

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



The screenshot shows the z/OS Projects interface. On the left, there's a tree view of source files under 'src'. Under the 'COBOL' folder, several files are listed, including .gitkeep, lgacdb01.cbl, lgacdb02.cbl, lgacus01.cbl, lgacvs01.cbl, lgapdb01.cbl, lgapol01.cbl, lgapvs01.cbl, lgastat1.cbl, lgdpdb01.cbl, lgdpol01.cbl, lgdpvs01.cbl, lgicu01.cbl, lgicvs01.cbl, and lgipdb01.cbl. A red arrow points from the text above to the 'LGICDB01.cbl' file, indicating it is the one to be edited.

4.1.2 ► Use Ctrl + f to find the "%bug" on line 203 . You will modify the line 203..



The screenshot shows the COBOL editor with the file LGICDB01.cbl open. The code editor window displays several lines of COBOL code. Line 203 is highlighted with a red circle and labeled '3'. The code on line 203 is:

```

* %bug The line below is the BUG > add * at co
MOVE '0000-00-00' to CA-DOB

```

To the right of the editor is the 'Find/Replace' dialog box. The 'Find' field contains the text '%bug' (circled with red number 1). At the bottom of the dialog, the 'Find' button is highlighted with a red border (circled with red number 2).

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

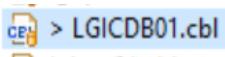
```

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

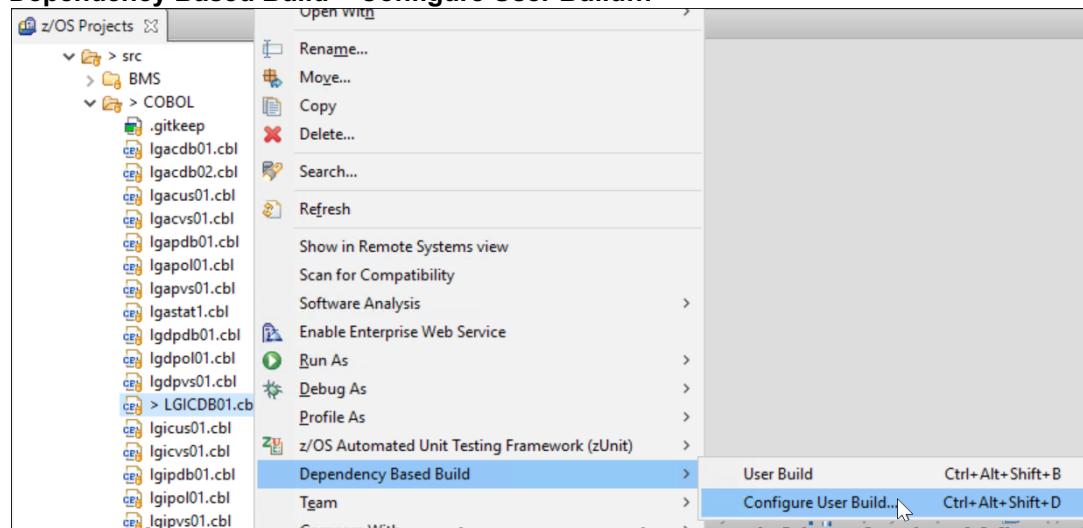


Section 5. Use IDz DBB User Build to compile/link and run the zUnit.

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

Use this configuration as the project default

[Preferences](#)

5.1.3 ► 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.

DBB User Build

Configure User Build Operation File Attributes

Specify file attribute resource information for the user build operation.

Use project traversal to compose attributes. (Recommended)
`\cics-genapp\gitattributes`

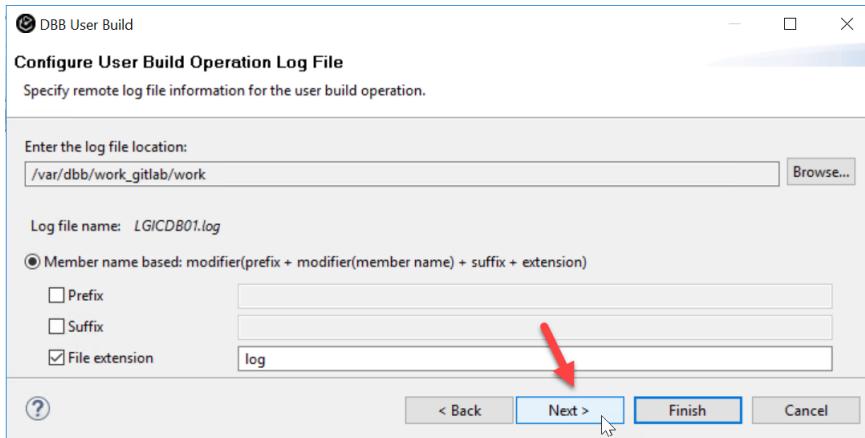
Use the project attribute file.
`C:\git_POT\dbb-pipeline\gitattributes`

Use a selected attribute file.

Select the encoding file to use:

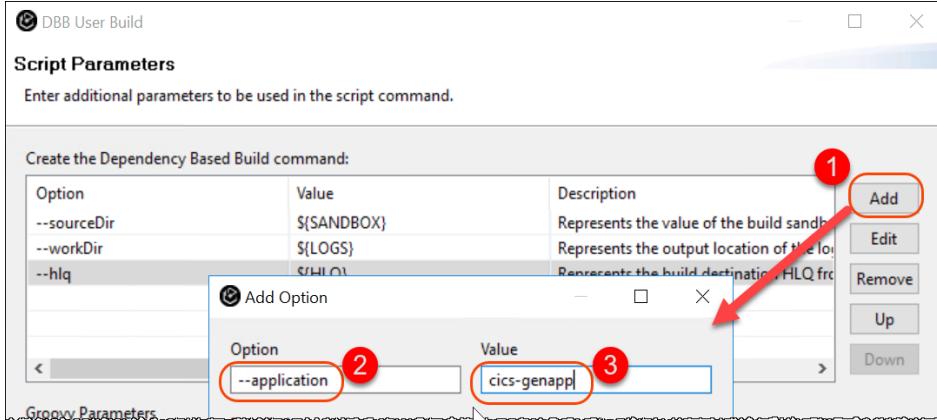
Do not use an attributes file.

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.

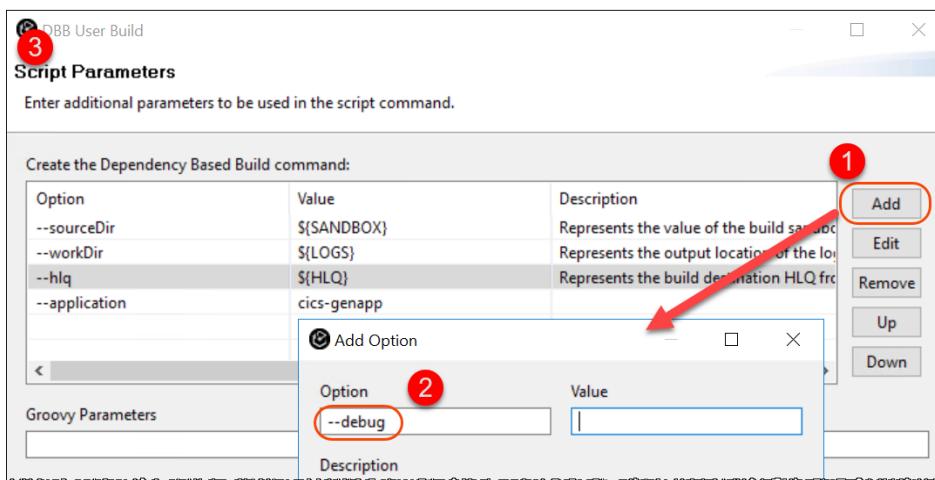
► Click **Add** to insert **--application** and **cics-genapp** (mixed case) and click **OK**



5.1.6 **In case you need to add --debug.**

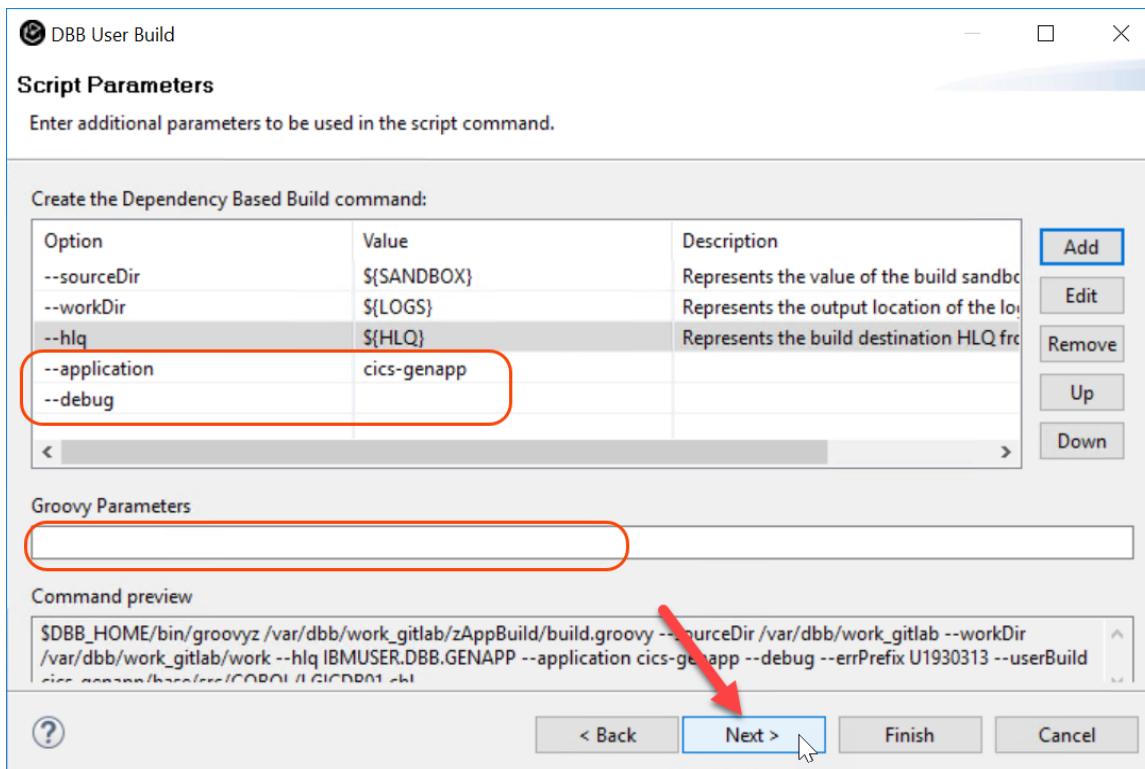
► Click **Add** to insert **--debug** and click **OK**

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

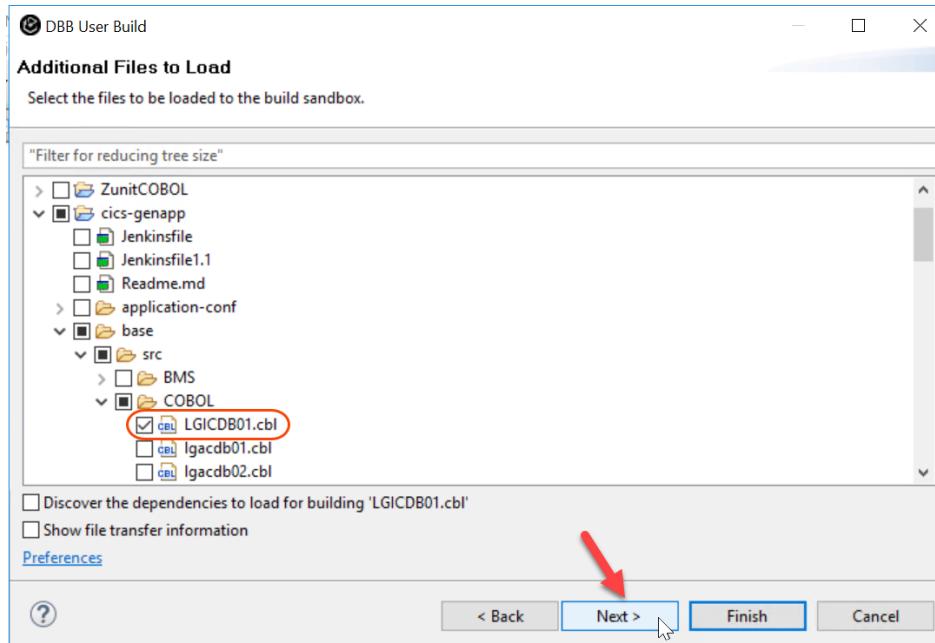
▶| Click **Next**



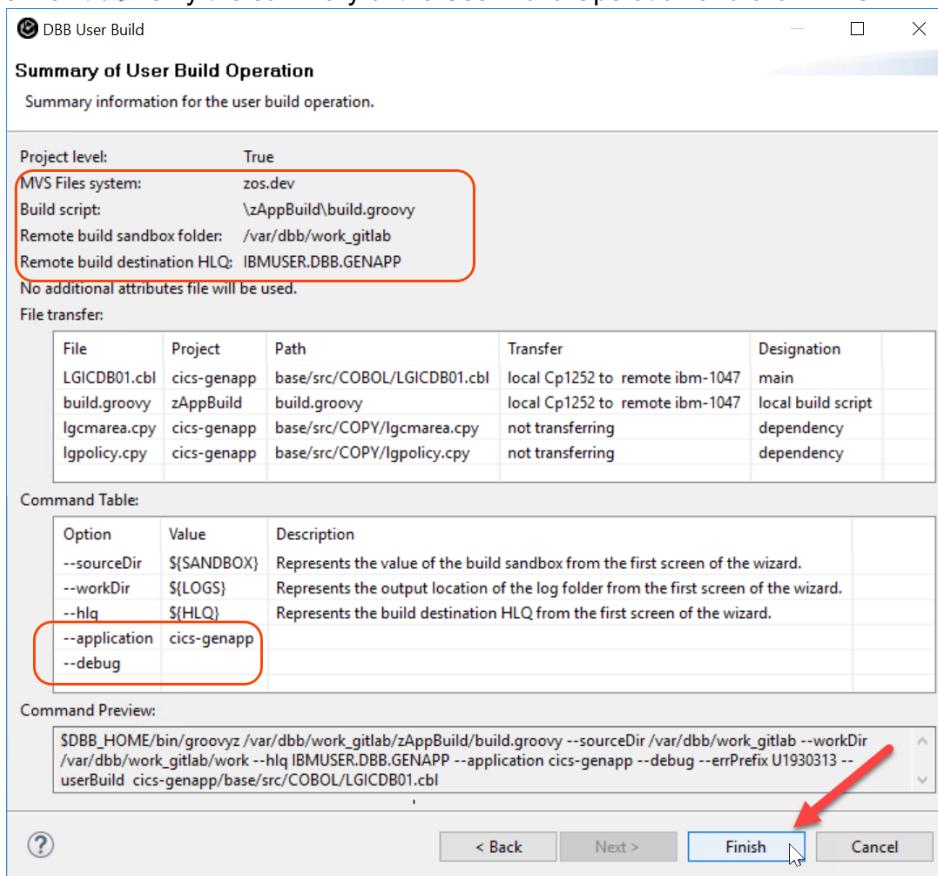
5.1.8 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

▶| Expand **cics-genapp** and **base/src/COBOL** to verify that the **LGICDB01.cbl** is selected

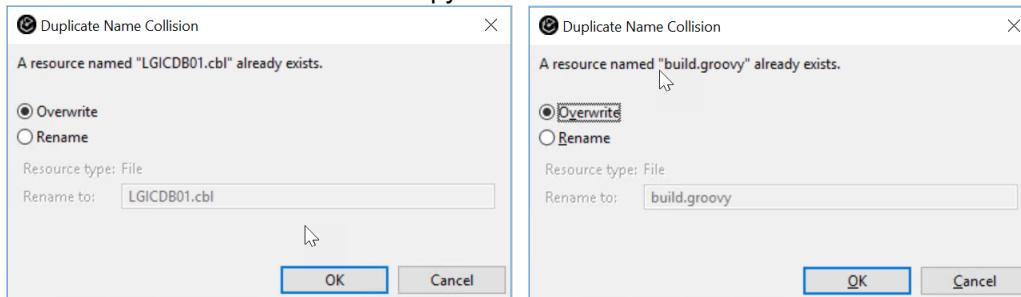
▶| Click **Next**



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

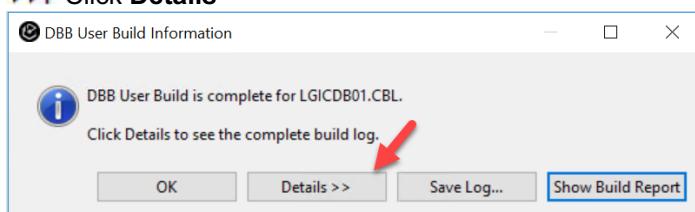


The DBB User build will be running and the messages will be shown at the Console view..

► 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:

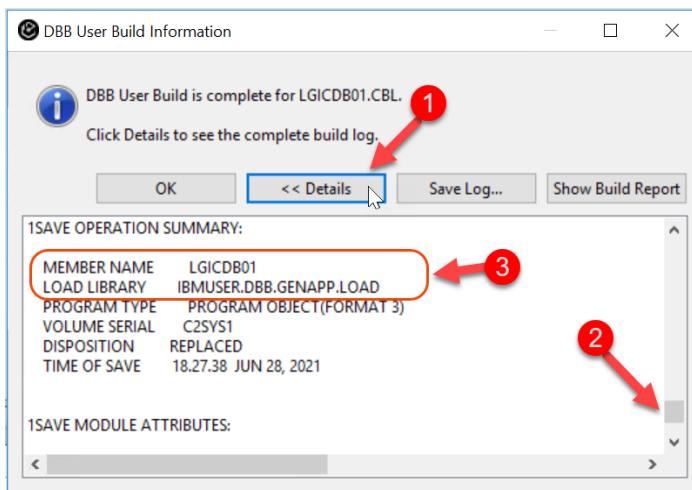
► Click **Details**



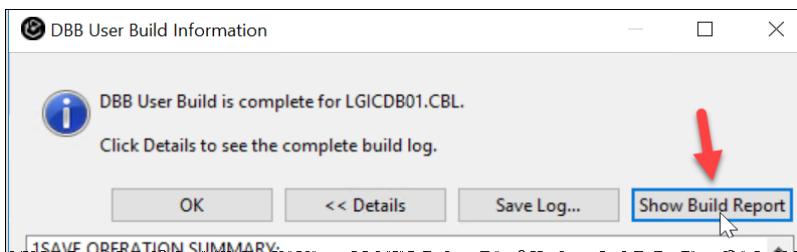
5.2 Verify the DBB User Build results

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.



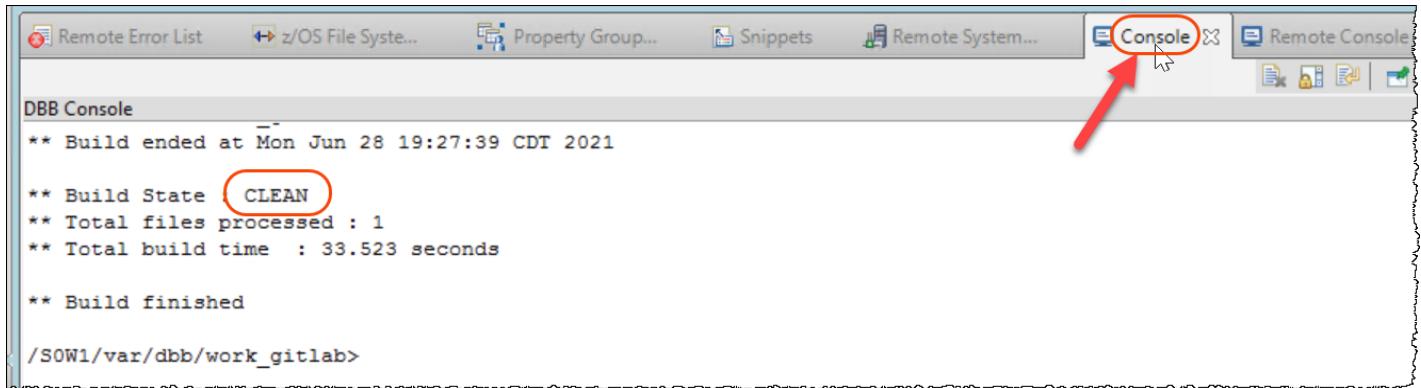
5.2.2 ➡ Click **Show Build Report** to see the report created:



	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	cics-genapp/base/src/COBOL/LGICDB01.cbl Show Dependencies	IGYCRCTL	4	IBMUSER.DB.B.GENAPP.COOL (LGICDB01)	IBMUSER.DB.B.GENAPP.DBRM (LGICDB01)	DBRM	LGICDB01.log
		IEWLINK	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:

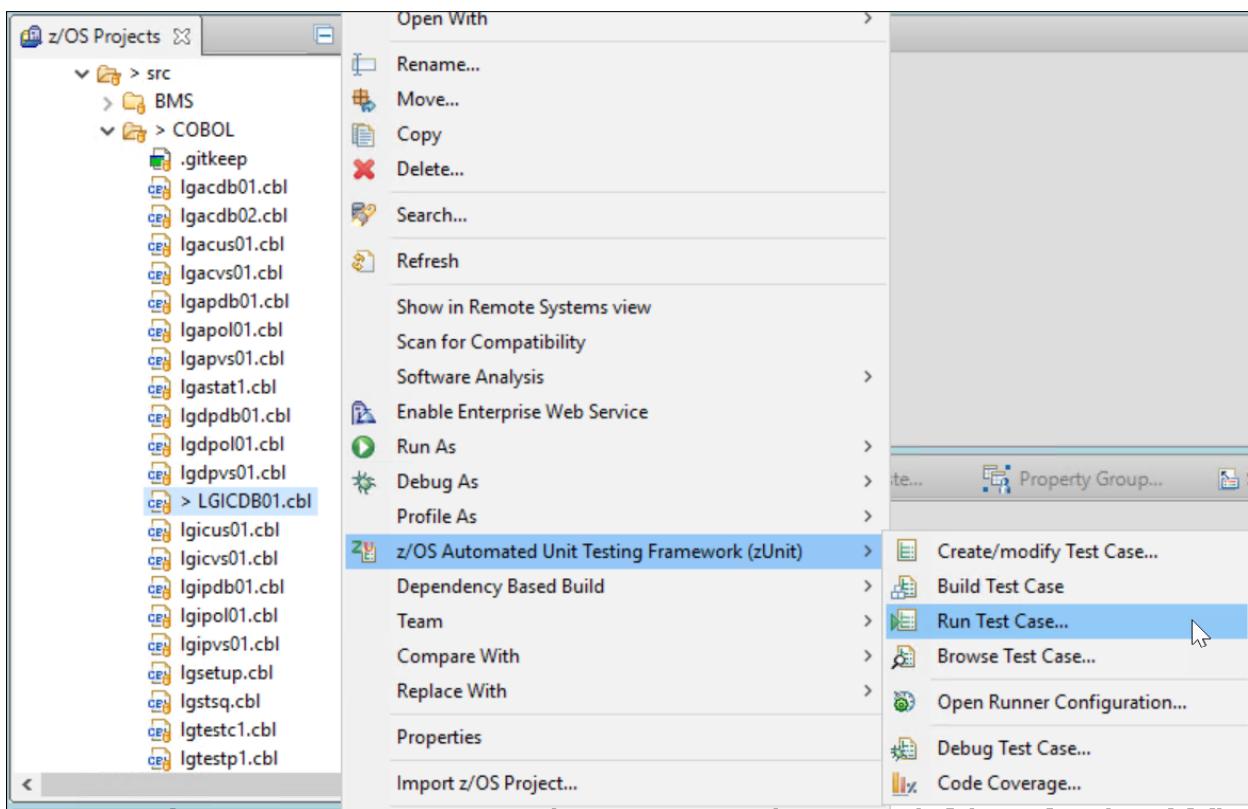
```
** Build ended at Mon Jun 28 19:27:39 CDT 2021
** Build State CLEAN
** Total files processed : 1
** Total build time : 33.523 seconds
** Build finished
/SOW1/var/dbb/work_gitlab>
```

A red arrow points to the "Console" tab in the top right corner of the window, which is highlighted with a red circle.

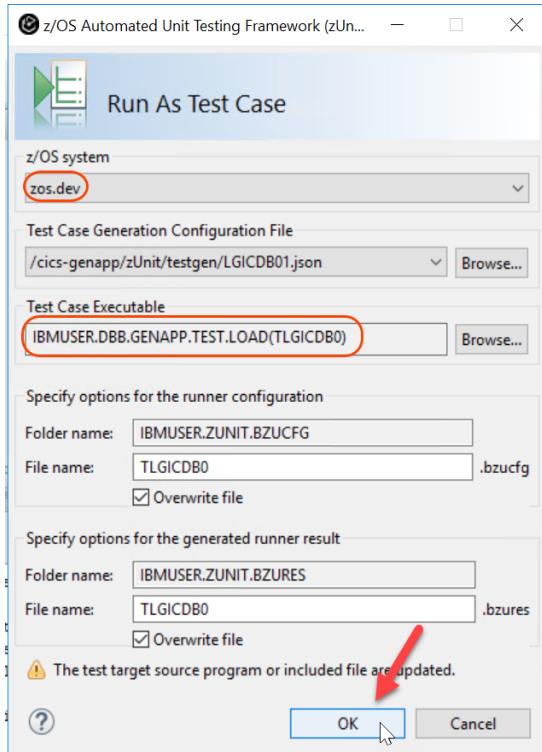
5.3 Running the zUnit again

You should now run the zUnit test case again to verify that the bug has been fixed.

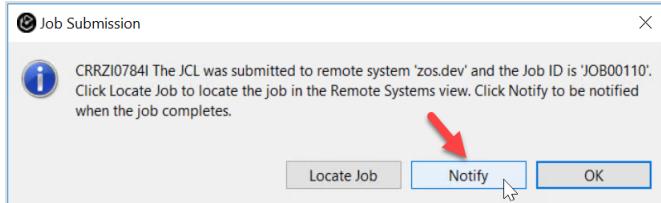
5.3.1 ► Right click on **LGICDB01.cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > . Run Test Case**



5.3.2 ➔ Be sure that the values below are in effect and click **OK**

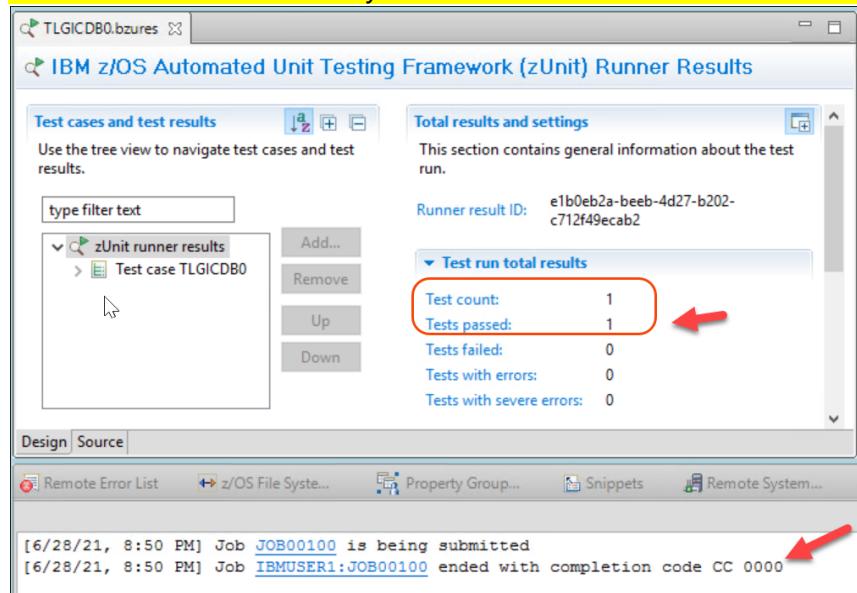


5.3.3 ➔ A Job Submission dialog opens, click **Notify** to be notified of job completion.



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

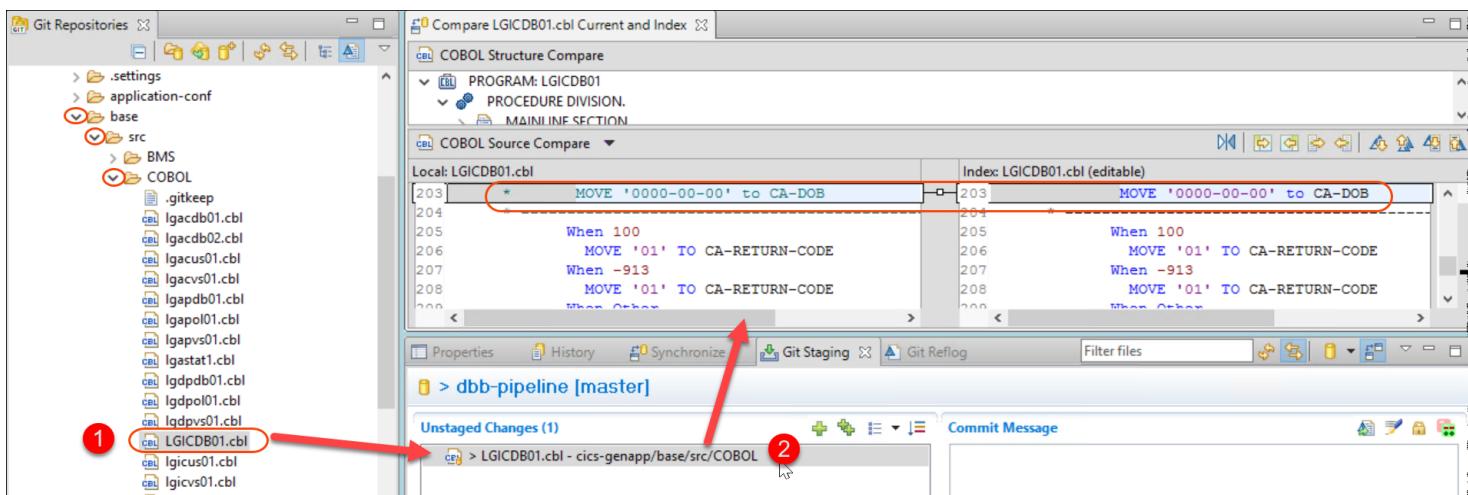


6.1.2 ►| ① Expand the **base/src/COBOL** folders and click on the program **LGICDB01.cbl** that you have modified.

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.

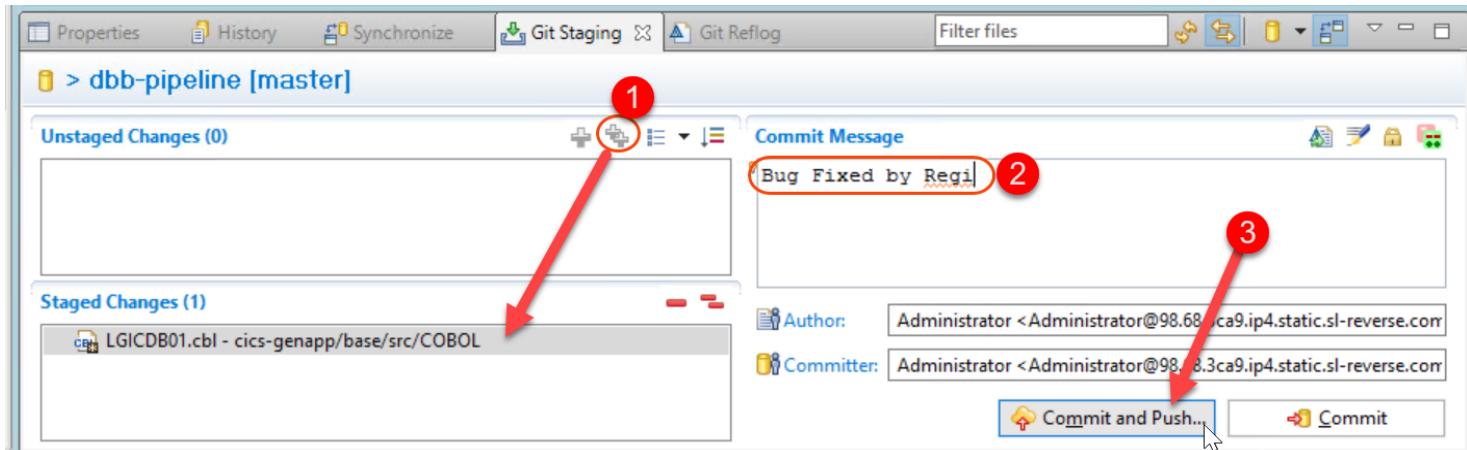


6.1.3 ►| Use **Ctrl + Shift + F4** to close the editors.

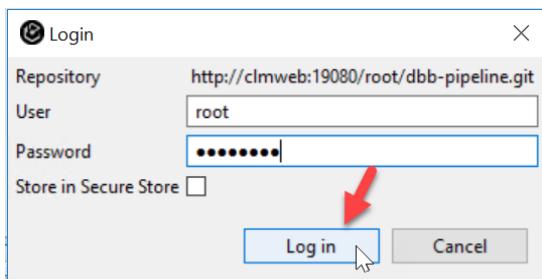
6.2 Push and Commit the changes to GitLab

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 ...**

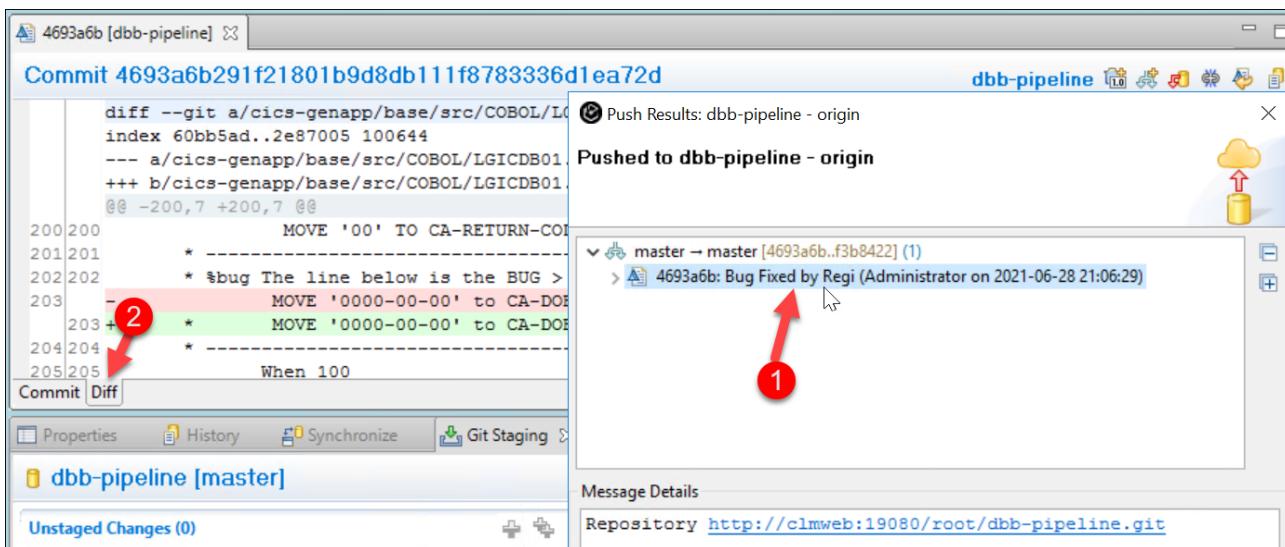


6.2.2 ►| For login credentials use **root** and password **zdtlinux** and click **Log in**



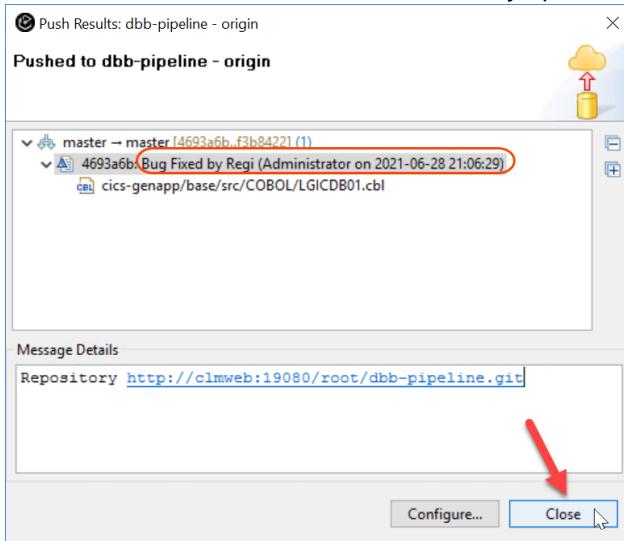
6.2.3 To see the changes you committed to Git:

- | 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).

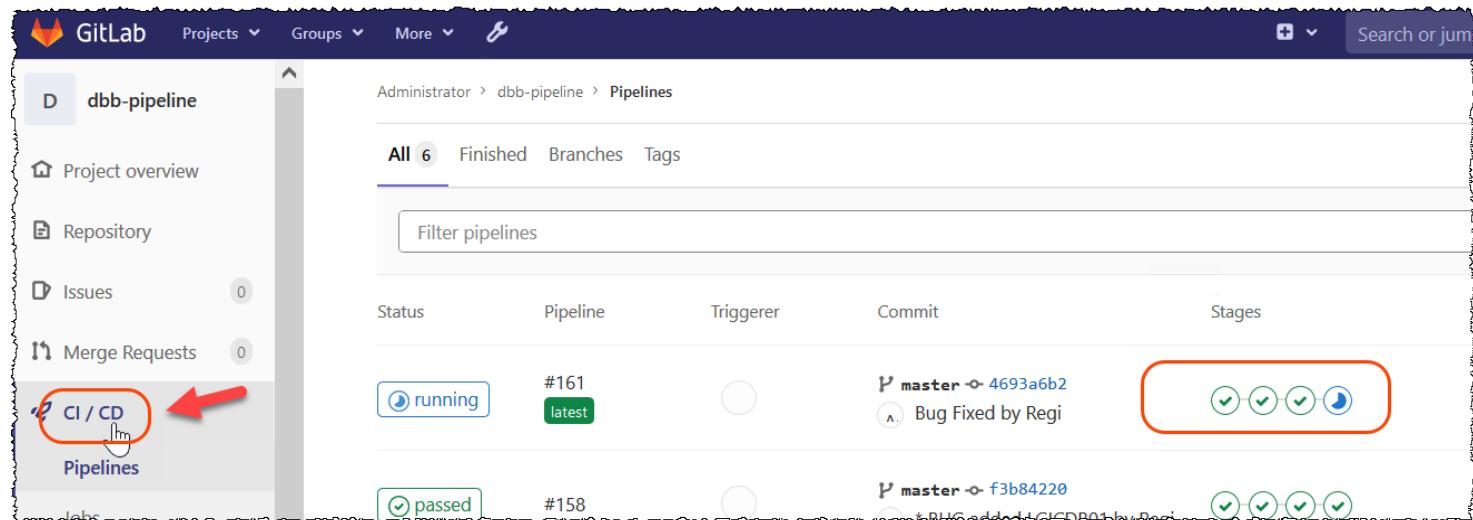
Name	Last commit	Last update
cics-genapp	Bug Fixed by Regi	5 minutes ago

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

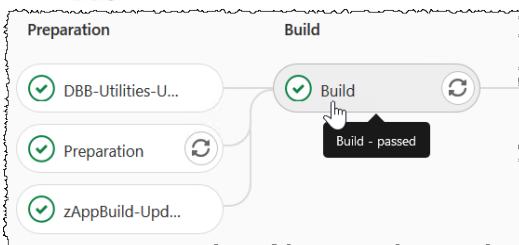


Status	Pipeline	Triggerer	Commit	Stages
 running	#161 latest		 master -> 4693a6b2  Bug Fixed by Regi	   
 passed	#158		 master -> f3b84220	   

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 SSSH

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...
9 Getting source from Git repository
10 Skipping Git repository setup
11 Skipping Git checkout
12 Skipping Git submodules setup
14 Executing "step_script" stage of the job script
15 $ echo "*REGI* HLQ ${dbbHlq} DBBHome ${dbbHome}"
16 *REGI* HLQ IBMUSER.DB8 DBBHome /var/dbb/1.1
17 $ echo "*REGI* GitLab workspace created at $CI_PROJECT_DIR"
18 *REGI* GitLab workspace created at /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline
19 $ dbbHome/bin/groovyz $dbbGroovyOpts $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
20 ** Build start at 20210628.081053.010
21 ** Repository client created for https://clmweb:11043/dbb/
22 ** Build output located at /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010
23 ** Build result created for BuildGroup:cics-genapp-161 BuildLabel:build.20210628.081053.010 at https://clmweb:11043/dbb/rest/buildResult/3048
24 ** --impactBuild option selected. Building impacted programs for application cics-genapp
25 ** Writing build list file to /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/buildList.txt
26 ** Invoking build scripts according to build order: BMS.groovy,Cobol.groovy
27 ** Invoking test scripts according to test order: ZunitConfig.groovy
28 ** Building files mapped to Cobol.groovy script
29 *** Building file cics-genapp/base/src/COBOL/LGICDB01.cbl
30 ** Building files mapped to ZunitConfig.groovy script
31 *** Building file cics-genapp/zUnit/testcfg/LGICDB01.bzucfg
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.xls; done;
12 Process zUnit ./LGICDB01.zunit.report.log
13 Job succeeded

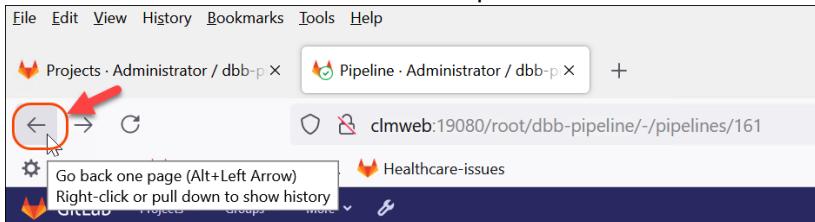
```

Section 8. Verify the GitLab CI Package and Deploy to UCD and test the CICS transaction again using 3270

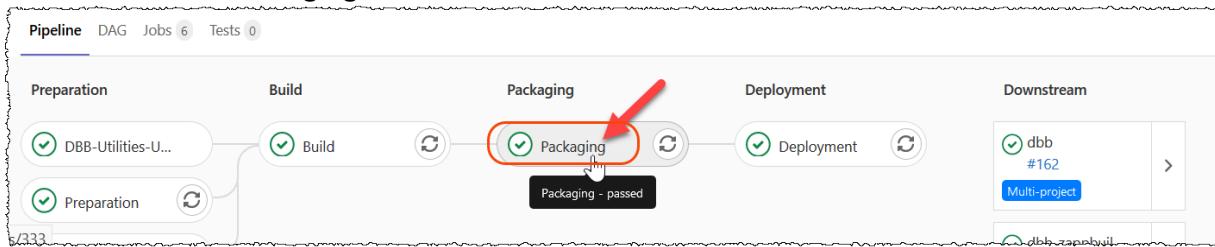
You will verify the results of the second stage (deploy using UCD) and also test the deployed application using CICS.

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
6  Preparing environment
7  Running on S0W1.DAL-EBIS.IHOST.COM via W-30830...
8
9  Getting source from Git repository
10 Skipping Git repository setup
11 Skipping Git checkout
12 Skipping Git submodules setup
13
14 Executing "step_script" stage of the job script
15 $ echo "*REGI* UCDBUZTOOL_PATH ${ucdBuztool}"
16 *REGI* UCDBUZTOOL_PATH /apps/ucd/v7/bin/buztool.sh
17 $ cd ${CI_PROJECT_DIR}/BUILD-$CI_PIPELINE_ID
18 $ export buildListFile=`find . -name "buildList.txt"`
19 $ 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
20 $ 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
21 Move the binaries created by build to UCD to be deployed
22 *REGI* Push to UCD Codestation
23 ** Create version start at 20210628.081314.013
24 ** Properties at startup:
25   component -> GenApp
26   startTime -> 20210628.081314.013
27   workDir -> /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010
28   preview -> false
29   buztoolPath -> /apps/ucd/v7/bin/buztool.sh
30 ** Read build report data from /var/dbb/gitlab/RVUKgUne/0/root/dbb-pipeline/BUILD-161/build.20210628.081053.010/BuildReport.json
31 ** Find deployable outputs in the build report

```

```

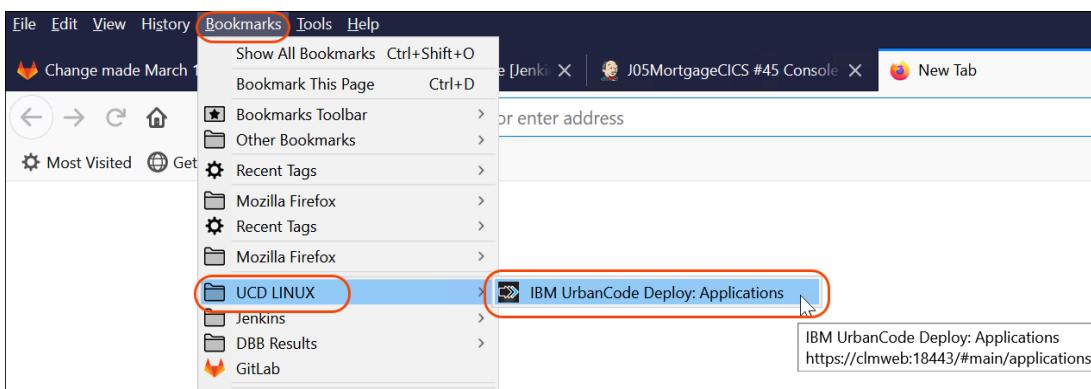
40 zOS toolkit config : /apps/ucd/v7/ (7.0.2.0,20190131-0837)
41 zOS toolkit binary : /apps/ucd/v7/ (7.0.2.0,20190131-0837)
42 zOS toolkit data set : BUZV7 (7.0.2.0,20190131-0837)
43 Reading parameters:
44 ....Command : createzosversion
45 ....Component : GenApp
46 ....Generate version name : 20210628-201345
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 ....IBMUUSER.DBB.GENAPP.DBRM.bin
57 ....IBMUUSER.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
74 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	Tags		
AccountMgmtCICS	20180627-1257160725	MVSCOMPONENT	
CICS	NO VERSION		
GenApp	20210628-201345	MVSCOMPONENT	

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

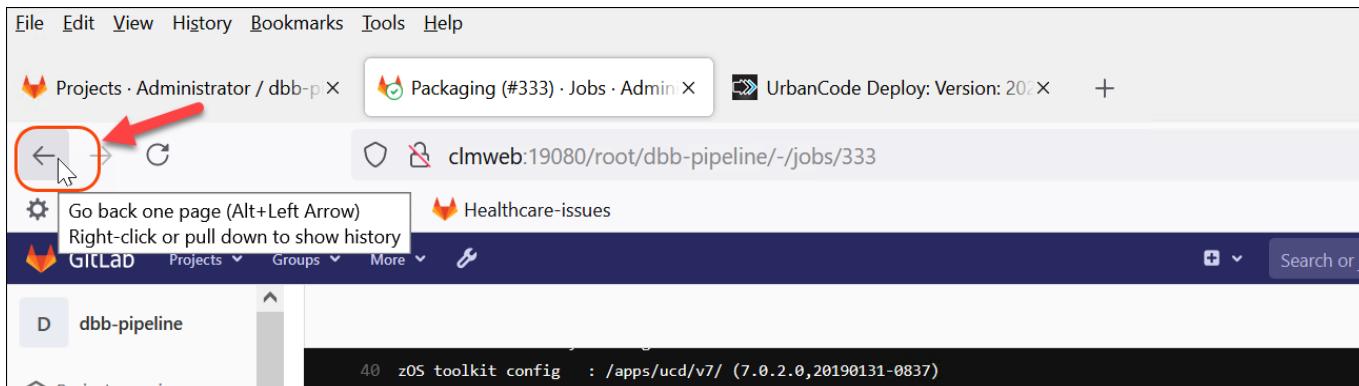
Total add: 2 members in 2 data sets

Expand All Co

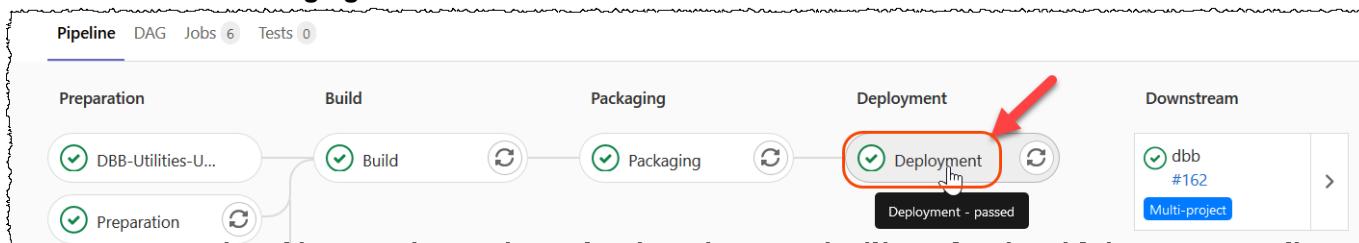
Name	Artifact Type	Deploy Type	Inputs	Last Modified	Properties
Filter	Filter	Filter	Filter		
BMUSER.DB.BGENAPP.DBRM	[PDS,ADD]				
LGICDB01		DBRM	cics-genapp/base/src/COBOL/LGICDB01.cbl		buildcommand=IGYCRCTL builddoptions=LIB,CICS,SQL
BMUSER.DB.BGENAPP.LOAD	[PDS,ADD]				
LGICDB01		LOAD	cics-genapp/base/src/COBOL/LGICDB01.cbl		buildcommand=IEWBLINK builddoptions=XREF,LIST,RENT

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:



- 8.3.2 ► Click on **Packaging**



- 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 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 $ cd ${CI_PROJECT_DIR}/BUILD-$CI_PIPELINE_ID
13 $ export buildListFile='find . -name "buildList.txt"'
14 $ 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
15 $ 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
16 *REGI* Deploy Appl GenApp_CICS Process Deploy_to_CICS to Environment :DEV on CICSTS53
17 ** Deploying component versions: GenApp:latest
18 *** Starting deployment process 'Deploy_to_CICS' of application 'GenApp_CICS' in environment 'DEV'
19 *** Follow Process Request: https://clmweb:18443/#applicationProcessRequest/17a55554-6e1b-43b7-ecbd-385707b4df5f
20 **** The deployment result is SUCCEEDED. See the UrbanCode Deploy deployment logs for details.
21 ** Build finished
22 Job succeeded

```

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 with the 'Applications' tab selected. A list of applications is displayed in a table format. The application 'GenApp_CICS' is highlighted with a red circle and a mouse cursor is hovering over its row.

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 PDTOOLS demo
DemoDeployment		
GenApp_CICS		Deploy Genap Application

8.4.2 ► Click on the **History** tab

The screenshot shows the 'Application: GenApp_CICS' page with the 'History' tab selected. The tab bar also includes 'Environments', 'Configuration', 'Components', 'Blueprints', 'Snapshots', 'Processes', 'Calendar', and 'Changes'. A red arrow points to the 'History' tab.

8.4.3 ► Click on the **View Request** entry that is related to your deploy (check the date/time)

The screenshot shows the 'Application: GenApp_CICS' page with the 'History' tab selected. A table lists deployment processes. An arrow points to the 'View Request' link in the 'Actions' column of the first row, which corresponds to the 'Deploy_to_CICS' process.

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
v Deploy GenApp to CICS (GenApp 20210628-201345)				
1. Download Artifacts for zOS		9:14:51 PM	0:04:10	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 -DBBG 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 PLANMNGT OFF
271 PLANMNGTSCOPE STATIC
272 CONCURRENTACCESSRESOLUTION
273 EXTENDEDINDICATOR
274 PATH
275 DSNT275I -DBBG DSNTBCM2 BIND OPTIONS FOR
276 PACKAGE = DALLASB.GENASA2.LGICDB01.()
277 QUERYACCELERATION
278 GETACCELARCHEIVE
279 DSNT232I -DBBG 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**

The screenshot shows a pipeline named 'GenApp' with several steps listed:

- Deploy GenApp to CICS (GenApp 20210628-201345)
- Download Artifacts for zOS
- Deploy Data Sets
- GenBndCard
- Generate Program List
- Bind Package & Plan
- NEWCOPY Programs** (Status: 9:18:15 PM)

A red arrow points to the 'Output Log' button at the bottom right of the pipeline card.

8.4.8 Notice that **LGICDB01** program was deployed to CICS Dev environment. and a CICS NEWCOPY was succeeded.

The 'Output Log' window displays deployment logs:

```

Working Directory /apps/ucd/v7/var/work/GenApp
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
=====

2021/06/29 01:19:20.028 GMT BUZCP00061 Connected to "10.1.1.2:1490". CICS TS version: 050300.
2021/06/29 01:19:25.633 GMT BUZCP00371 Perform NEWCOPY Operation.
2021/06/29 01:19:26.277 GMT BUZCP00241 NEWCOPY PROGRAM "LGICDB01" succeeded. (highlighted)
2021/06/29 01:19:26.489 GMT BUZCP00291 Summary: 1 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.

=====
Post Processing Script Execution Console Output:

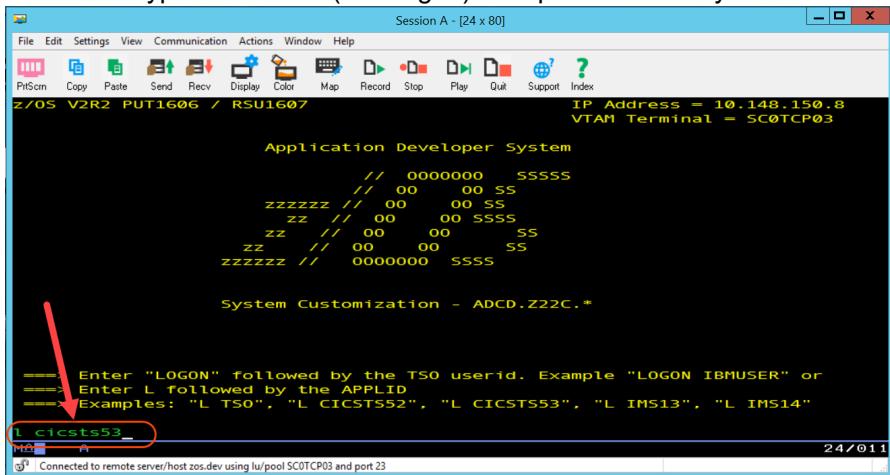
```

8.5 Testing the CICS code deployed to Dev environment

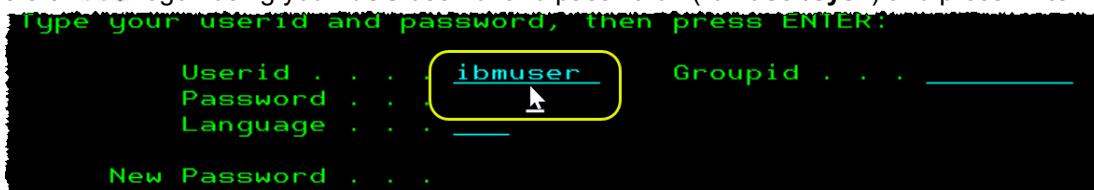
8.5.1 ► Double click on the **3270 terminal emulator** that is on the Windows desktop



8.5.2 ► Type **I cicsts53.** (I as logon) and press **Enter** key.



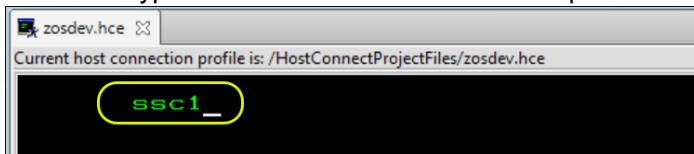
8.5.3 ► Logon using your z/OS user id and password (**ibmuser/sys1**) and press **Enter**



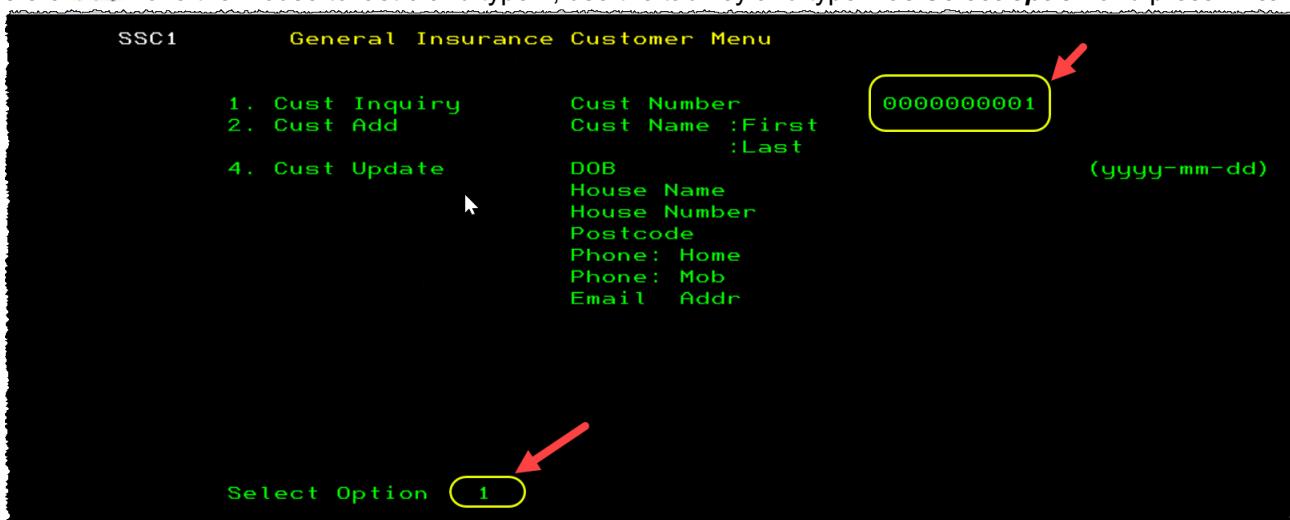
8.5.4 The sign-on message is displayed



8.5.5 ► Type the CICS transaction **ssc1** and press the **Enter** key.



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**





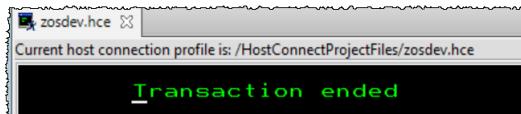
8.5.7 ➡ The data below is retrieved from a DB2 table .
Notice that **DOB** (*Date Of Birthday*) is now correct. **The bug is fixed.**

```
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
                                Phone: Mob
                                Email Addr
                                34
                                PI10100
                                01962 811234
                                07799 123456
                                A.Pandy@beebhouse.com

Select Option  1
```

8.5.8 ➡ Press **F3** to end the application.



Congratulations! You have completed the Lab 10. .

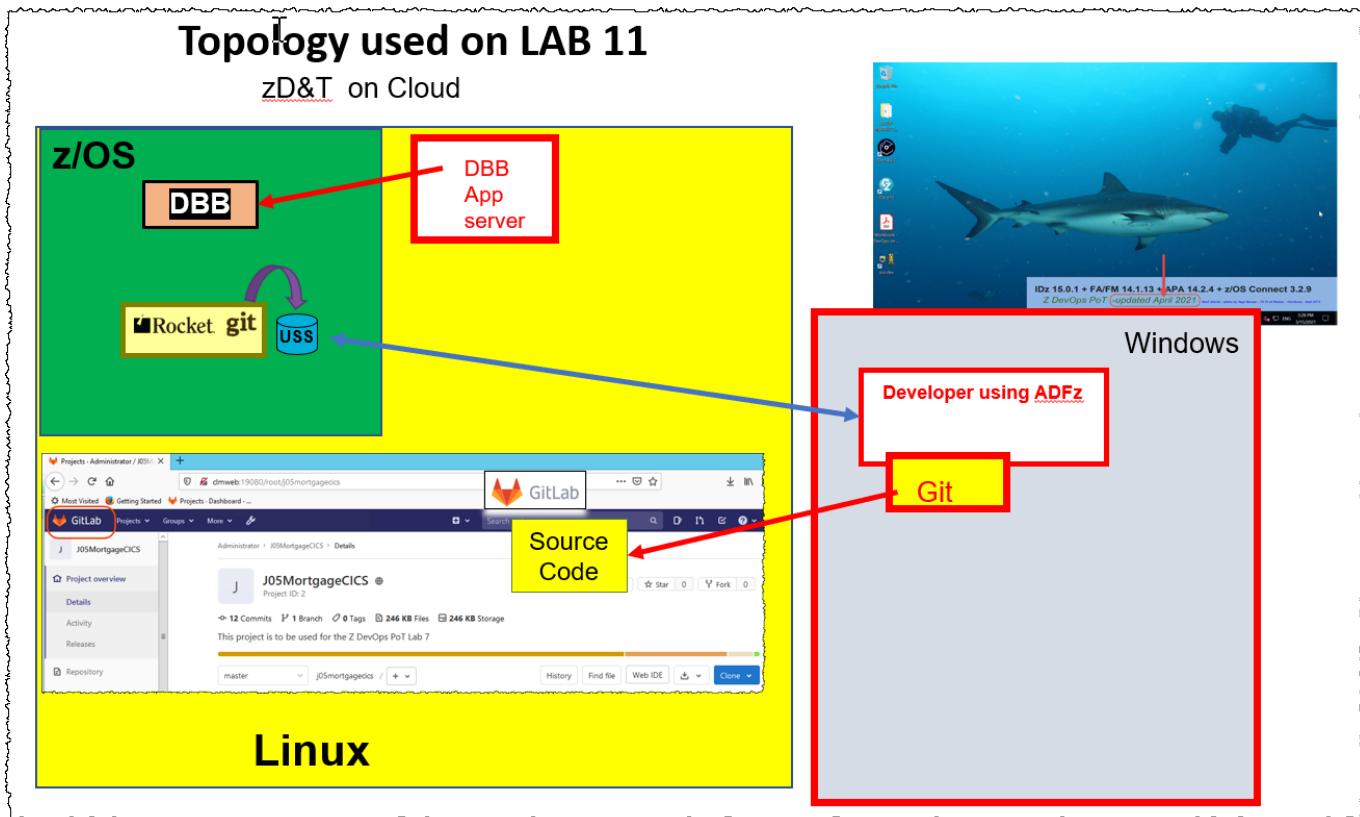
LAB 7 – (OPTIONAL) Migrating partitioned data sets (PDS) to a distributed Git repository. (50 minutes)

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**

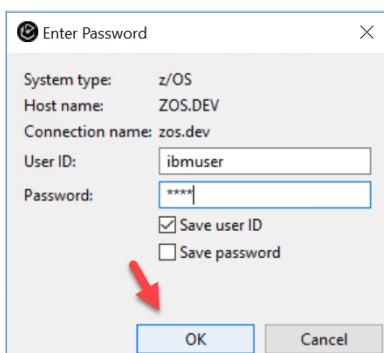
IMPORTANT -> This icon will start an IDz workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.



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

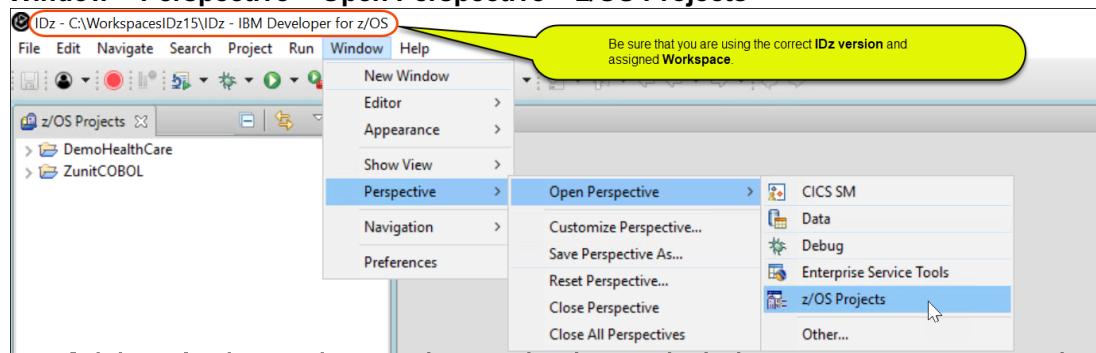
1.1.3 ► Right click on **zos.dev** and select **Connect**.

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

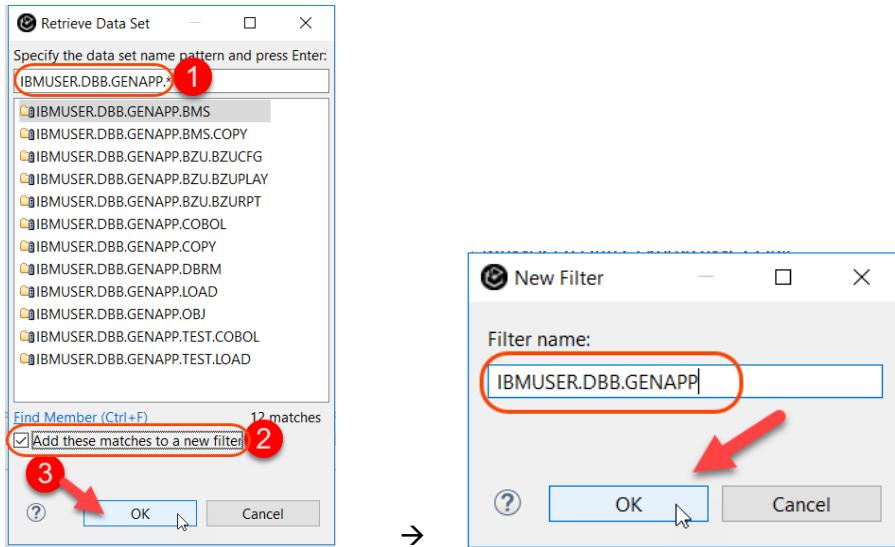


1.2 Verify the Z/OS datasets to be migrated

The application to be migrated is the GENAPP application and the source code is on the HLQ named **IBMUSER.DB.BGENAPP.***. 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.DB.B.GENAPP.*** and press **Enter**. Select **Add these matches to a new filter type IBMUSER.DB.B.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.

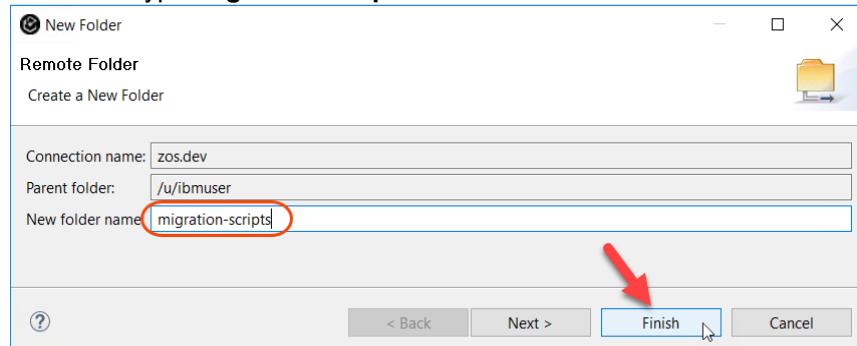
► You can expand those PDS to see the content if you want.

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.

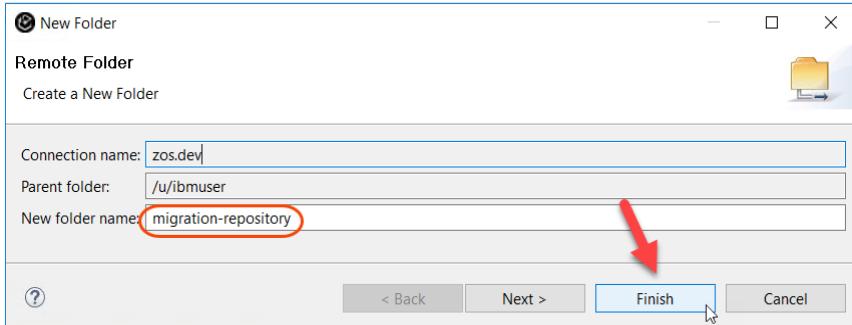
1.3.1 To create folders on ZFs. ► Expand the **z/OS Unix Files** and right click **My Home > New > Folder**

1.3.2 ► Type **migration-scripts** and click **Finish**

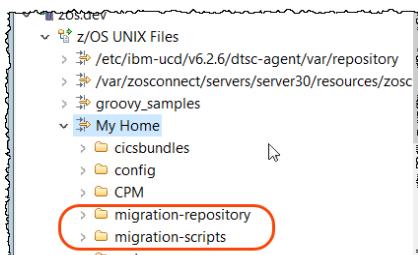


1.3.3 ► Repeat for the folder **migration-repository** and click **Finish**

IMPORTANT → Be sure that you have no spaces after the folder name.



1.3.4 The folders are created under **My Home (u/ibmuser)**

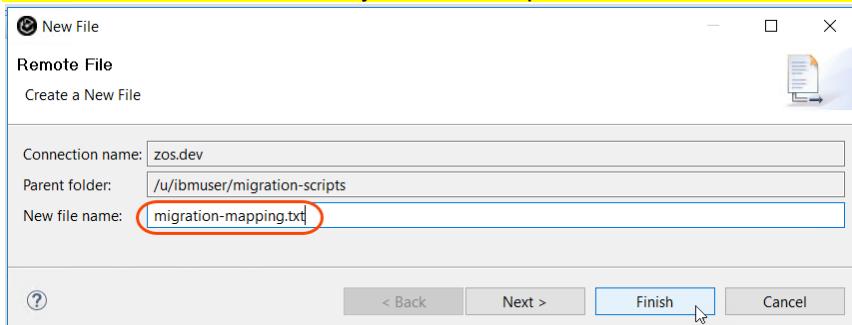


1.3.5 To create the migration scripts..

► 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.

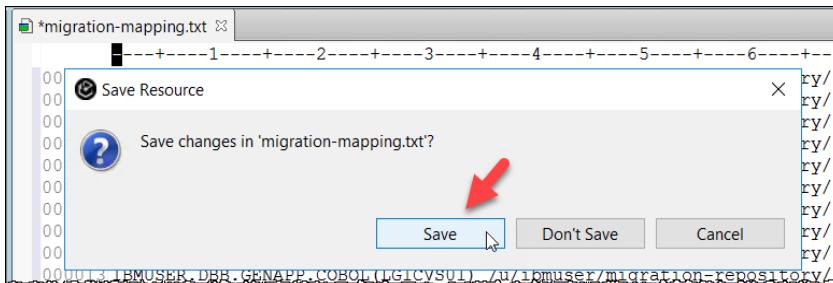
Notice that for each PDS member you specify the location where the code will be moved to in the USS directory named **migration-repository**.

```
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/lgapcvs01.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/lgdadb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGDPOL01) /u/ibmuser/migration-repository/cobol/lgdpol01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGDPVS01) /u/ibmuser/migration-repository/cobol/lgdapvs01.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(LGUICDB01) /u/ibmuser/migration-repository/cobol/lguicdb01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUICUS01) /u/ibmuser/migration-repository/cobol/lguicus01.cbl
IBMUSER.DBB.GENAPP.COBOL(LGUICVS01) /u/ibmuser/migration-repository/cobol/lguicvs01.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
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 (LGDPDB01) /u/ibmuser/migration-repository/cobol/lgdadb01.cbl
000009 IBMUSER,DBB,GENAPP,COBOL (LGDPOL01) /u/ibmuser/migration-repository/cobol/lgdapol01.cbl
000010 IBMUSER,DBB,GENAPP,COBOL (LGDPVS01) /u/ibmuser/migration-repository/cobol/lgdavps01.cbl
000011 IBMUSER,DBB,GENAPP,COBOL (LGICDB01) /u/ibmuser/migration-repository/cobol/lgidb01.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 (LGUPDB01) /u/ibmuser/migration-repository/cobol/lgupdb01.cbl
000023 IBMUSER,DBB,GENAPP,COBOL (LGUPOL01) /u/ibmuser/migration-repository/cobol/lgupo101.cbl
000024 IBMUSER,DBB,GENAPP,COBOL (LGUPVS01) /u/ibmuser/migration-repository/cobol/lgupvs01.cbl
000025 IBMUSER,DBB,GENAPP,COPY (LGCMAREA) /u/ibmuser/migration-repository/copy/lgcmarea.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/adfz/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**

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

2.1.4 See the message displayed:

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`

►| `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

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

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
```

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/*
```

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
```

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/lgapol01.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/lgdadb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9872 Aug 13 10:08 cobol/lgdpol01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      6671 Aug 13 10:08 cobol/lgdpsv01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      10875 Aug 13 10:08 cobol/lgicab01.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/lgipol01.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/igucdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9281 Aug 13 10:10 cobol/igucus01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      6812 Aug 13 10:10 cobol/igucvs01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      24889 Aug 13 10:10 cobol/igupdb01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      11048 Aug 13 10:10 cobol/igupol01.cbl
t IBM-1047 T=on -rw-r--r-- 1 IBMUSER SYS1      9855 Aug 13 10:10 cobol/igupvs01.cbl
/u/ibmuser/migration-repository>
```

2.3.4 Use the git status command, to check the current state of the repository.

```
▶▶ git status
```

2.4 Finishing migration process

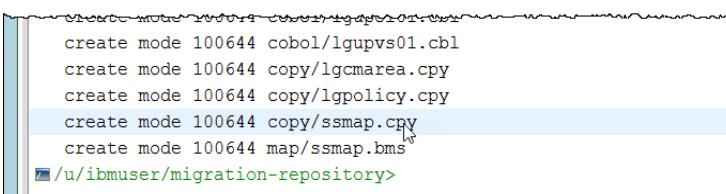
Before pushing to the Git server, you need to add the newly copied files to the local Rocket git and also commit the changes.

2.4.1 Add the newly copied files to the local Git repository database by navigating to the local Git repository directory and perform a Git 'add'.

```
▶▶ git add .
```

2.4.2 Commit the changes specifying an appropriate commit message.

```
▶▶ git commit -m "Migrate source libraries to Git"
```



```
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.

A terminal window with the command `git status` highlighted by a red oval. The output shows the repository is clean: "On branch master" and "nothing to commit, working tree clean". A red arrow points to the word "clean".

```
/u/ibmuser/migration-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)

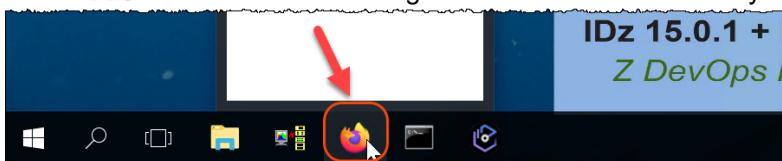
When all of the source files have been copied to the local Git repository, you are ready to push the files to the Git server.

In our Lab we will create a new Gitlab repository and push the migrated code to this newly created Git server repository.

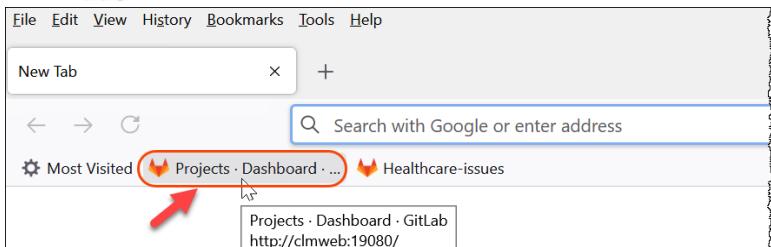
3.1 Creating a new Gitlab repository

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

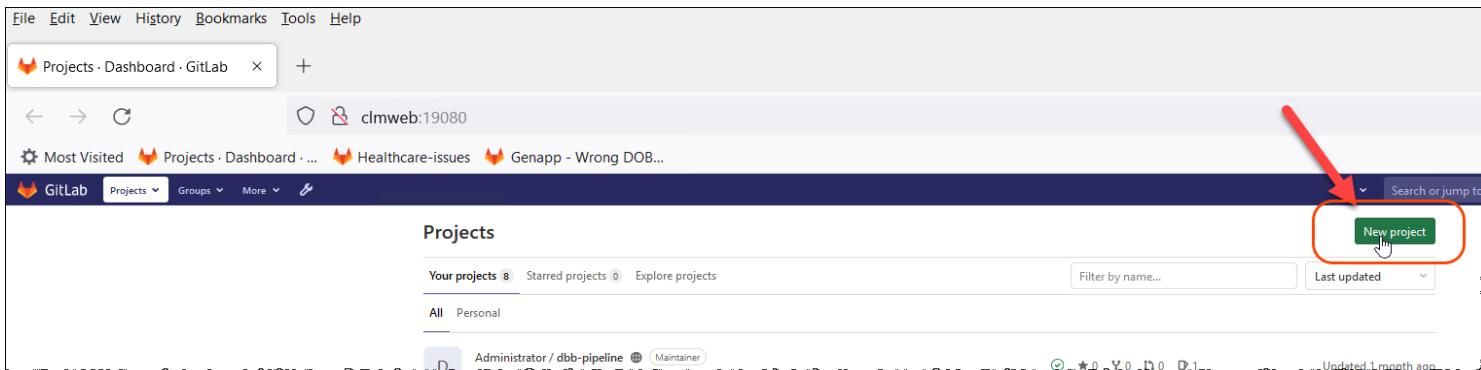


3.1.2 Click on this bookmark below to start **GitLab**. The URL used is <http://clmweb:19080/>



3.1.3 If ask for credentials use **root** and password: **zdtlinux**

3.1.4 ➡ To create a new repository click **New project**



3.1.5 ➡ Type **Migrated from Z** as Project name and click **Create project**

A screenshot of the 'New project' creation form. On the left, there's a sidebar with information about projects and tips. The main form has three tabs: 'Blank project' (selected), 'Create from template', and 'Import project'. The 'Project name' field is highlighted with a red box and contains the text 'Migrated from Z' (marked with a red circle 1). Below it, the 'Project URL' is set to 'http://clmweb:19080/ root' and the 'Project slug' is 'migrated-from-z'. There's a note about creating a group and a 'Project description (optional)' section. Under 'Visibility Level', 'Private' is selected (marked with a red circle 2). At the bottom, there's a 'Create project' button with a red arrow pointing to it (marked with a red circle 2).

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 | New file | Add README | Add LICENSE | Add CHANGELOG | Add CONTRIBUTING

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

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

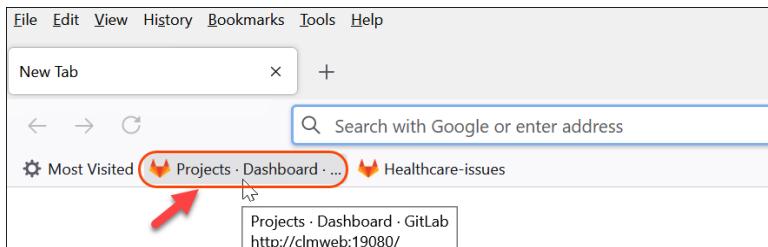
3.4 Access Gitlab repository to verify the code migrated

You can now access the Gitlab server repository to verify the migrated code.

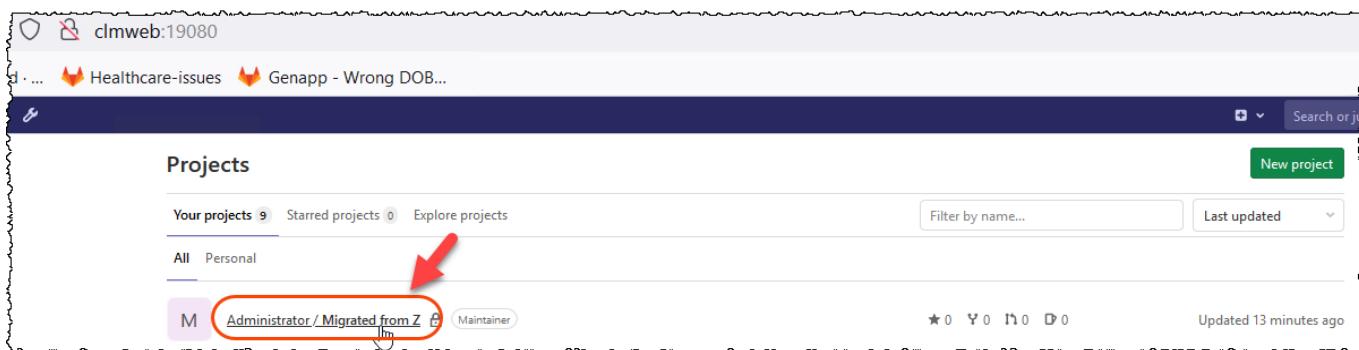
3.4.1 ► Start a browser again clicking in the icon in the bottom of your screen



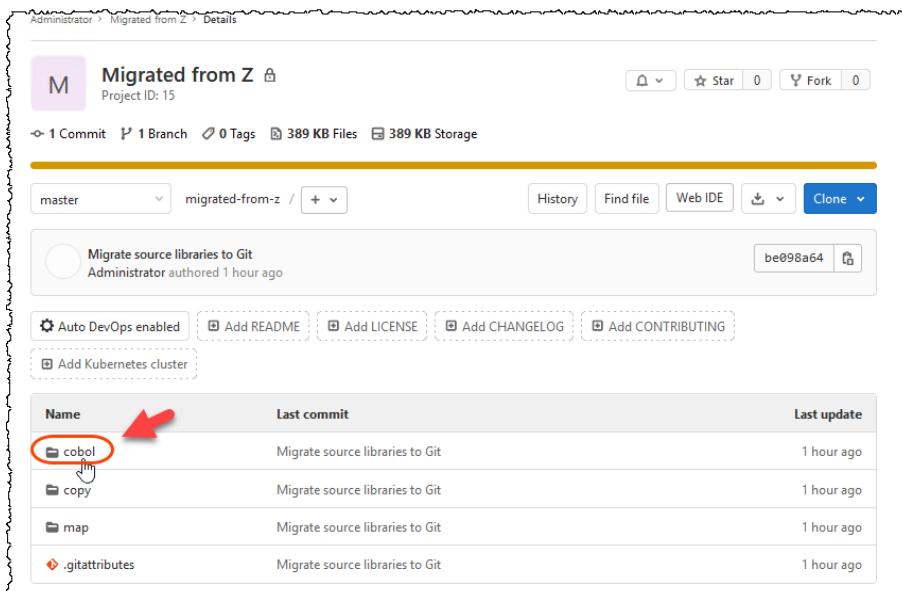
3.4.2 ►| Click on this bookmark below



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 / + v 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

Admin Migrate source libraries to Git Administrator authored 1 hour ago

lgacdb01.cbl 13.7 KB

```

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.

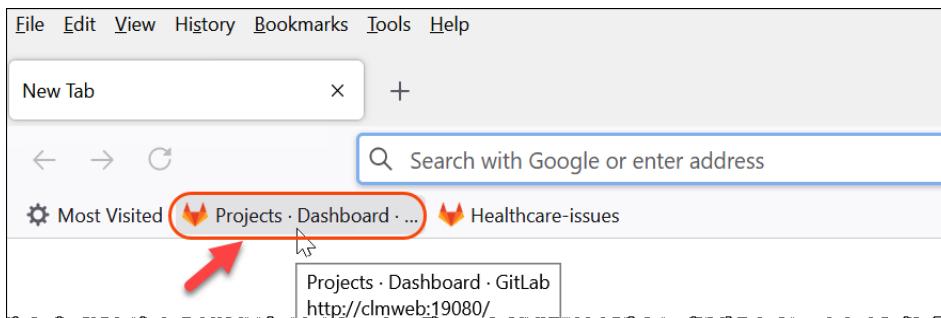
Section 4. Load the source code from Git to the local IDz workspace and start working with the migrated assets

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.

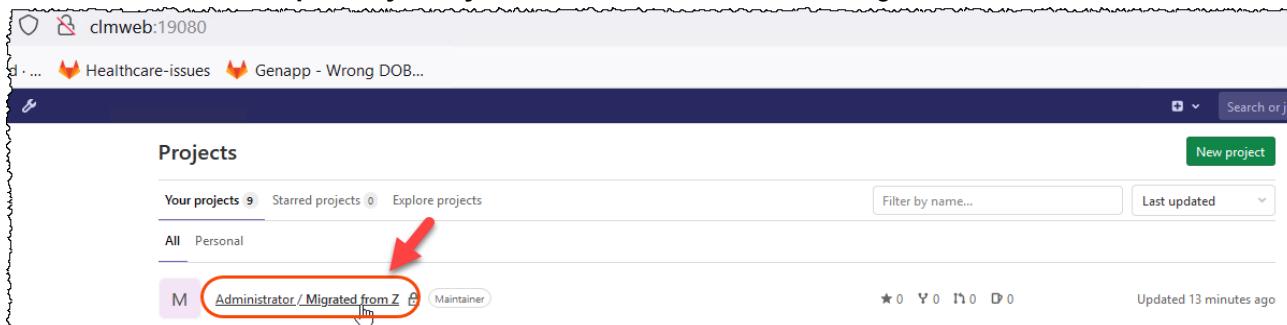
4.1 Cloning the Git server repository using IDz

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

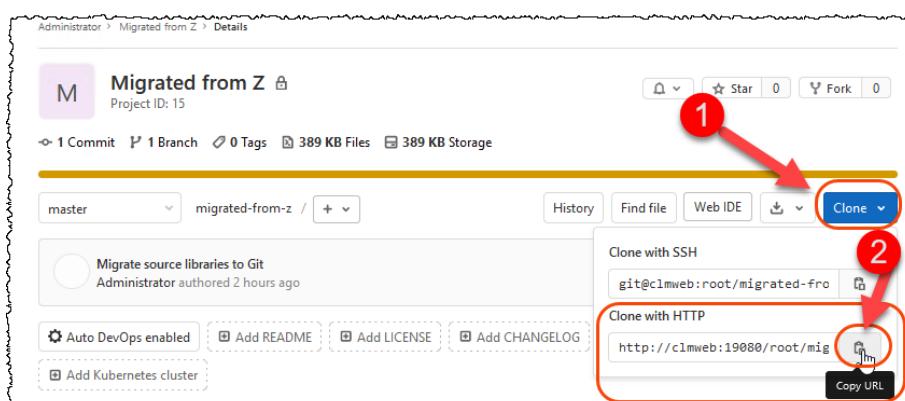


4.1.2 ► Click on the **repository that you created** and have the code migrated



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**.

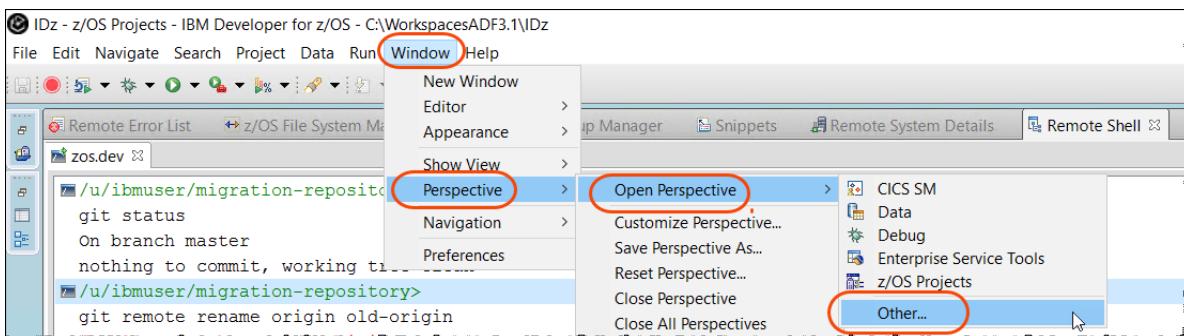
► ① Click on **Clone** (the blue button) and ② in the icon to **copy the URL**. This value will be kept in the windows clipboard.



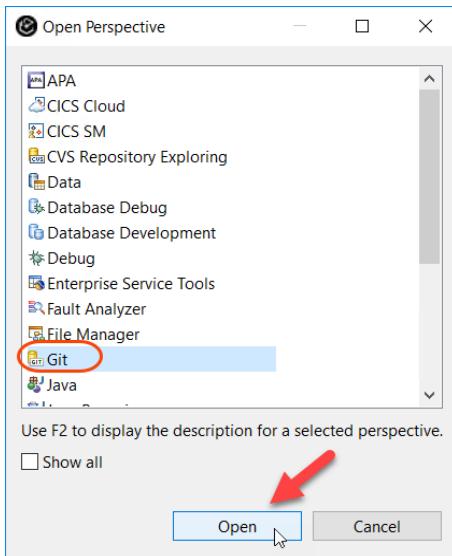
4.1.4 ► Go back to IDz using the icon  that is on the base of your screen:



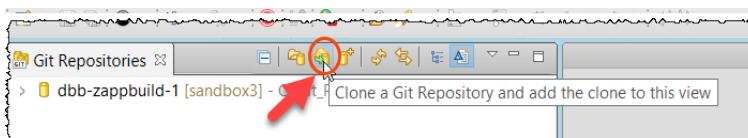
4.1.5 ► Open the Git perspective by selecting
Window > Perspective > Open Perspective > Other...



4.1.6 ► Select Git and click Open

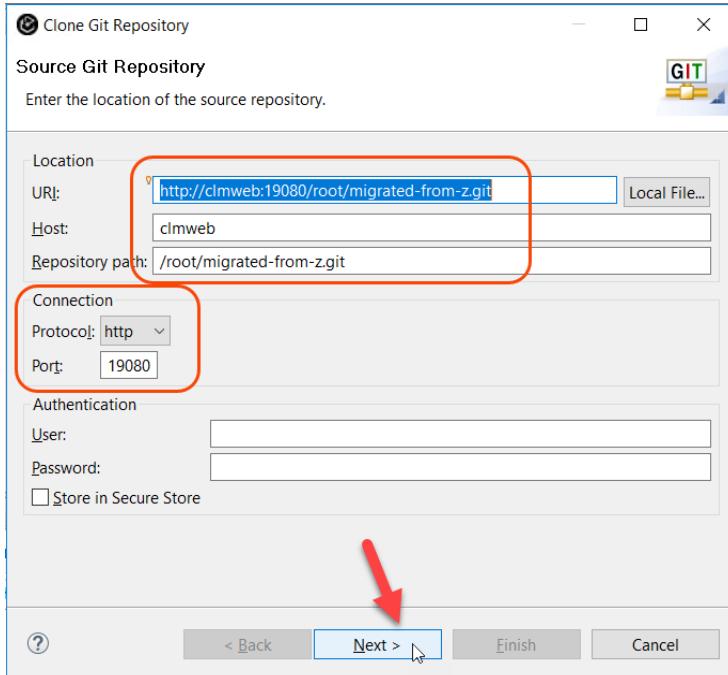


4.1.7 ► In the Git Repositories tab, click the  to Clone a Git Repository



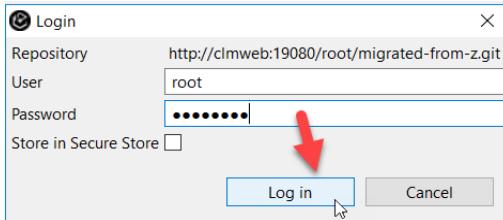
4.1.8 ► The values copied from the web page (*copy URL*) will be shown in the Clone Git Repository.
Tip: In case you don't see it, got back to the page and copy it again (steps 4.1.3 above).

► Click **Next**



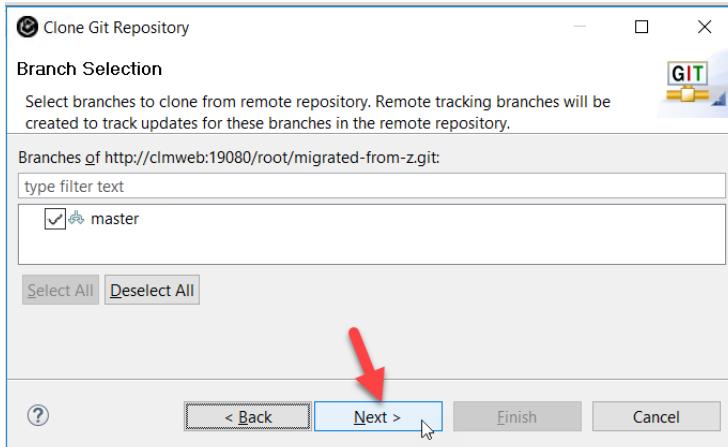
4.1.9 For credentials use: **root** and password **zdtlinux**

► Click **Log in**



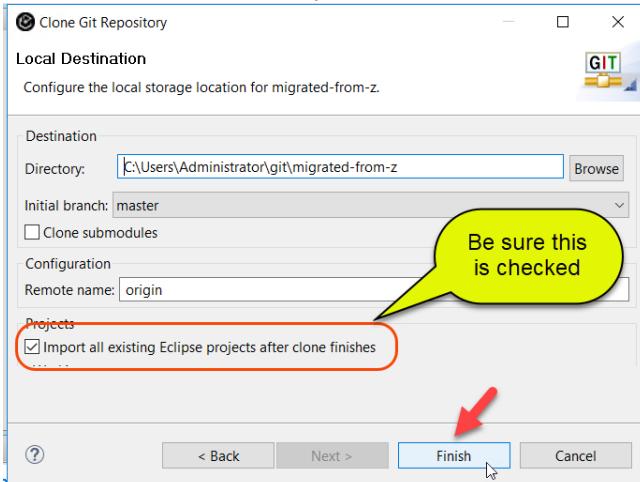
4.1.10 We have only one branch named **master**.

► Click **Next**



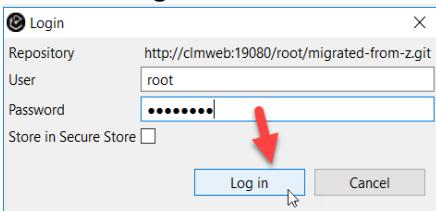
4.1.11 We will clone the repository on your local windows and import to be shown on IDz
You could find a specific folder to hold the cloned repository.

► Click **Finish** and accept the default location.

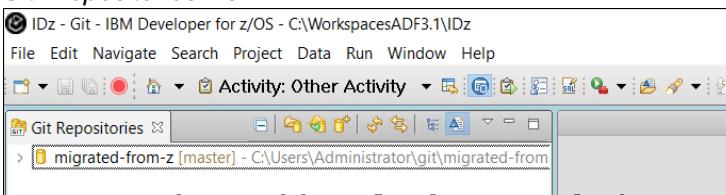


4.1.12 If credentials are asked again use: **root** and password **zdtlinux**

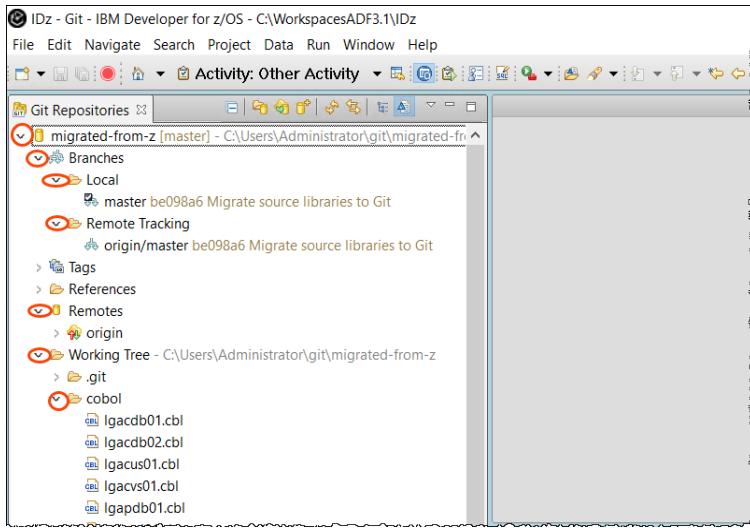
► Click **Log in**



4.1.13 The migrated repository was cloned from the Remote master repository and will appear in the *Git Repositories* view.

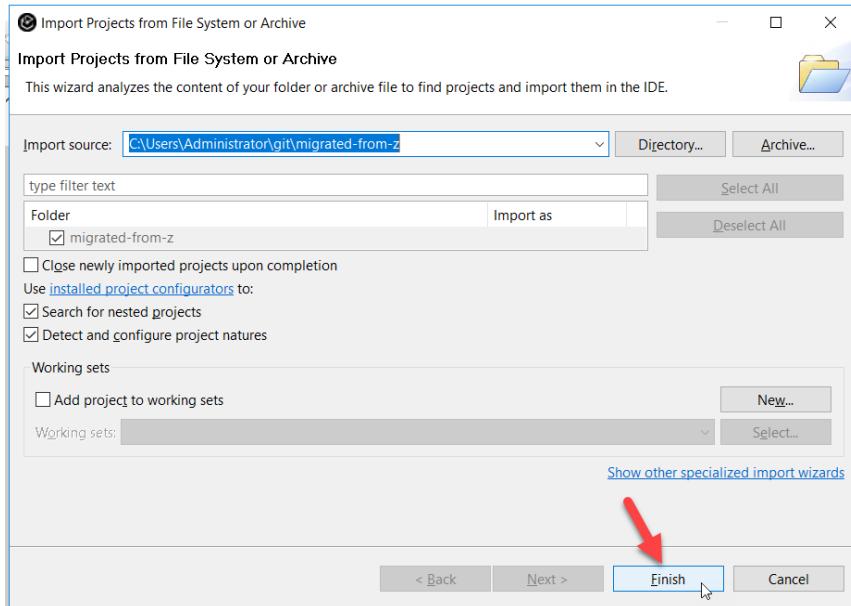


4.1.14 ► Expand the nodes by left clicking on the icon as shown below:



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.

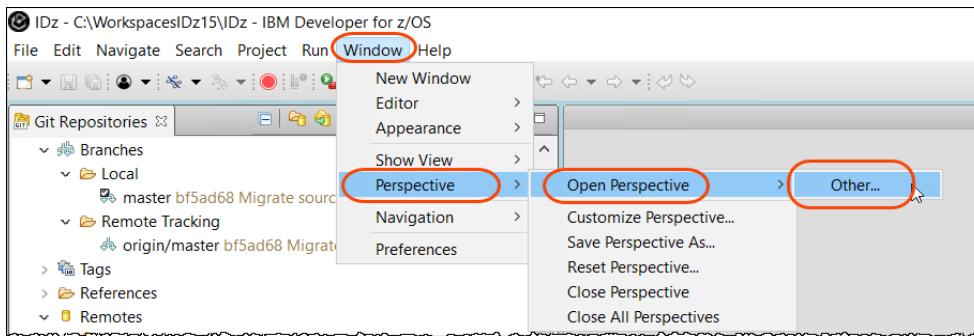
```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>genapp-migration</name>
    <comment></comment>
    <projects>
        </projects>
        <natures>
            <nature>com.ibm.ftt.dbbz.integration.dbbzprojectnature</nature>
            <nature>com.ibm.ftt.ui.views.project.navigator.local</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

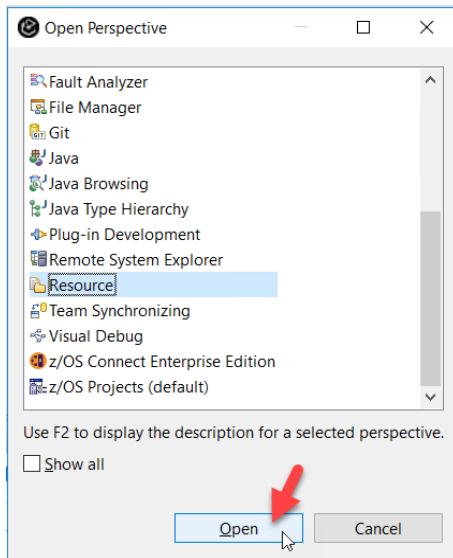
The z/OS projects perspective is the IDz perspective that developers use to work with the source code. Notice that you have cloned the project at your local workstation, but this will not be automatically moved to the Z/OS Projects perspective. You need to create a file named .project with the correct contents..

The easy way to do that is described below..

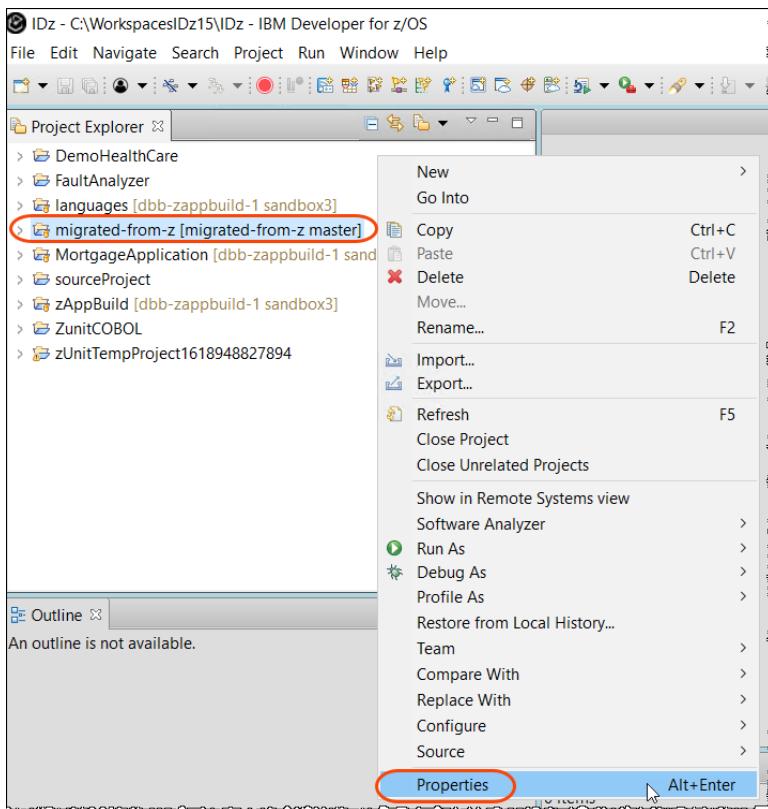
4.2.1 Switch to the **Resource Perspective** by selecting **Window > Perspective > Open Perspective > Other...**



4.2.2 Scroll down, select **Resource** and click **Open**

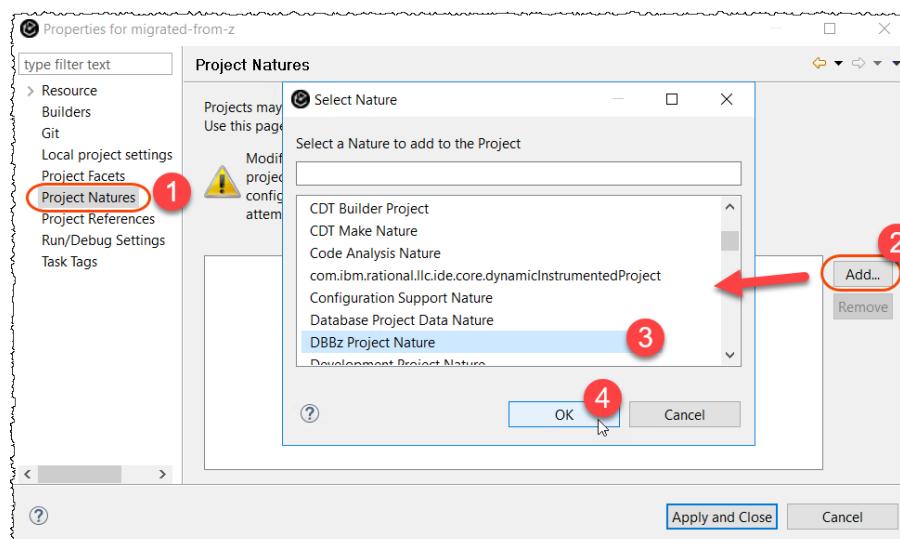


4.2.3 ► Right click **migrated-from-z** and select **Properties**

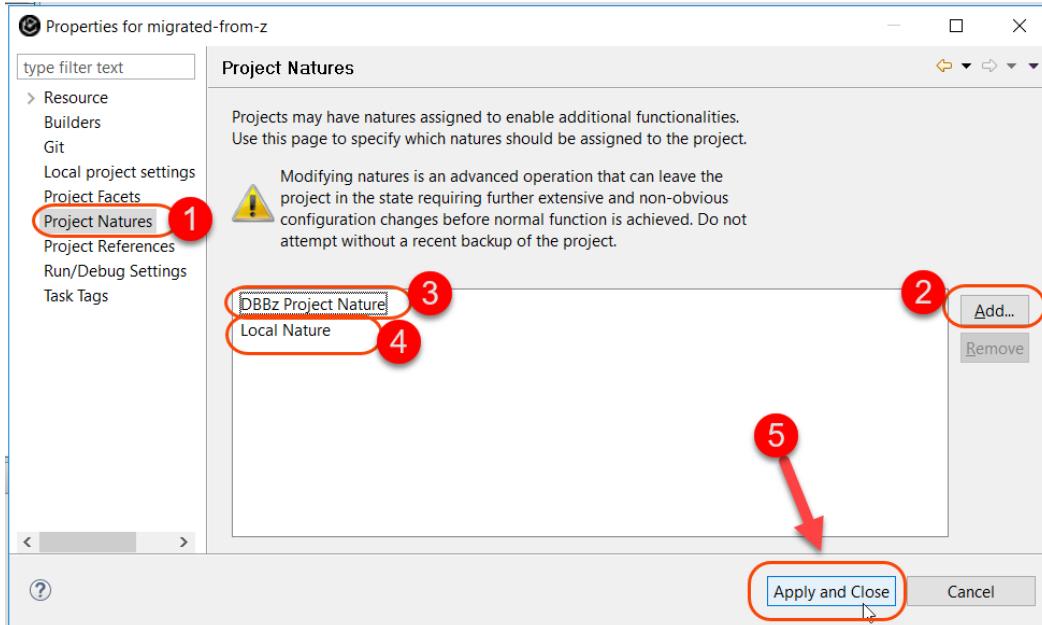


4.2.4 ► Select **Project Natures**, click **Add...** and **OK**

► 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



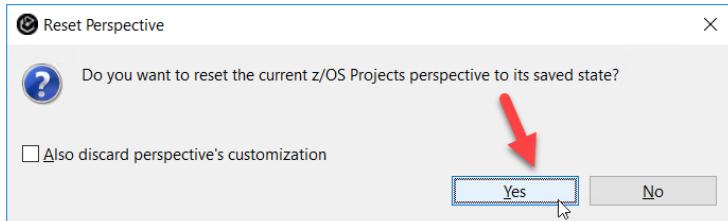
This will create the file named `.project` under the Project where the content will be as below..
Notice that this file is hidden in the project

```
<?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

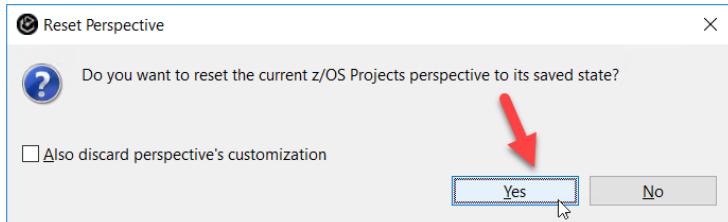


4.2.9 Notice that the project is still not showing there.

► Close the z/OS Projects

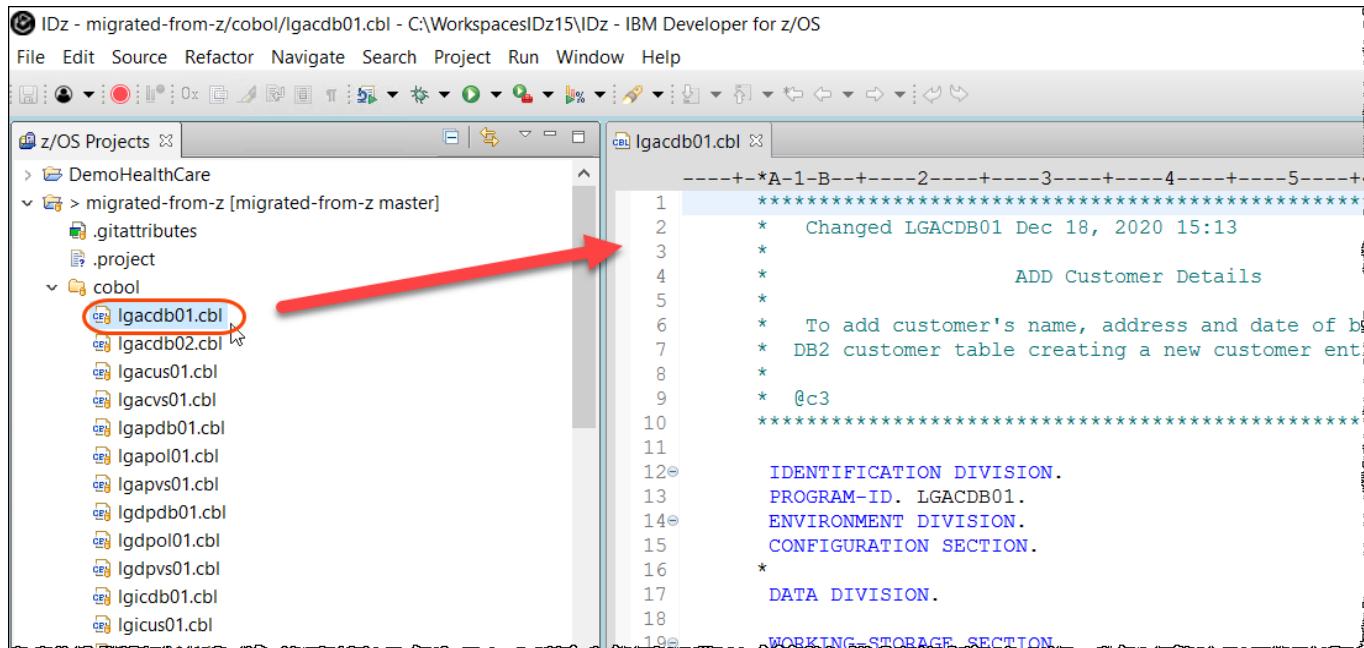
4.2.10 ► Perform the **Reset Perspective** again

4.2.11 ► Click **Yes** to reset the perspective to the saved state.



4.2.12 You will have the Project loaded now

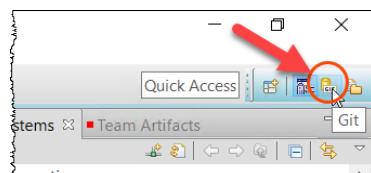
▶▶ Expand the project and double click on **lgacdb01.cbl** to see the migrated content.



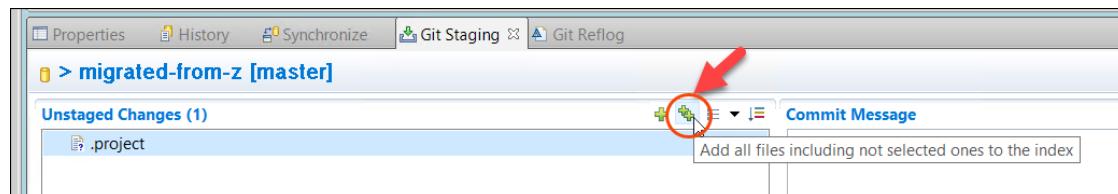
4.3 Commit the update to the Git server repository

The file *** .project** created on IDz workspace must be staged and committed to the Git server.

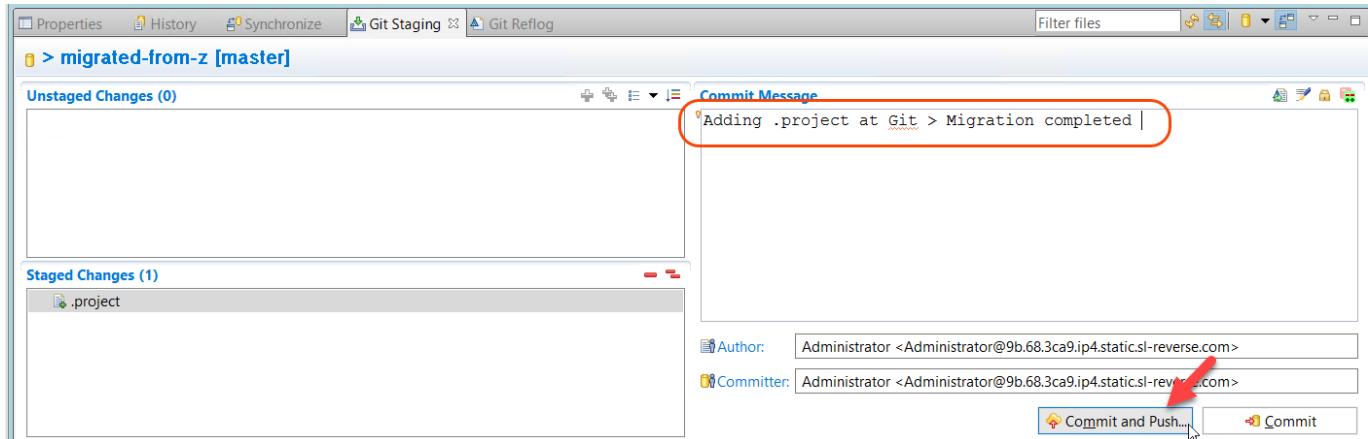
4.3.1 ▶▶ Switch to the **Git Perspective** by selecting the icon below



4.3.2 ▶▶ Using the *Git Staging* view click to stage the update.

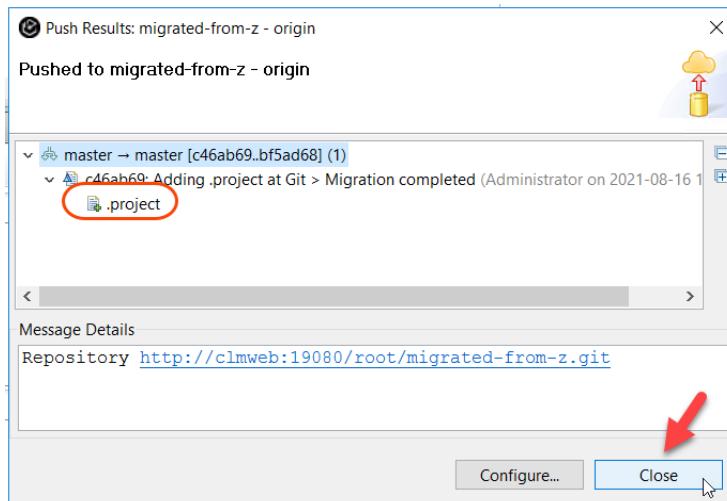


- 4.3.3 ➡ Add a message like “Adding .project at Git > Migration completed” and click **Commit and Push...**



- 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 highlighted with a red arrow pointing to it:

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

Congratulations! You have completed the Lab 11.

LAB 8 -(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, it says "Try IBM Z® software at no charge and with no installation. Your hands-on trial is available within 2 hours for 3 days (including weekends)." Below this are four main categories with arrows pointing to "Register for the trial" and "Explore the product first".

Category	Description	Action Links
Cloud, DevOps and API integration software trials	Updated! IBM z/OS® Connect Enterprise Edition Create efficient and scalable RESTful APIs for mobile and cloud applications. Learn how to: <ul style="list-style-type: none">Create REST APIs to: CICS® programs, IMS® transactions, DB2® for z/OS data, IBM® MQ Queues.Call a REST API from a COBOL application.	→ Register for the trial → Explore the product first
New! IBM Z Development and Test Environment	Use a web-based tool to learn how to self-provision a z/OS® environment on emulated Z hardware for agile development and test activities. Learn how to: <ul style="list-style-type: none">Explore how to use a z/OS system.Self-provision a z/OS from the IBM pre-built software stack ADCD.Extract CICS® and DB2® application artifacts from your source z/OS environment and deploy them to your z/OS target environment.	→ 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. Learn how to: <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.	→ 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 and 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.	→ Register for the trial → Explore the product first
Updated! IBM Application Discovery and Delivery Intelligence	Manage your software lifecycle and digital and cloud transformation. Learn how to: <ul style="list-style-type: none">Identify maintainability issues and update unwatchable code.Find CICS® programs able to be converted to APIs.View business terms and rules inventory in ADDI.Discover and manage terms and rules.	→ 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.	→ 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 Trialing

— 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 you “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

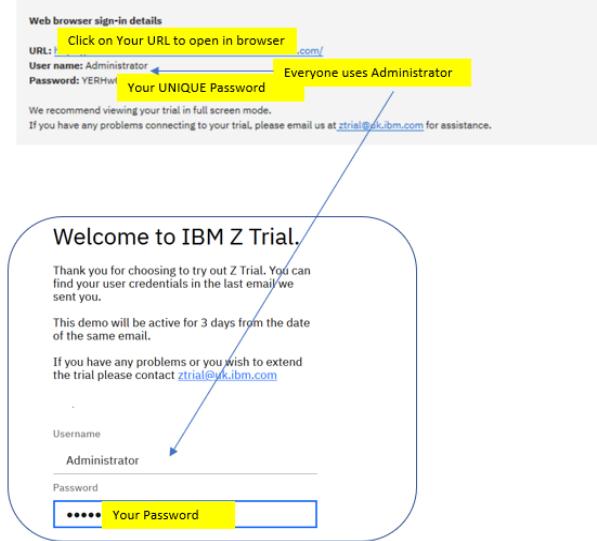
2.1.1 ► Open your ADDI zTrial instance

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.



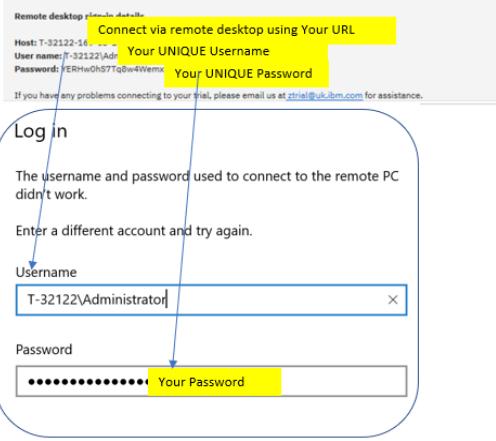
2 - Thru your Remote Desktop

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.

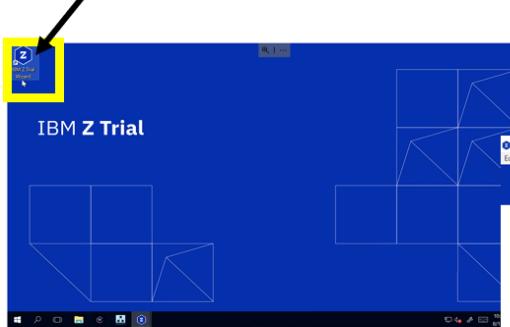


2.1.2 ► Start the ADDI Discover API Candidate wizard

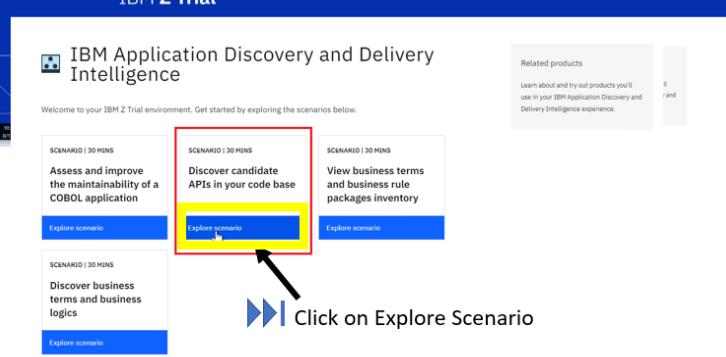
Depending on how you start your zTrial instance
One of the following screens will open

If this screen opens 1st

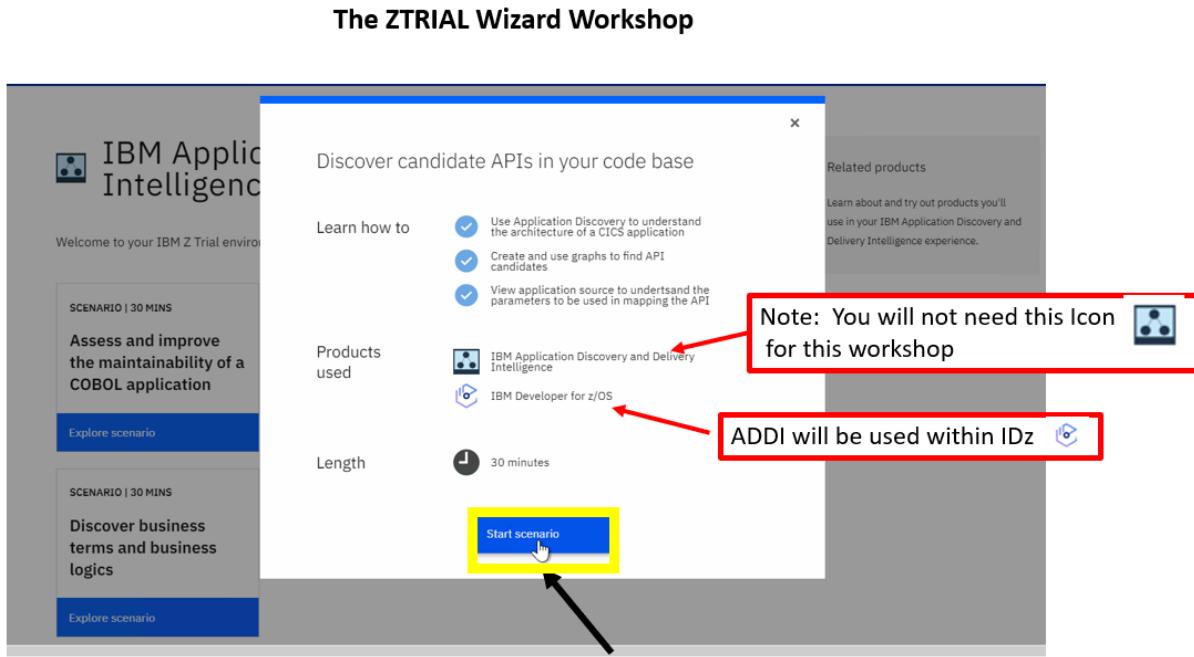
Click on the IBM Z Trial Wizard Icon to open the Z Trial Wizard screen



If the ZTRIAL Wizard screen opens directly
Click Explore scenario to start the workshop
It will open the Discover candidate API scenario/workshop overview



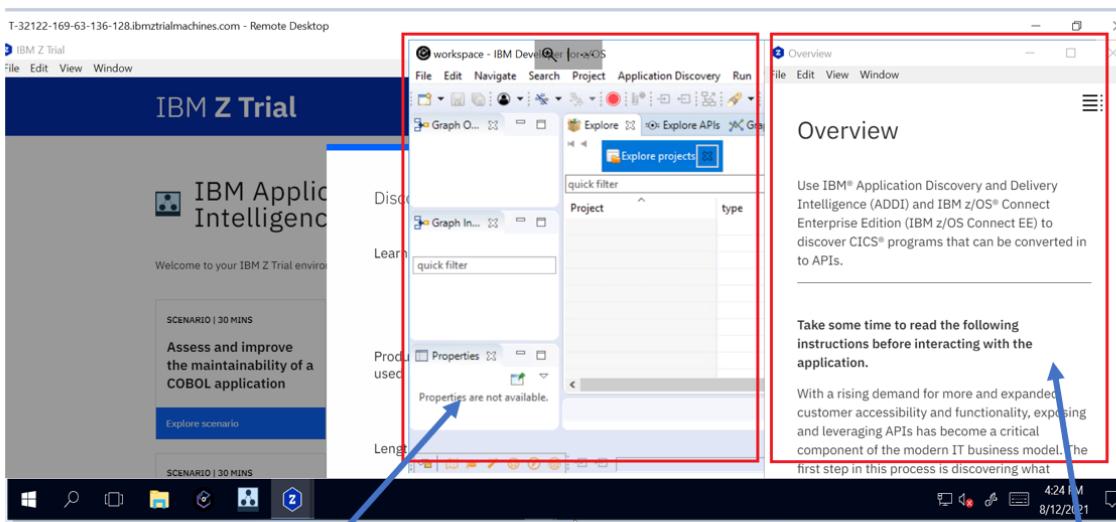
2.1.3 ► Start the Workshop scenario



► 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



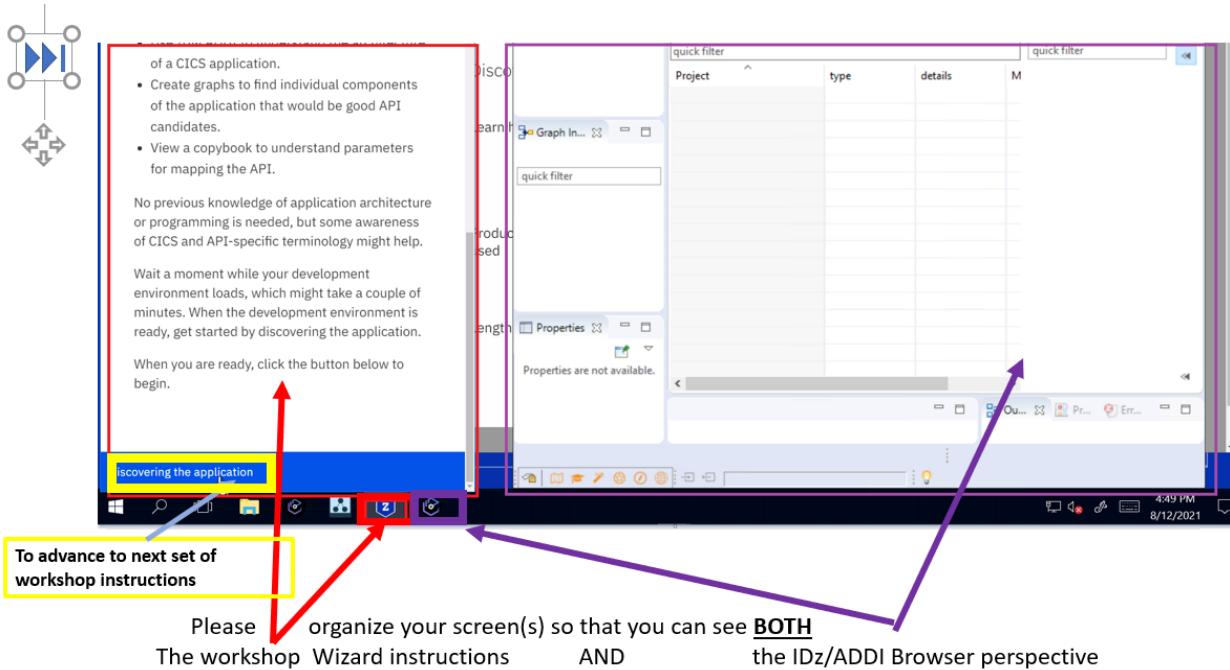
Start scenario will start IDZ and OPEN the ADDI Perspective where you will be doing all of your ADDI Lab work

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**

Go back to prior topic/ set of workshop instruction

To advance to next topic/set of workshop instruction

Overview

- Discovering the application
- 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

If you encounter any issues or have questions, contact the instructor.

You may also try any of the other scenario/workshops.

LAB 9 – (OPTIONAL) Running 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>.

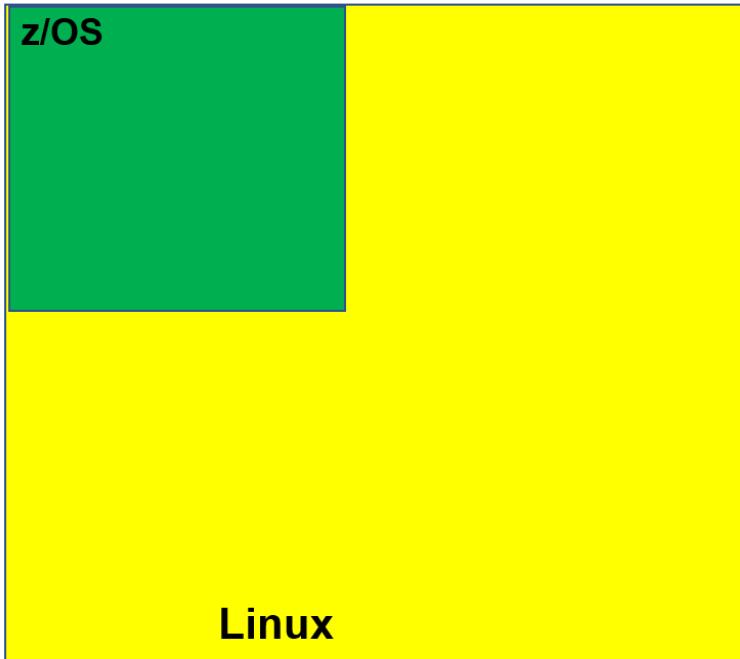
The IBM Distribution for Galasa is supported through [IBM Z Virtual Test Platform](#).

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

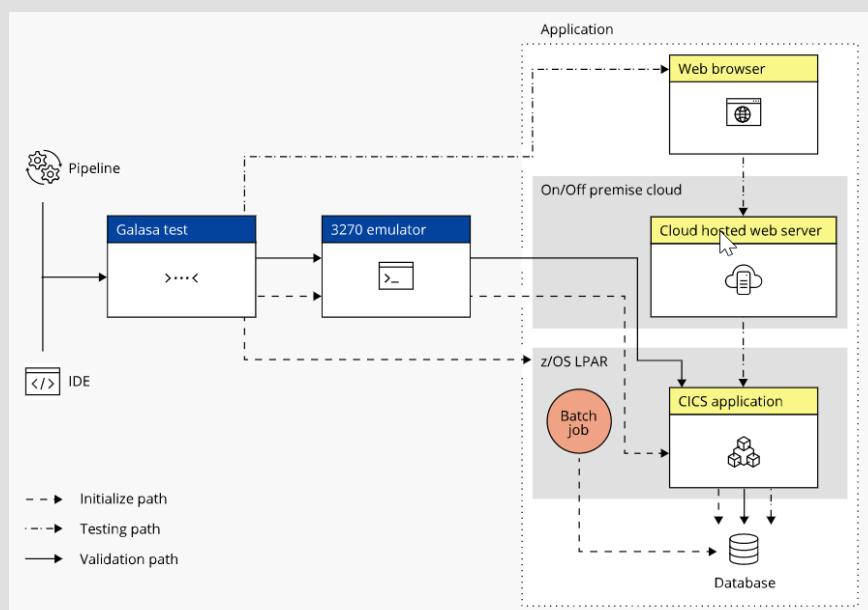
PART #3 –

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

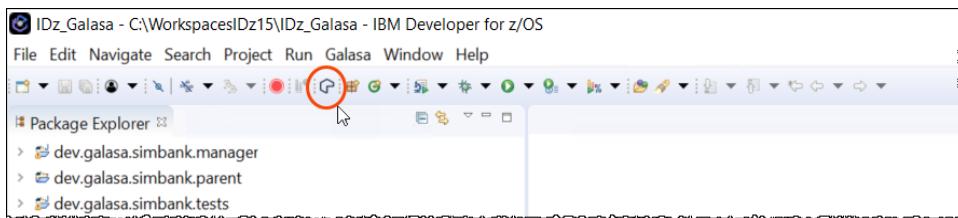
1.1 Start IBM Developer for z/OS

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**”.

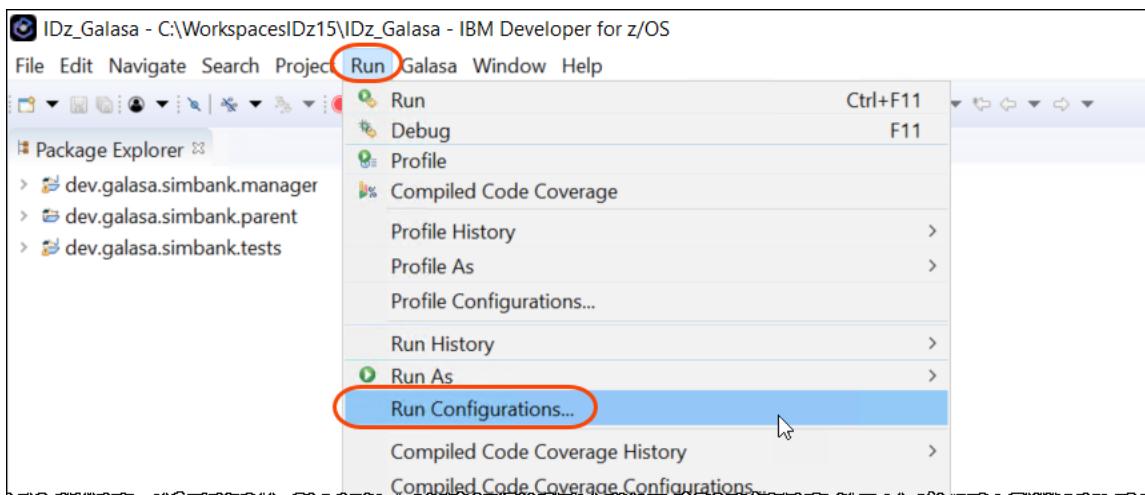


This will start the IDz environment. To speed this lab the Galasa eclipse plugin has already been installed into IDz and windows environment. You can see that the plugin is available by the inclusion of a ‘Galasa’ menu item and the **Galasa logo** within the tool bar seen below:



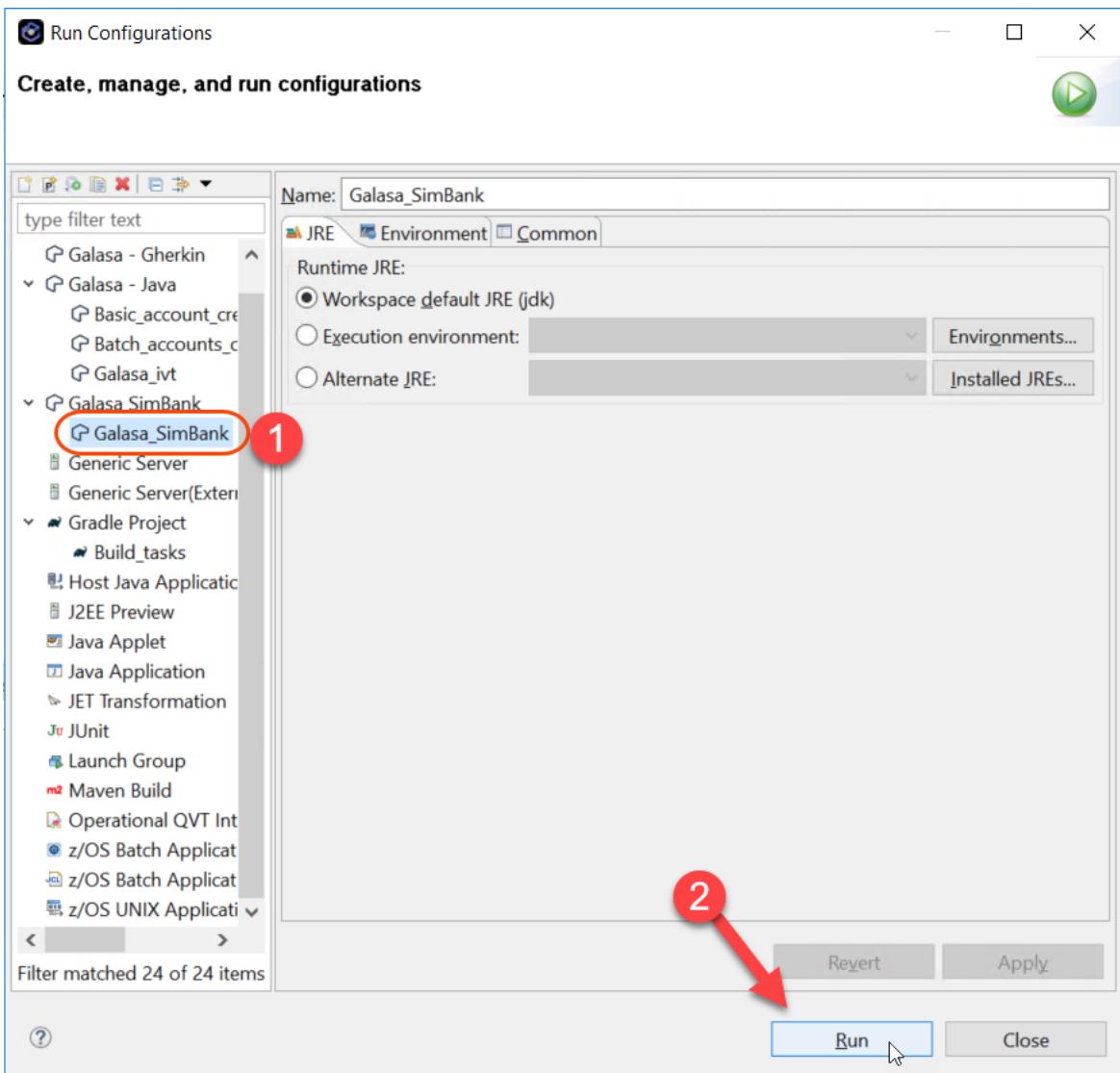
1.2 Run Galasa SimBank

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.ManagementFacilityList...
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.



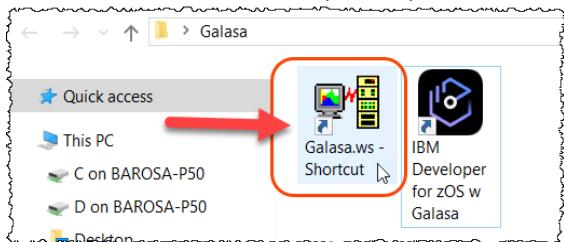
By exercising the Galasa framework against *SimBank*, you can pre-empt a lot (but not all) of the work and learning necessary to eventually hook your own tests up with a genuine mainframe environment. If the provided *SimBank* tests do not work, then it is unlikely that you will be able to run your own 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.

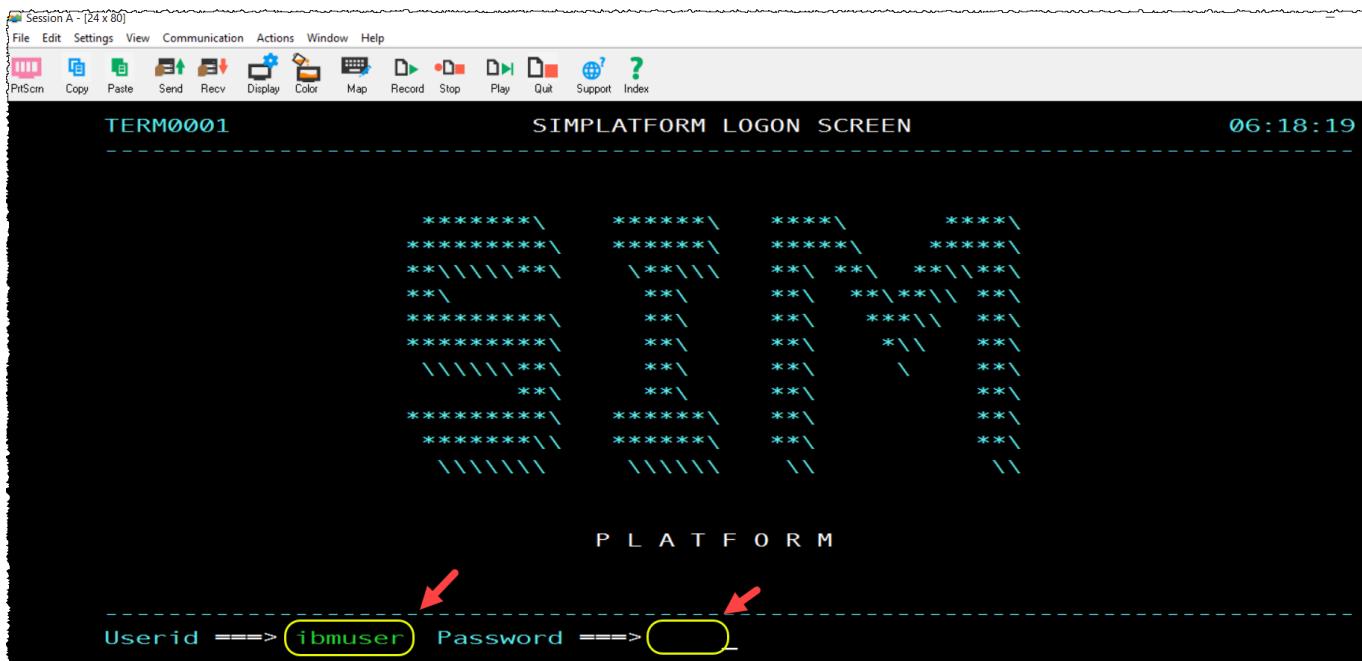
2.1 ► Minimize IDz.

2.2 ► From the Galasa folder on the desktop double click on the **Galasa.ws – Shortcut** icon to start IBM Personal Communications (PCOM).

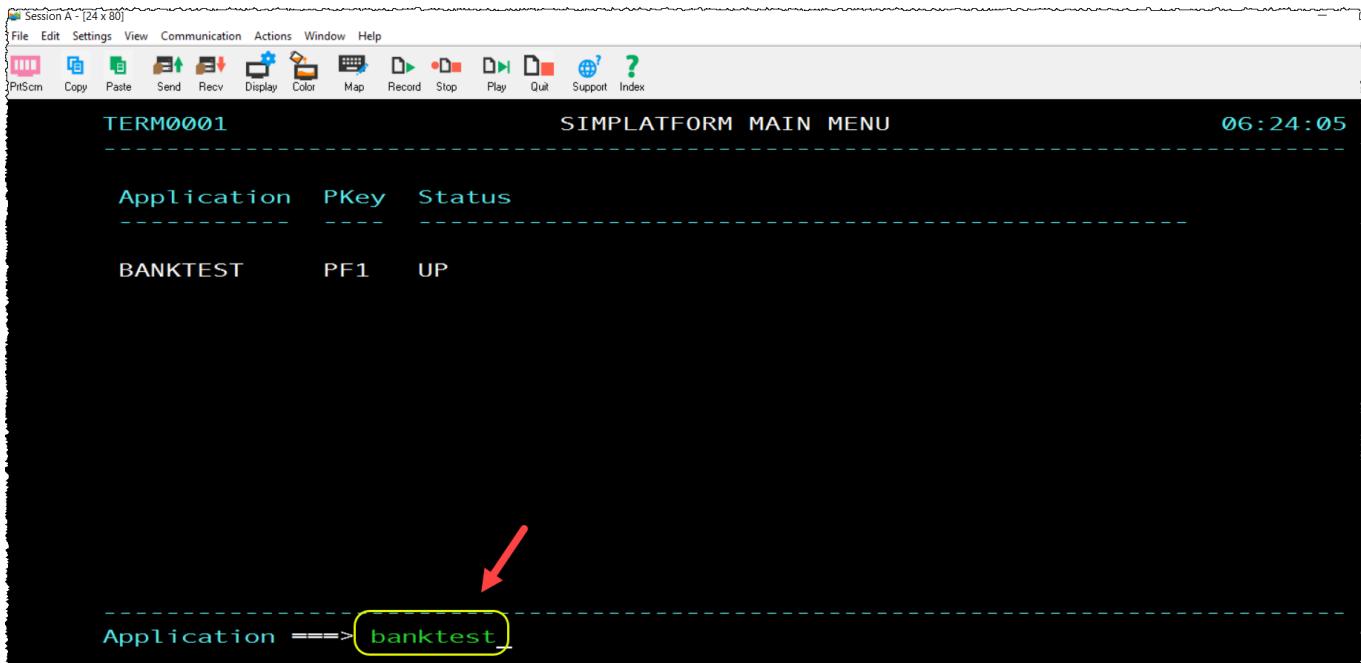


2.3 This is pre-configured to connect to the *SimBank* Application that is running within the virtual Environment.

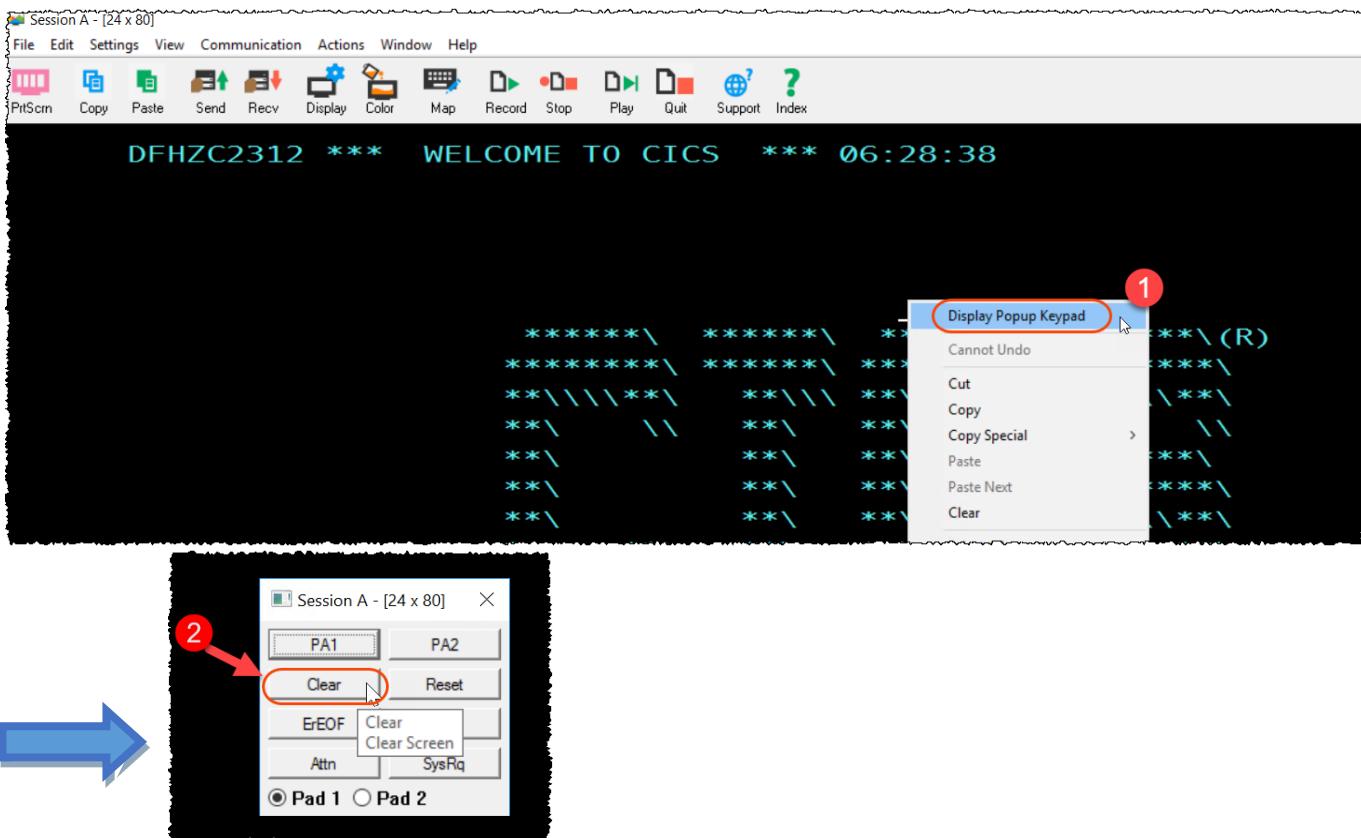
► Logon onto the application using **ibmuser** and **sys1** and press **enter**



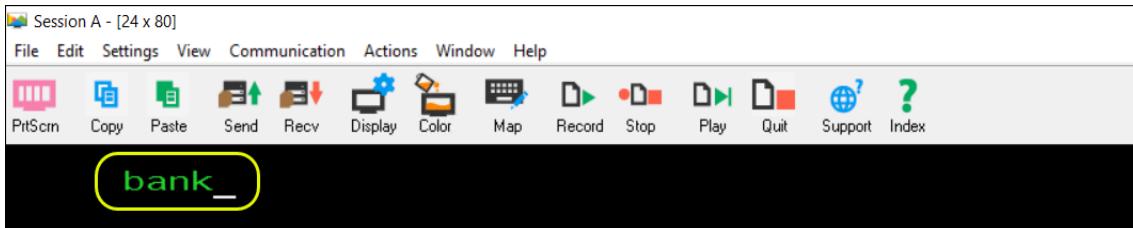
2.4 ➡ Enter the application by either typing **banktest** and pressing **enter** or by pressing **PF1**



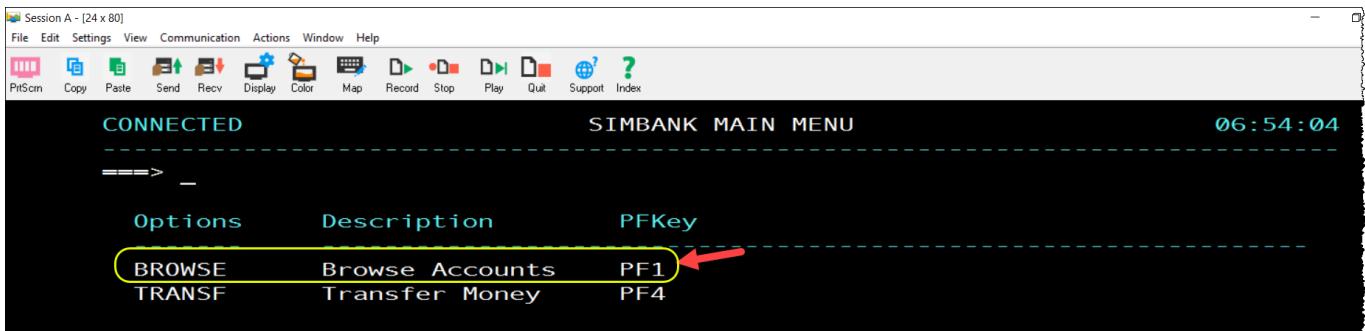
2.5 ➡ To clear the screen, **right click** on the dark space and select **Display Popup Keypad** and click the **Clear** button.



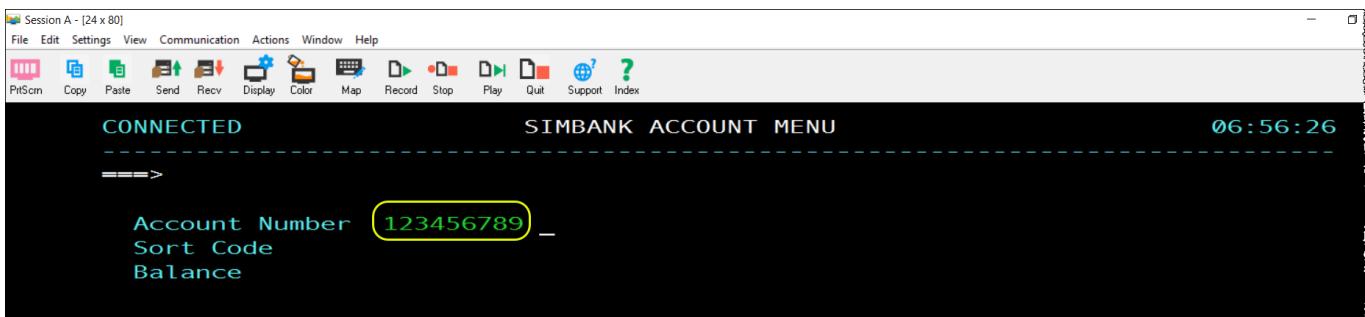
2.5 ➡ Run the bank transaction by typing **bank** and pressing **enter**



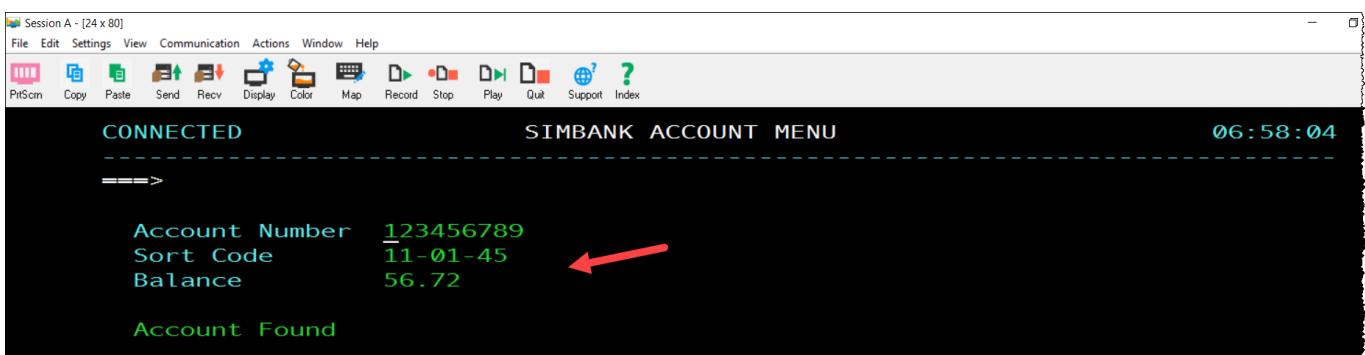
2.6 ➡ Press **PF1** to execute the **browse** option



2.7 ➡ Type the account number **123456789** and press **enter** to retrieve the customer details.

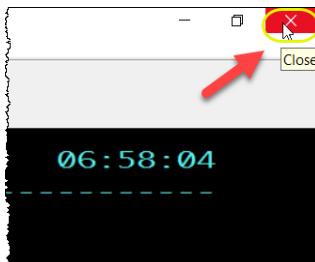


2.8 The customer data will be displayed.



The customer **987654321** also exists and can be retrieved.

2.9 ► The Personal Communications (PCOM) application can now be **closed**.

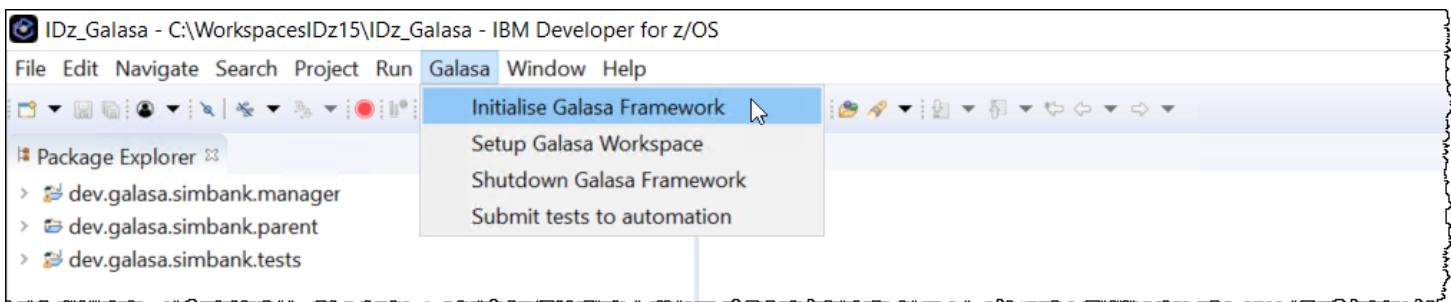


Section 3. Run the IVT test to validate that Galasa can connect to the SimBank Application

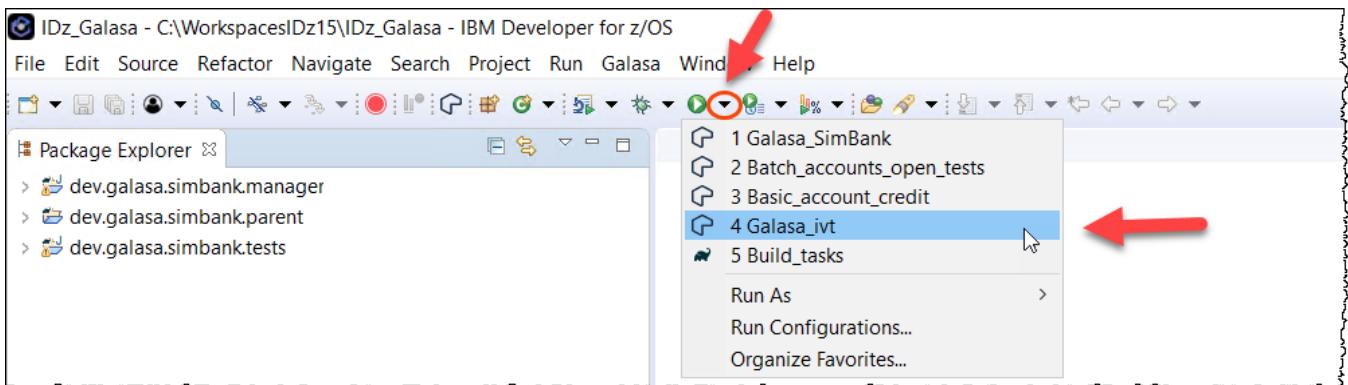
This basic test logs on to Galasa SimBank and examines an account.

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.



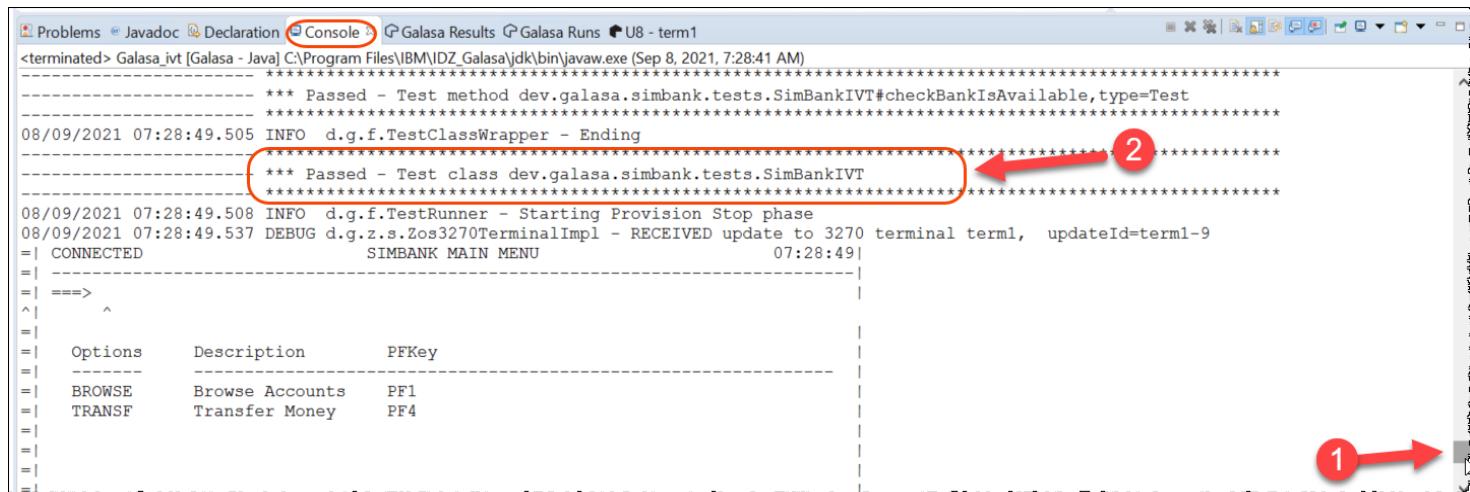
3.3 ► Click on the small downward pointing arrow to the right of the green 'play' icon. From the dropdown that appears select **Galasa_ivt**
Notice that the order could be different from the screen capture below.



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.



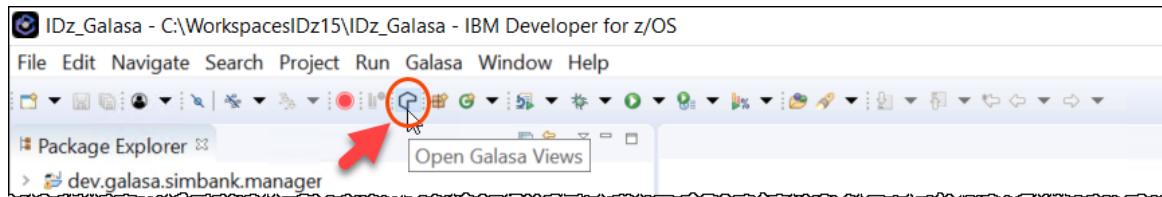
The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays a log of test execution. A red box highlights the line '*** Passed - Test class dev.galasa.simbank.tests.SimBankIVT'. Red arrows point from the number '1' to the 'Console' tab and from the number '2' to the highlighted text.

```
<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. On the left, there's an 'Overview' section with fields for 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), and Finished (Today, 15:39:10.835). To the right, a table titled 'SimBankIVT - Passed' lists two test methods: 'testNotNull' and 'checkBankIsAvailable', both with a 'Passed' result. At the bottom, tabs for General, Run Log, and Stored Artifacts are visible.

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..

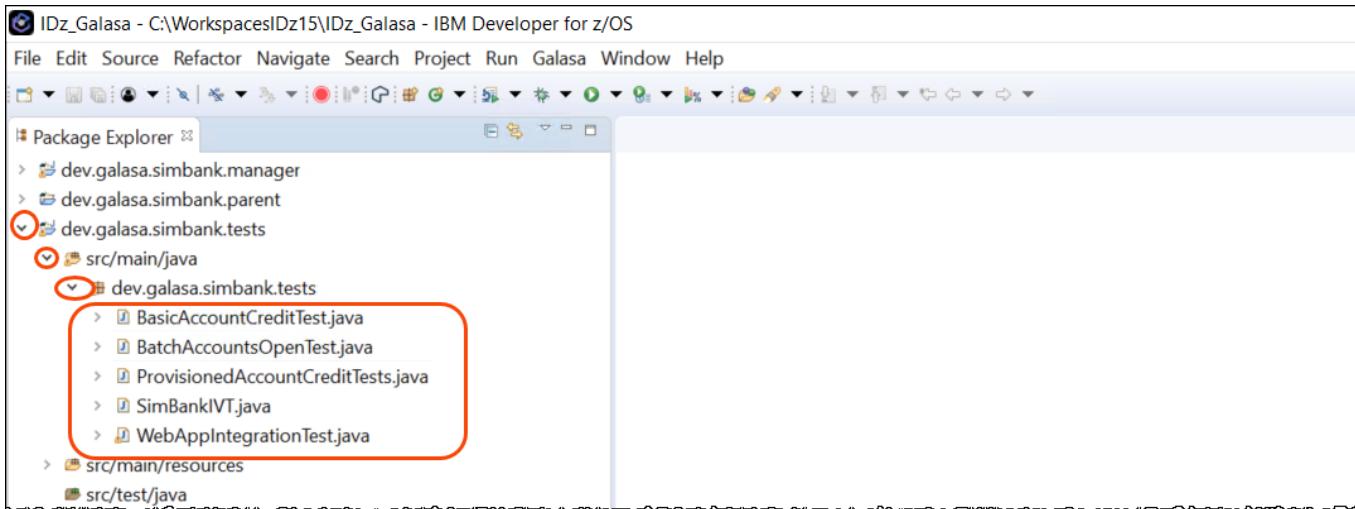
5.1 ► Ensure that IDz is open and the SimBank application is running as per the instructions in the previous section. In the *Package Explorer* view you will note that three projects are available:

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, and Help. The toolbar has various icons for file operations. The 'Package Explorer' view on the left shows three projects: 'dev.galasa.simbank.manager', 'dev.galasa.simbank.parent', and 'dev.galasa.simbank.tests'. The rest of the screen is mostly blank.

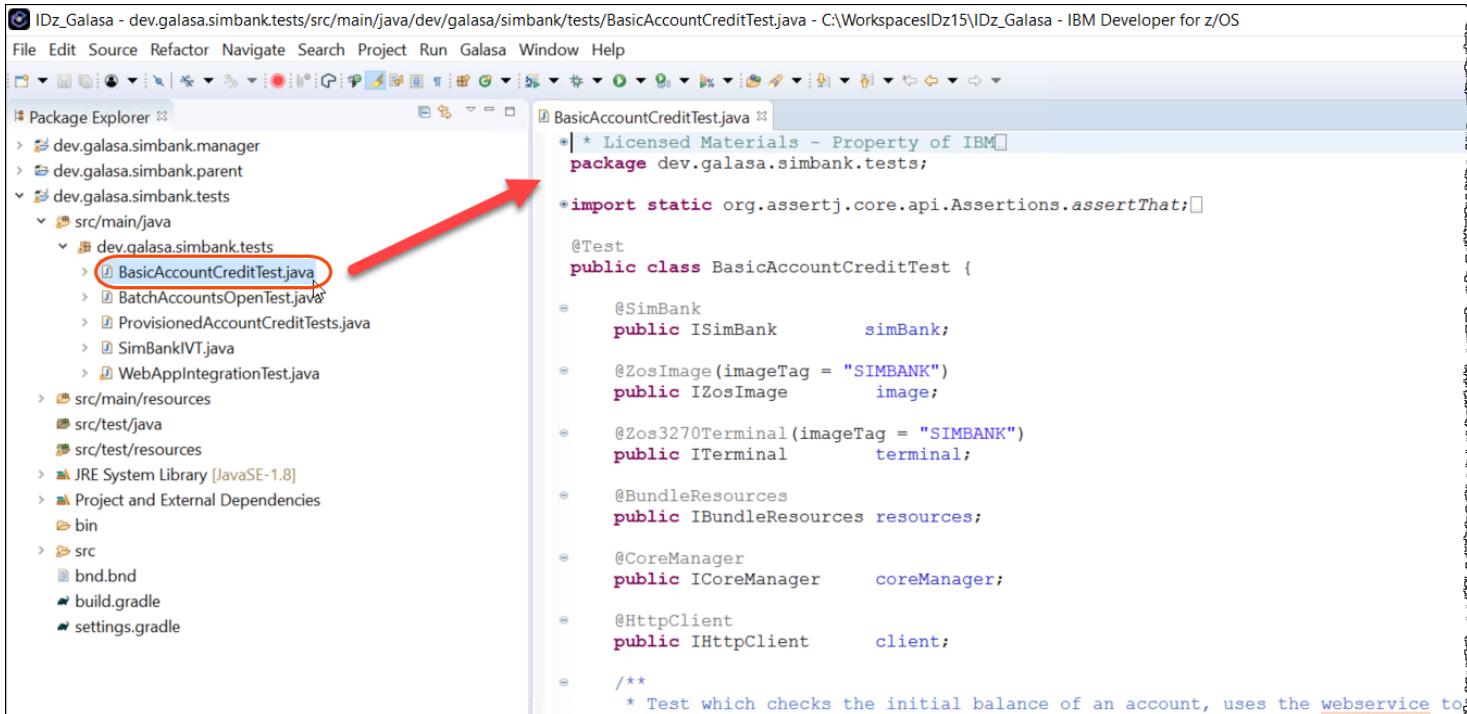
5.2 These three projects are the standard sample tests that are included as examples within the eclipse Galasa plugin.

▶ Expand the project **dev.galasa.simbank.tests**, **src/main/java** and **dev.galasa.simbank.tests**.
The project should contain the five tests.

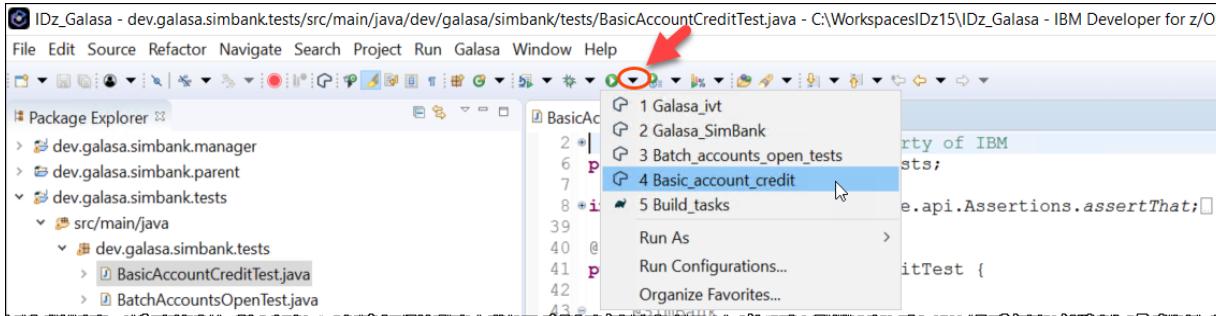
The **SimBankIVT test** was the test that was executed in *Part #1*. Please feel free to examine the source code of the test.



5.3 ▶ Double click on **BasicAccountCreditTest.java** to open the test

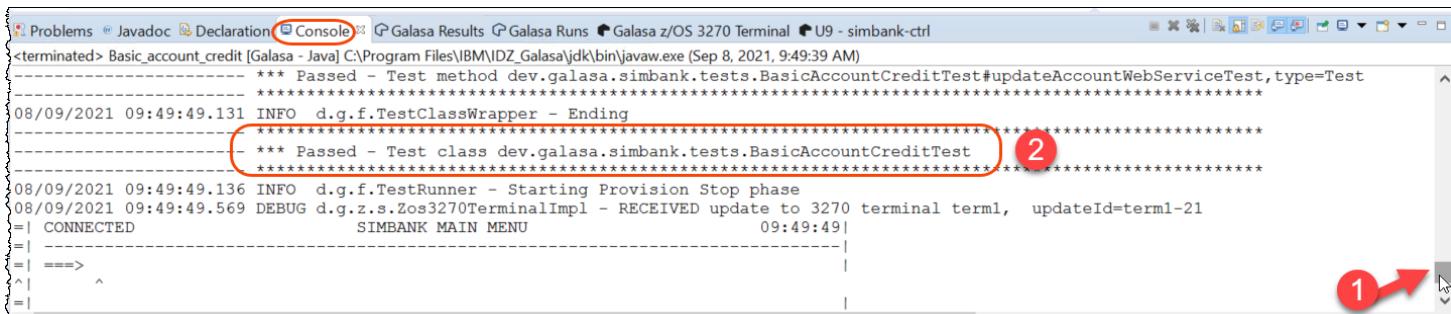


5.4 ► To execute this test, similarly as before, click the drop down next to the green play icon and select 'Basic_account_credit' – note that the numbering might be different to that in the screenshot



5.5 The test will execute.

► Using the **Console** view, scroll up a bit and you will see the message as below:



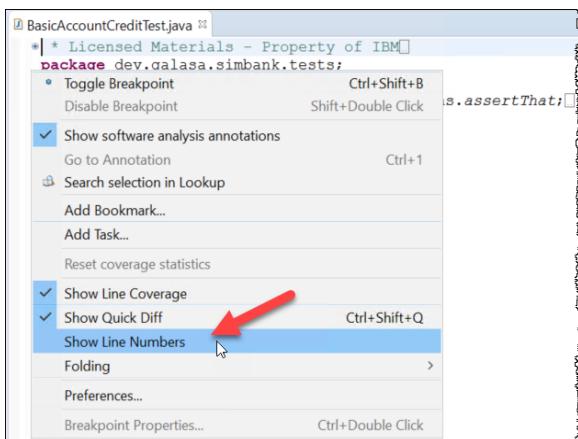
This message ensure that the test has correctly finished. If this message does not appear or the test appears to have failed. Ensure that the SimBank application is running by following the steps in Part #1. Remember if IDz is closed then the SimBank application will exit and will need to be manually 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
    @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 its 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 IHttClient 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 objects 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.

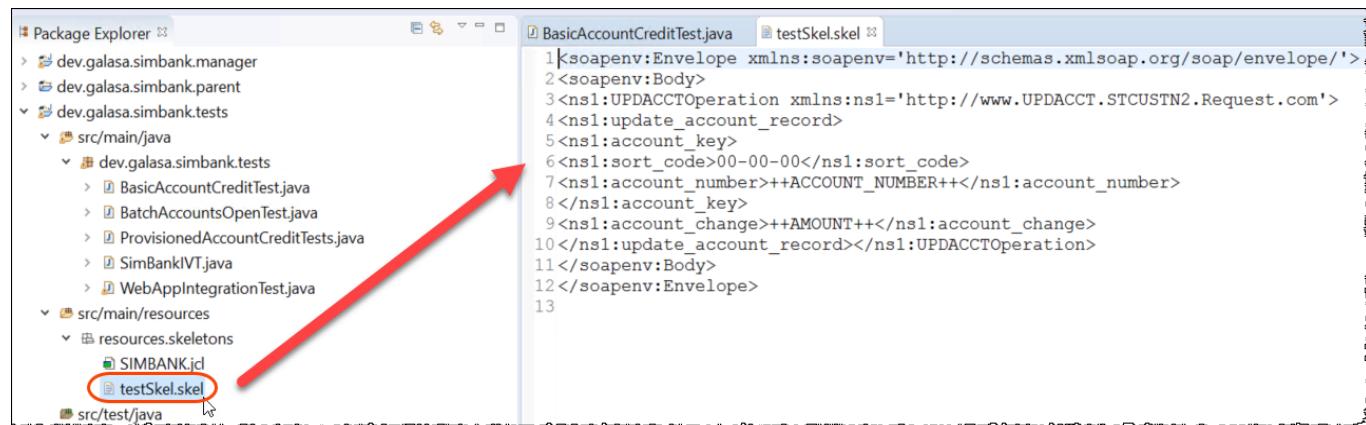
First a *HashMap* is created and populated with an *account number* and an *amount*.

This is then used to fill out a skeleton file found at

Src/main/resources -> resources.skeletons -> testSkel.skel

```
96     // Set the amount to 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

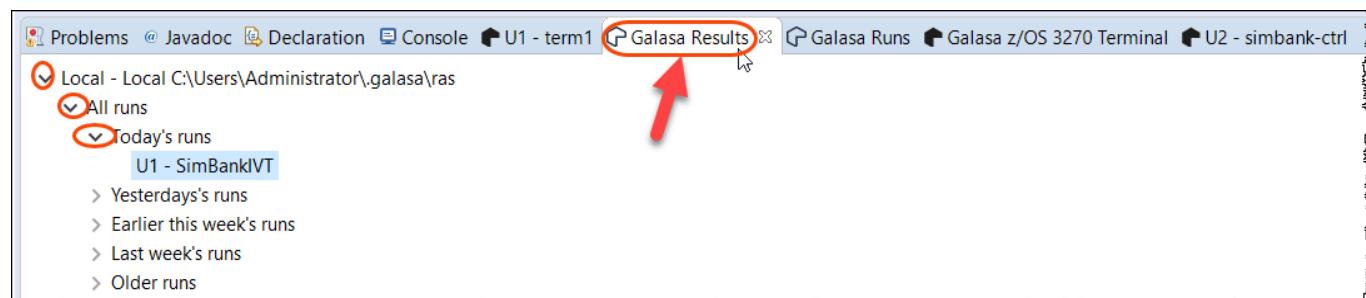


Section 7. Examine the Run Editor to see the stored artifacts

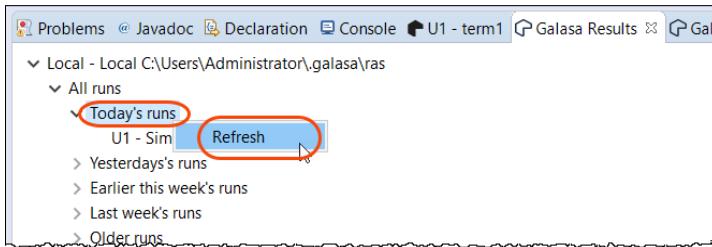
Now you have seen the source code that makes up the test and run it within the Galasa framework we will now examine the Galasa run editor.

7.1 Examine the Run Editor to see the stored artifacts

►| Using the '**Galasa Results**' View, expand Local, All runs and Today's runs

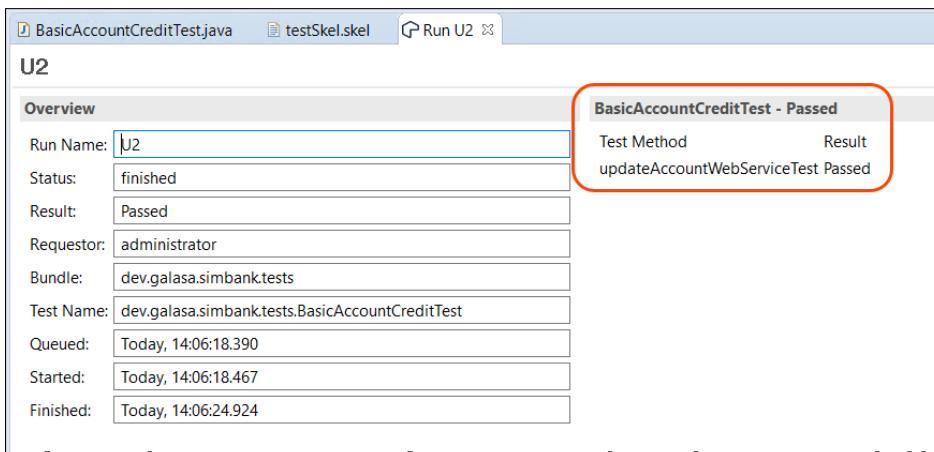


7.2 ► Right click on Todays runs and select Refresh.



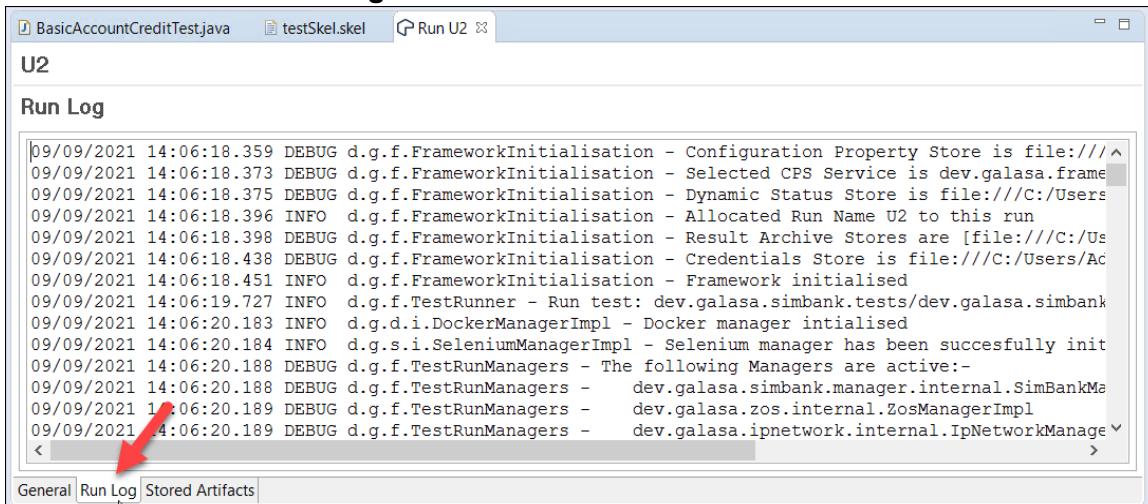
This should bring up (if not already available) **U2 – BasicAccountCreditTest**
(Notice that this number can vary. Usually is **U2**, but it may be different if you run multiple times)

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



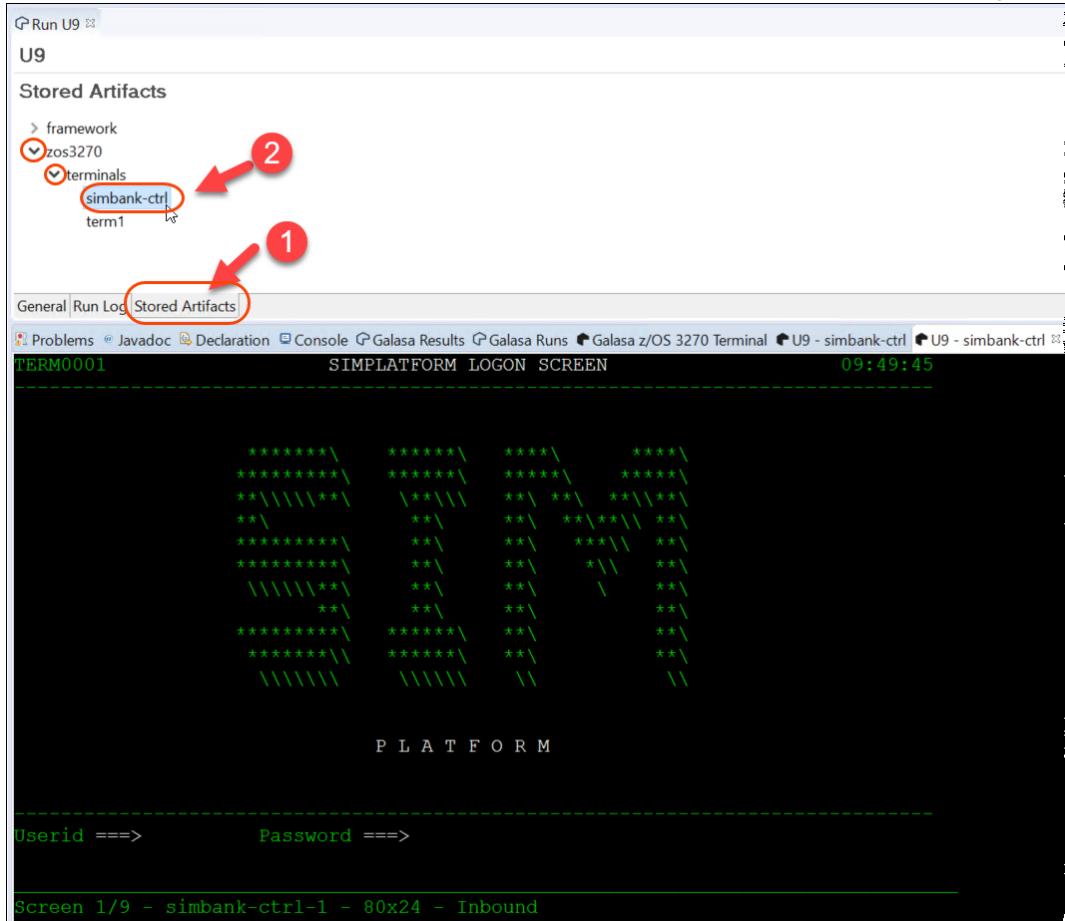
7.5 This view shows the entire log of the test as it ran.

► **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.

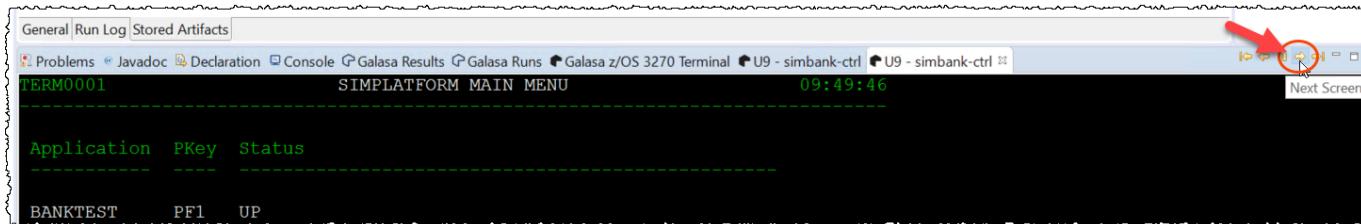


7.6  Click on the **Stored Artifacts** view of the Run editor Any resources that are captured during the test will be stored in the stored artifacts view.

▶| Expand **zos3270 -> terminals** and double click on **simbank-ctrl** to bring up the terminal view



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

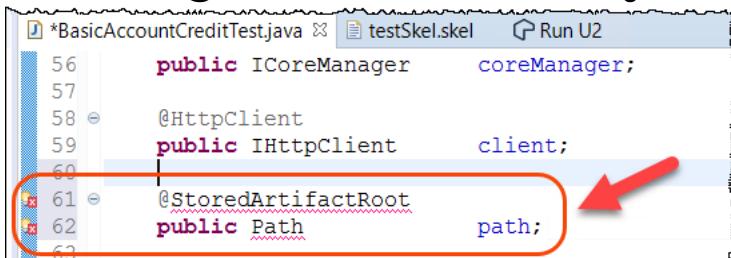
Unlike with 3270 screens, the HTTPManager cannot store every request and response that it processes as this might contain a lot of data which we would not want to always log, so Galasa provides a way to easily log this content. In this case we will change this test to log the request and response messages 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)

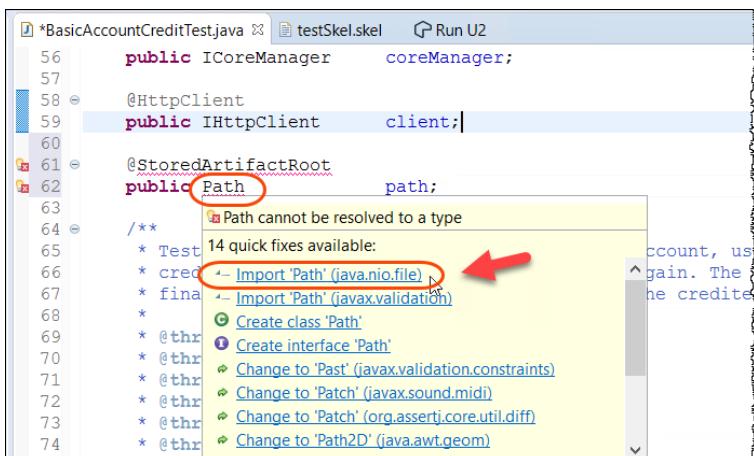
```
@StoredArtifactRoot  
public Path path;
```

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 `import StoredArtifactRoot` 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);

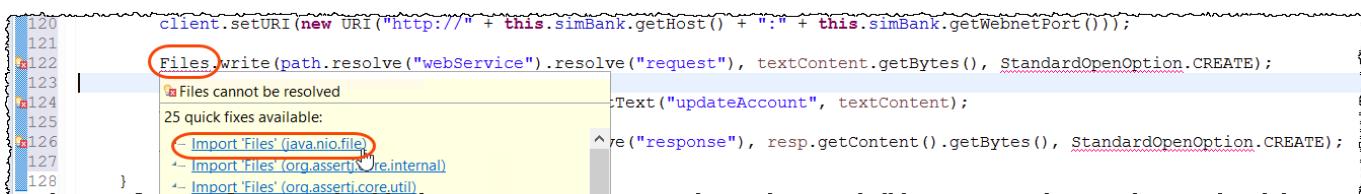
HttpClientResponse<String> resp = client.postText("updateAccount", textContent);

Files.write(path.resolve("webService").resolve("response"), resp.getContent().getBytes(),
StandardOpenOption.CREATE);
```

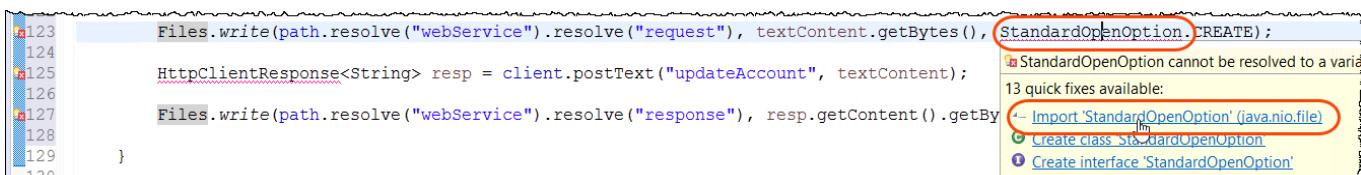
Notice that more errors are flagged. We must correct like we did a while ago.



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

A screenshot of an IDE showing Java code. A tooltip appears over the word 'HttpClientResponse' with the message 'HttpClientResponse<String> resp = client.postText("updateAccount", textContent);' and 'HttpClientResponse cannot be resolved to a type'. Below the tooltip, a list of '12 quick fixes available' includes 'Import 'HttpClientResponse' (dev.galasa.http)'. The code itself shows a snippet like this:

```
126
127
128
129
130 } | HttpClientResponse<String> resp = client.postText("updateAccount", textContent);
131
132 }
```

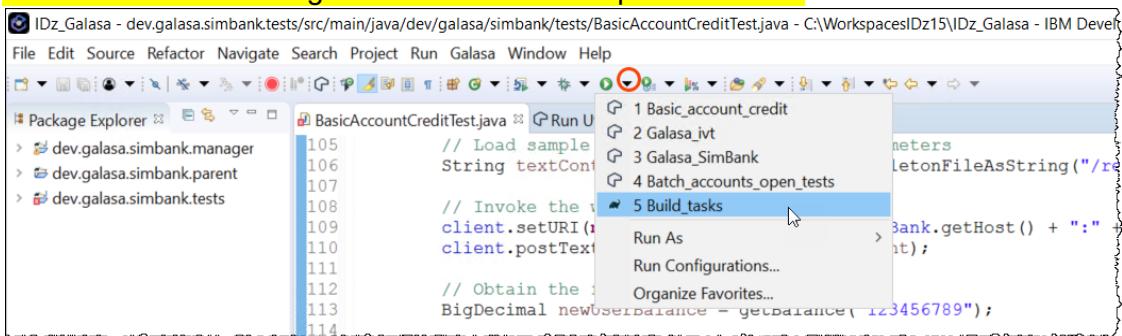
The Java errors should be gone

8.8 ► Use **Ctrl + S** to save the change.

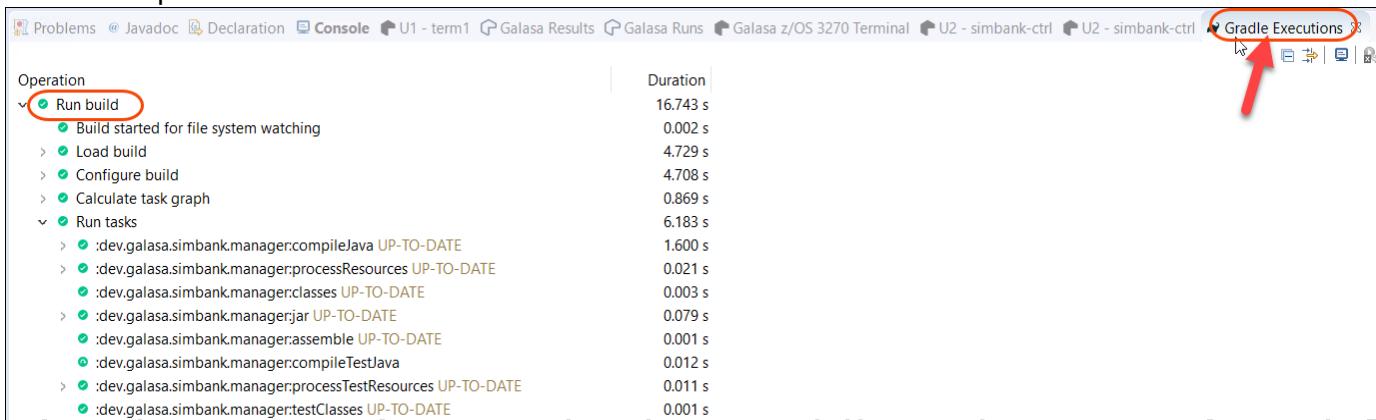
8.9 You can now build the tests by

► 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

```

1 Build_tasks
2 Basic_account_credit
3 Galasa_jvt
4 Galasa_SimBank
5 Batch_accounts_open_tests
Run As
Run Configurations...
Organize Favorites...

```

8.12 The test will execute.

► Using the **Console** view, scroll up a bit and you will see the message as below:

```

08/09/2021 09:49:49.131 INFO d.g.f.TestClassWrapper - Ending
*** Passed - Test class dev.galasa.simbank.tests.BasicAccountCreditTest

```

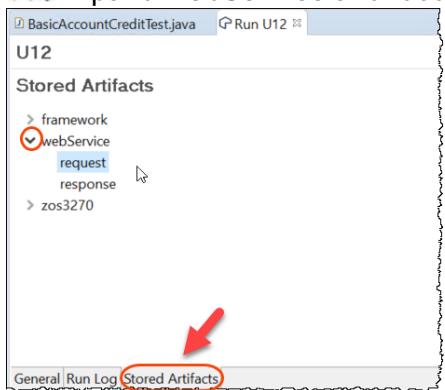
8.8 ► Within the Galasa Results View right click on Today's runs and select refresh

8.9 ► Double click the latest run of the **BasicAccountCreditTest** to open the Galasa run editor for your new run

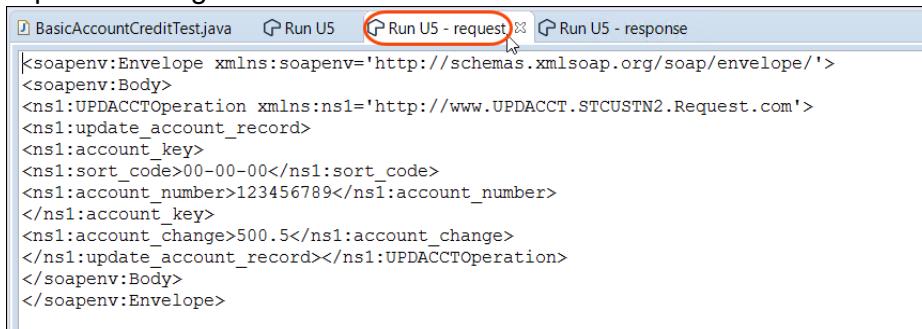
Run Name:	U12
Status:	finished
Result:	Passed
Requestor:	administrator
Bundle:	dev.galasa.simbank.tests
Test Name:	dev.galasa.simbank.tests.BasicAccountCreditTest
Queued:	Today, 08:31:04.142
Started:	Today, 08:31:04.220
Finished:	Today, 08:31:09.990

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.

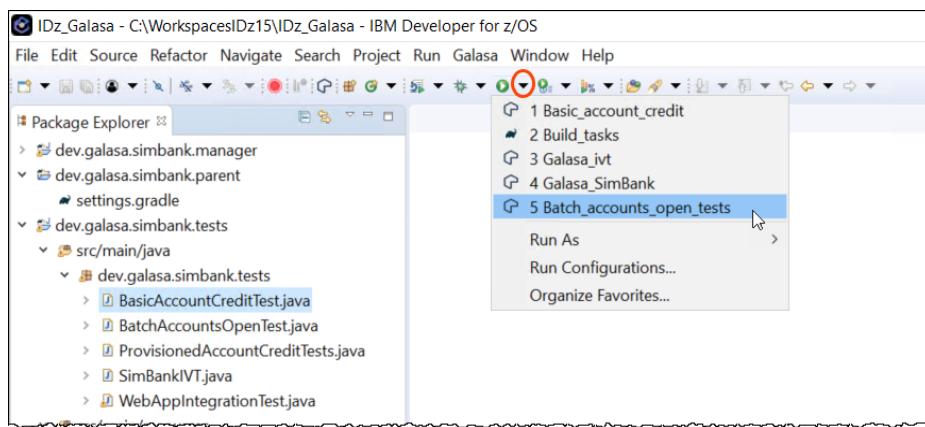
- 8.12 ► Use **Ctrl + Shift + F4** to close all opened editors.

PART #3 – Execute a Batch test

Section 9 Execute a batch test within the Galasa framework

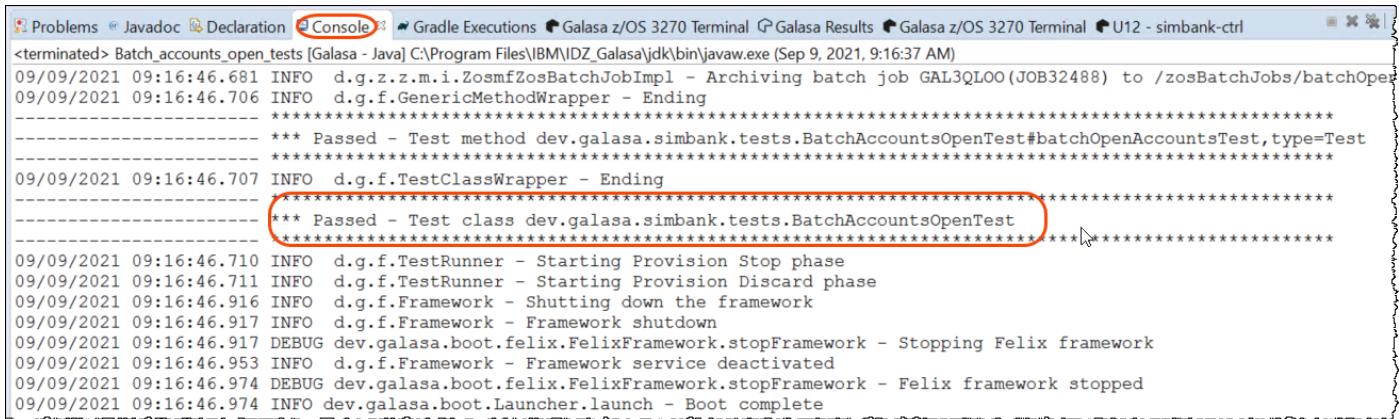
This test executes a sample JCL Batch job against the z/OS Application to load a set of accounts into the application.

- 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.

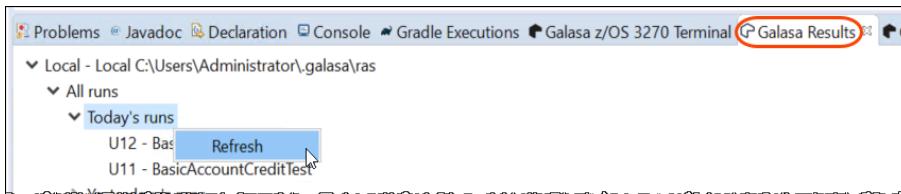
▶ Using the **Console** view you will see the message as below:



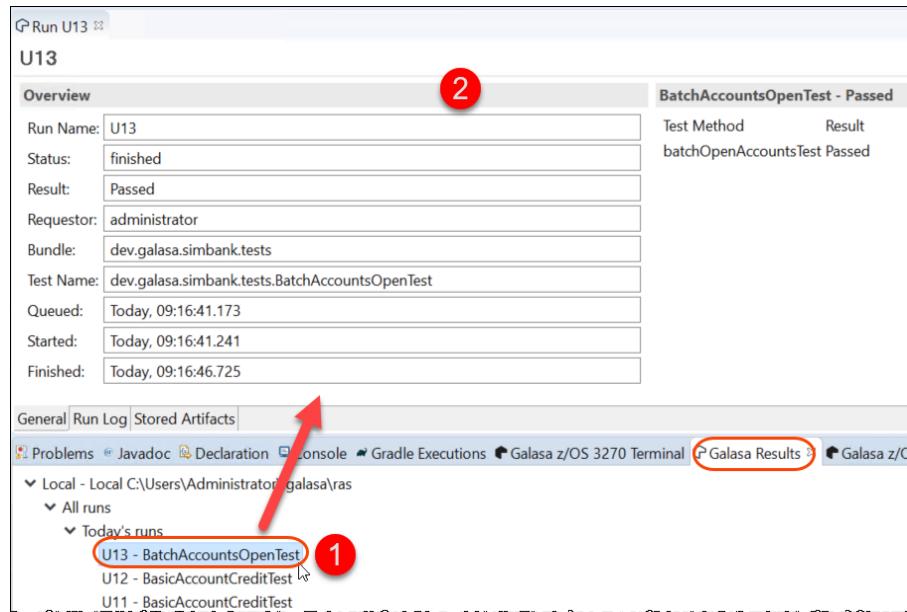
The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays log messages from a Java application named 'Batch_accounts_open_tests'. The log includes several 'INFO' and 'DEBUG' level messages related to the execution of a batch job and the stopping of a Felix framework. A red box highlights the line '*** Passed - Test class dev.galasa.simbank.tests.BatchAccountsOpenTest'.

```
<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 GAL3QL00(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
-----
***** Passed - Test class dev.galasa.simbank.tests.BatchAccountsOpenTest
*****
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 two views in the Eclipse interface. The top view is 'Run U13' showing details for a run named 'U13'. The bottom view is 'Galasa Results' showing a list of runs. A red arrow points from the 'Run Log' tab in the Run U13 view up to the 'Galasa Results' tab in the bottom view. A red circle labeled '1' is on the 'Galasa Results' tab. Another red circle labeled '2' is on the 'U13' run entry in the Run U13 view.

Run U13

U13

Overview

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

BatchAccountsOpenTest - Passed

Test Method	Result
batchOpenAccountsTest	Passed

General **Run Log** **Stored Artifacts**

Galasa Results

Local - Local C:\Users\Administrator\galasa\ras

▼ All runs
 ▼ Today's runs
 U13 - BatchAccountsOpenTest **1**
 U12 - BasicAccountCreditTest
 U11 - BasicAccountCreditTest

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 left side shows the "Package Explorer" view with a tree structure of Java packages and files. A red arrow points to the file "BatchAccountsOpenTest.java" which is highlighted with a red circle. The right side shows the code editor with the following Java code:

```
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
11     @ZosImage(imageTag = "SIMBANK")
12     public IZosImage image;
13
14     @ZosBatch(imageTag="SIMBANK")
15     public IZosBatch zosBatch;
16
17     @ZosBatchJobname(imageTag="SIMBANK")
18     public IZosBatchJobname zosBatchJobname;
19
20 }
```

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 code editor with the file "BatchAccountsOpenTest.java" open. A red box highlights the following code block:

```
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("901000001,20-40-60,1000");
60         accountList.add("901000002,20-40-60,1000");
61         accountList.add("901000003,20-40-60,1000");
62         accountList.add("901000004,20-40-60,1000");
63         accountList.add("901000005,20-40-60,1000");
64         accountList.add("901000006,20-40-60,1000");
65         accountList.add("901000007,20-40-60,1000");
66         accountList.add("901000008,20-40-60,1000");
67         accountList.add("901000009,20-40-60,1000");
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         // Load the JCL with the defined substitution parameters
75         IZosBatchJobname zosBatchJobname = null;
76         try {
77             zosBatchJobname = zosBatch.createJobname("TESTJOB", parameters);
78         } catch (ZosBatchException e) {
79             e.printStackTrace();
80         }
81         zosBatchJobname.submit();
82     }
```

A red arrow points to the bottom right corner of the code editor window.

- 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

U13

Stored Artifacts

- > framework
- zosBatchJobs
 - BatchOpenAccountsTest
 - DAL3QLOO_JOB32488_CC-0000
 - GAL3QLOO_JOB32488_JES2_JESJCL
 - GAL3QLOO_JOB32488_JES2_JESMSLG
 - GAL3QLOO_JOB32488_JES2_JESYMSG
 - GAL3QLOO_JOB32488_SIMBANK_SYSOUT
 - GAL3QLOO_supplied_JCL

Note that all of the spool files from the batch job were pulled back from z/OS including the *JESJCL*, *JESMSLG*, *JESSYMSG* 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

Run U13 BatchAccountsOpenTest.java Run U13 DAL3QLOO_JOB32488_SIMBANK_SYSOUT

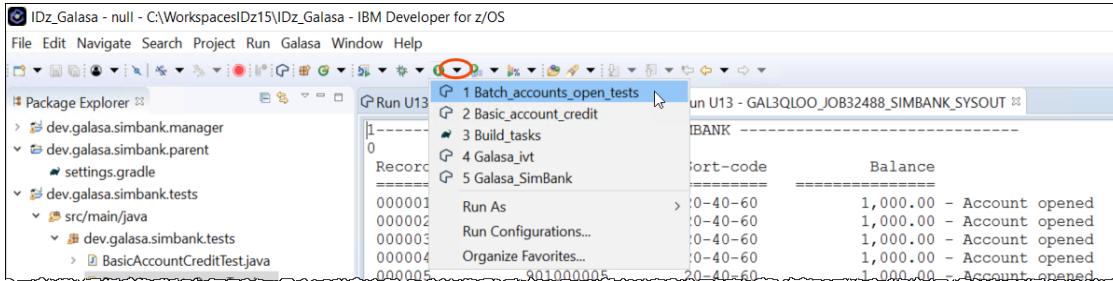
SIMBANK			
Record Number	Account Number	Sort-code	Balance
000001	901000001	20-40-60	1,000.00 - Account opened
000002	901000002	20-40-60	1,000.00 - Account opened
000003	901000003	20-40-60	1,000.00 - Account opened
000004	901000004	20-40-60	1,000.00 - Account opened
000005	901000005	20-40-60	1,000.00 - 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.

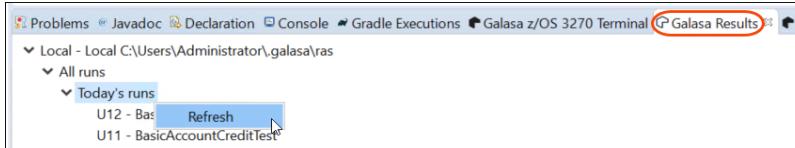
► Using the **Console** view you will see the message as below:

Note that the test **fails** because the accounts already exist and cannot be created again.

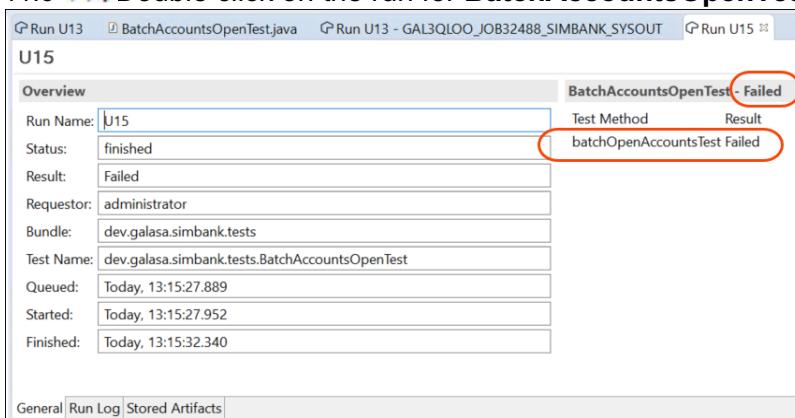
```
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:90)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:55)
at java.lang.reflect.Method.invoke(Method.java:508)
at dev.galasa.boot.felix.FelixFramework.runTest(FelixFramework.java:220)
at dev.galasa.boot.Launcher.launch(Launcher.java:163)
at dev.galasa.boot.Launcher.main(Launcher.java:117)

09/09/2021 13:15:32.334 INFO d.g.f.TestClassWrapper - Ending
***** Failed Test class dev.galasa.simbank.tests.BatchAccountsOpenTest
09/09/2021 13:15:32.336 INFO d.g.f.TestRunner - Starting Provision Stop phase
09/09/2021 13:15:32.337 INFO d.g.f.TestRunner - Starting Provision Discard phase
09/09/2021 13:15:32.649 INFO d.g.f.Framework - Shutting down the framework
09/09/2021 13:15:32.649 INFO d.g.f.Framework - Framework shutdown
09/09/2021 13:15:32.650 DEBUG dev.galasa.boot.felix.FelixFramework.stopFramework - Stopping Felix framework
09/09/2021 13:15:32.682 INFO d.g.f.Framework - Framework service deactivated
09/09/2021 13:15:32.700 DEBUG dev.galasa.boot.felix.FelixFramework.stopFramework - Felix framework stopped
09/09/2021 13:15:32.701 INFO dev.galasa.boot.Launcher.launch - Boot complete
```

11.5 ► As before in the **Galasa Results** view, right click on **Today's runs** and select **refresh** to load the latest test

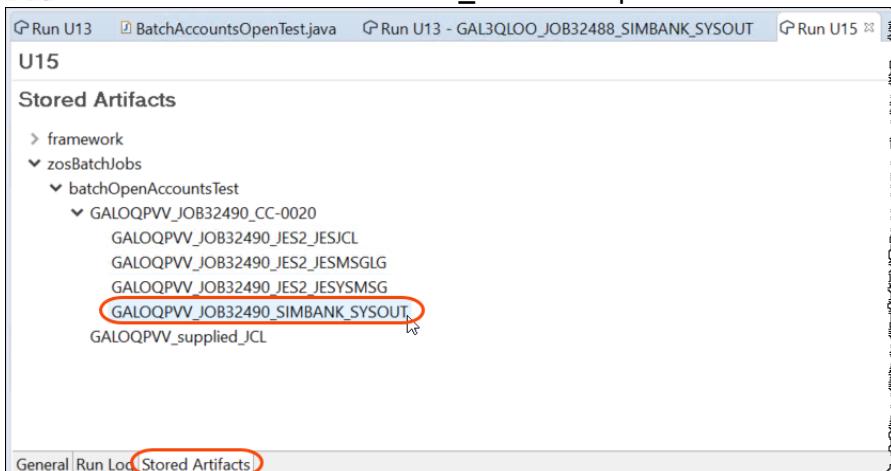


11.6 ► Double click on the run for **BatchAccountsOpenTest** and examine the results

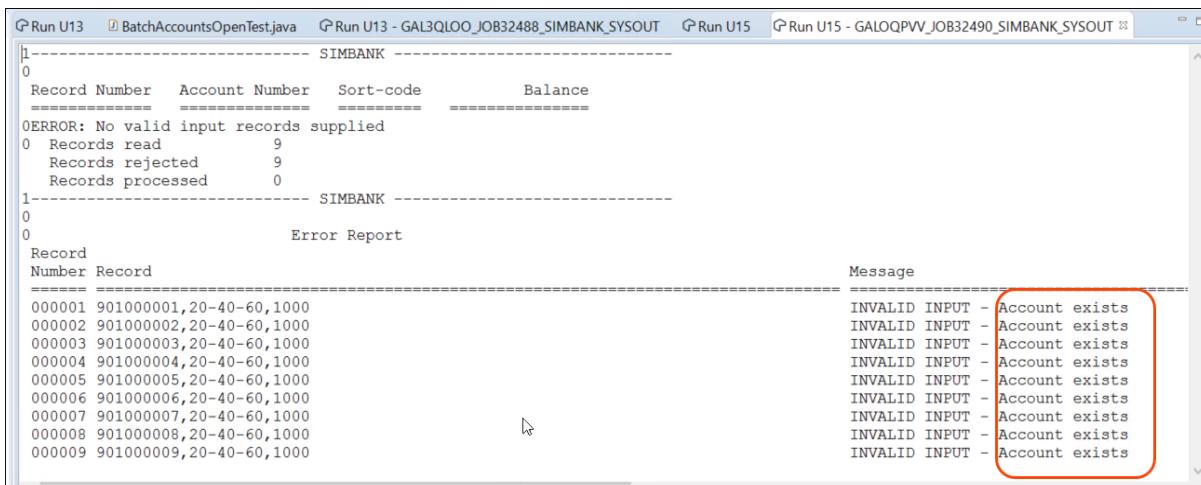


11.7  Open the **Stored Artifacts** tab of the Run Editor and expand the **zosBatchJobs** element within the view

▶ Double click on the **SIMBANK_SYSOUT** spool file



11.8 See the results below.



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>

LAB 10 – (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:

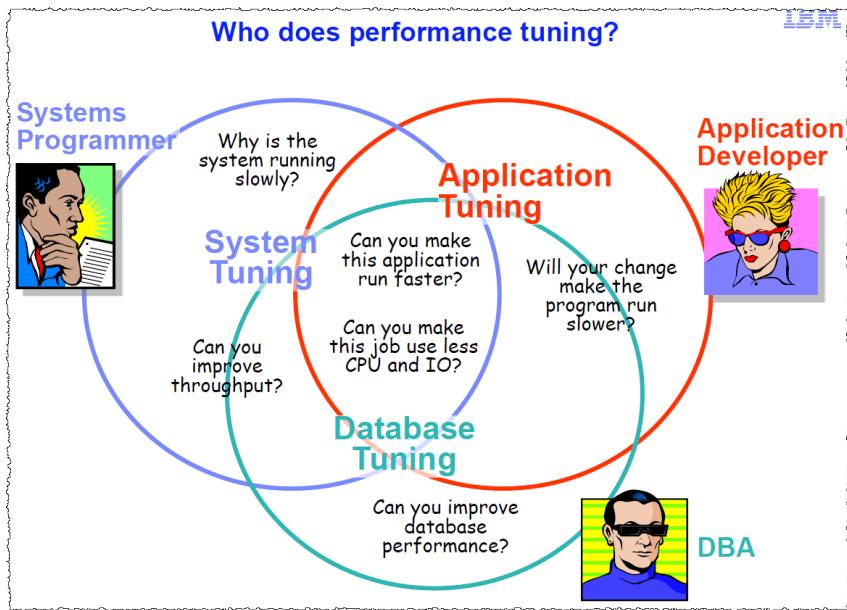
- 1. Create a new observation request for a job that is not running yet**
→ Using ADFz you will create an APA observation request.
- 2. Run a sample batch job to collect performance data**
→ You will submit a JCL that will execute a COBOL/DB2 batch program and will collect performance data.
- 3. Review 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.

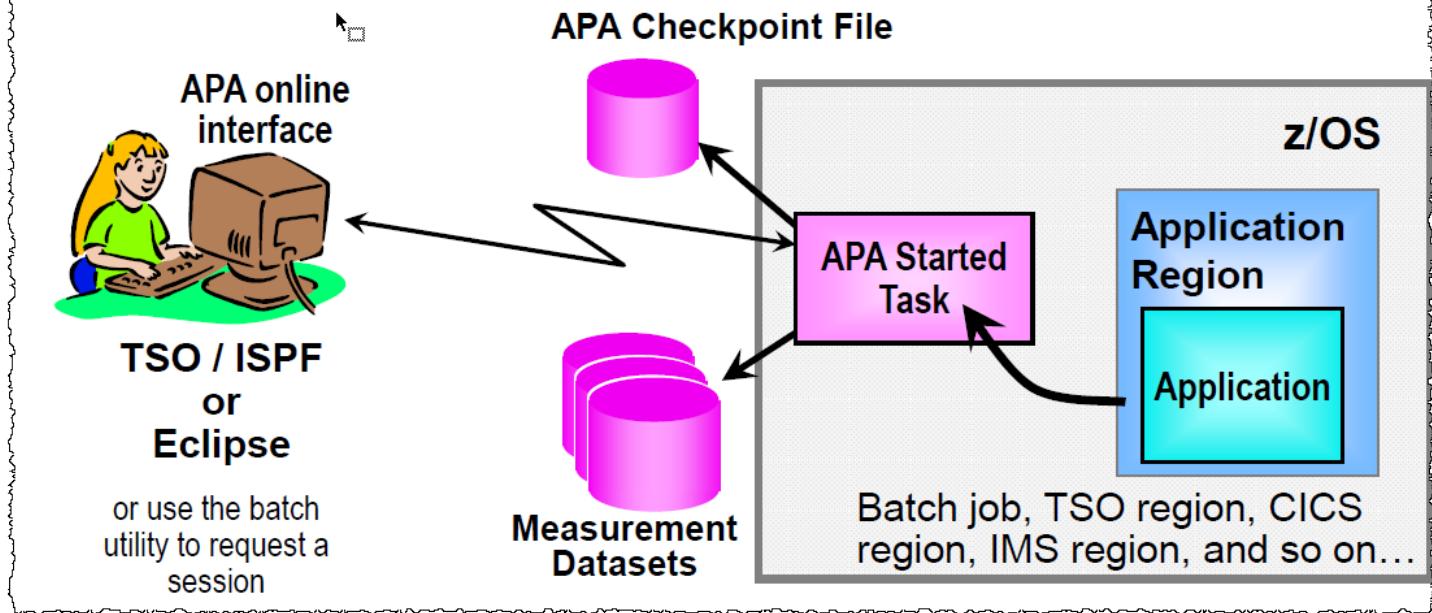


Section 1. Create a new observation request for a job that is not running yet

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.

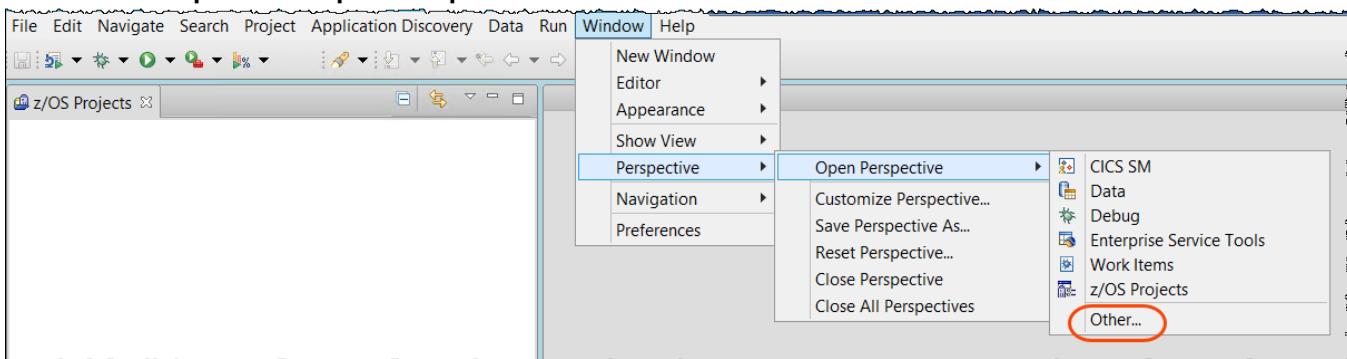
▶▶ Verify that the message indicates that it is Version 15.0.1

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.

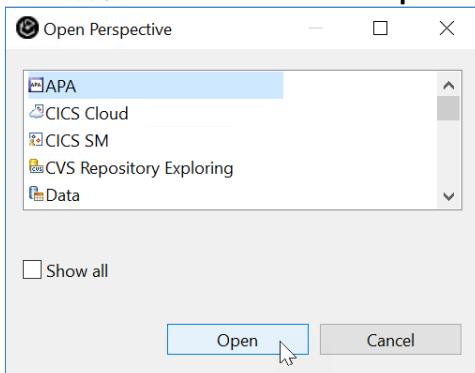


1.1.3 On other labs you may be used a different userid . Your new userid now will be **ibmuser**.

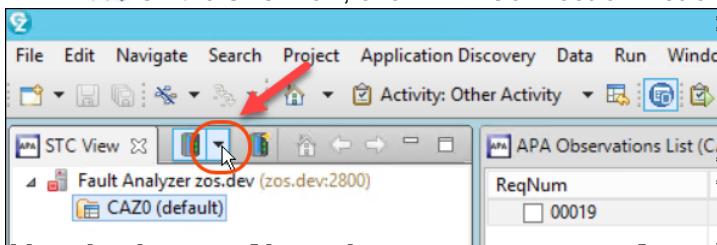
1.1.2 ► Open the APA perspective by selecting
Window > Perspective > Open Perspective > Other..



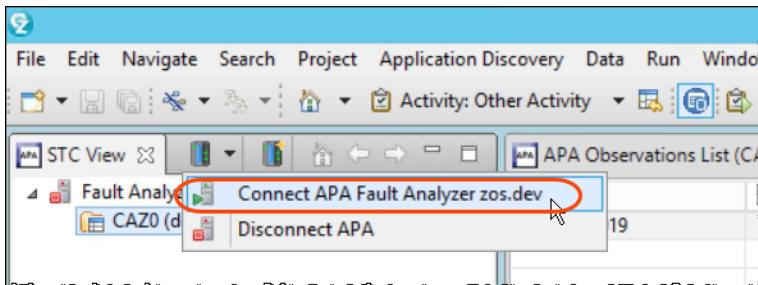
1.1.3 ► Select APA and click Open



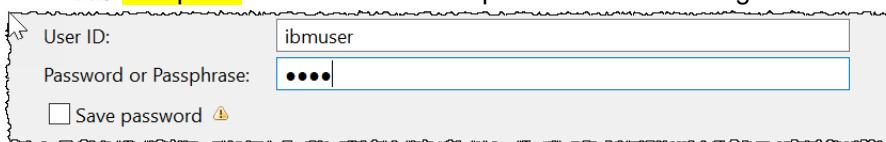
1.1.4 ► On the STC View, click APA Connection Actions icon



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.

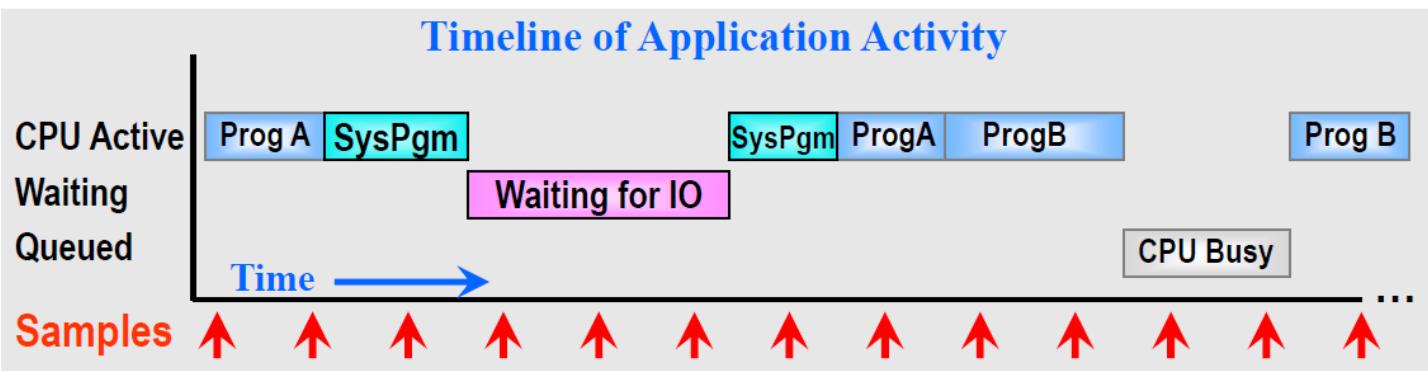
1.2 Begin a new APA observation session

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)

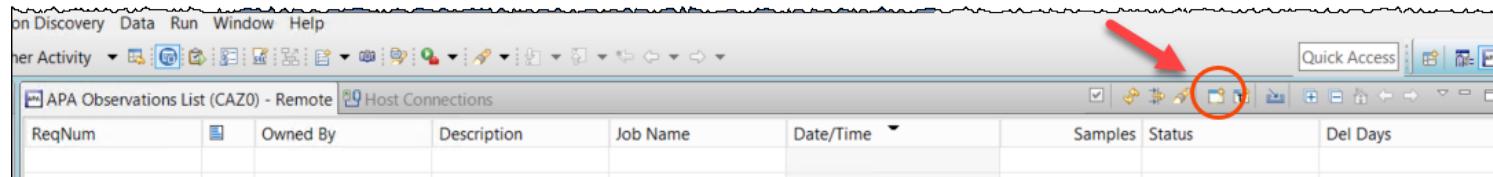
APA uses "Sampling"

- 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**.

Schedule New Measurement

① Enter the measurement information and click 'Submit' to schedule.

Job Information Options Multi Steps Active Jobs Subsystems Schedule Sched Options

System SOW1

Job Name/Pattern APA05 Get Active Jobs ASID

Inactive

Step Specification

Step Number Specify step number, program, step name or step name + proc step name. Use 'Multi Steps' tab to specify more than one step

Program Name

Step Name

Proc Step Name

Description

Number of Samples 5000 Measure to step end

Duration (min:sec) 1:00 Delay by (secs)

Notify TSO User Retain file for (days) 30

USS observations Max. 10

?

Load Preview Cancel Save 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	EMPTO1			APA05	Apr-28-2021 12:47	5,000	Sched	30	SOW1

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

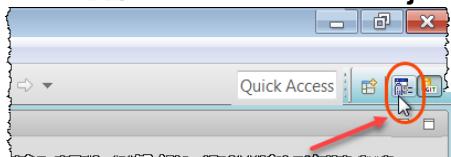
You will submit a COBOL/DB2 batch job named **APA05** and let APA collect performance data.

2.1 Connect to z/OS

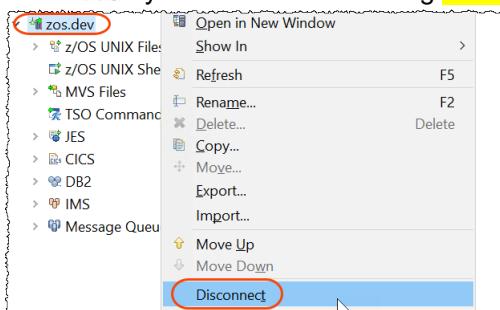
To submit a job, you first need to connect to z/OS using your assigned user id.

If you are already connected to z/OS using userid **ibmuser**, skip this and go to step 2.2.

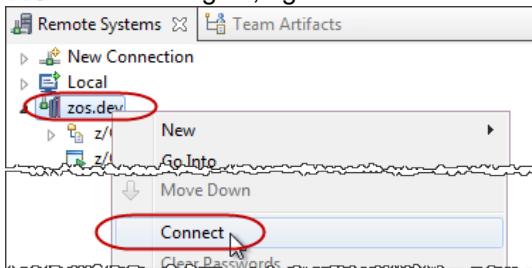
- 2.1.1 ► Switch to the **z/OS Projects** perspective clicking on icon  on the top right corner



- 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**

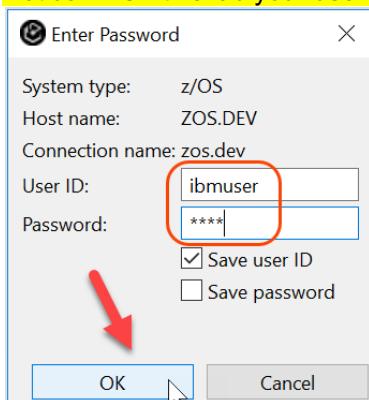


- 2.1.3 ► Type **ibmuser** as userid and **sys1** as password.

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)

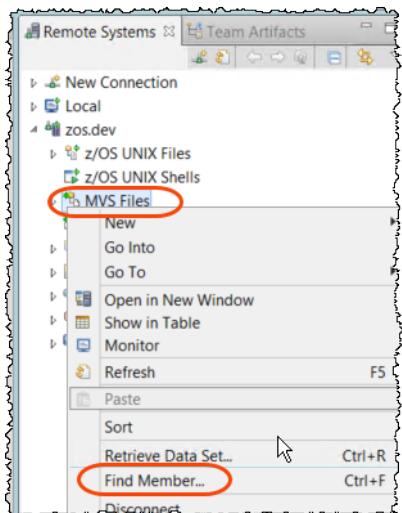


2.2 Find the JCL to be submitted

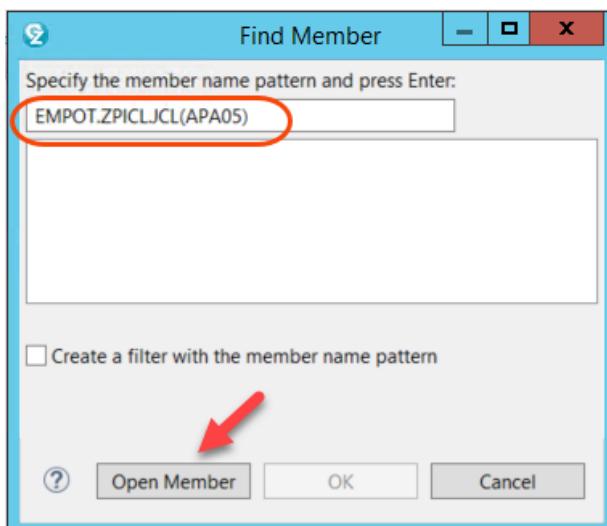
You will find a JCL that runs a COBOL/DB2 batch program.

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**.



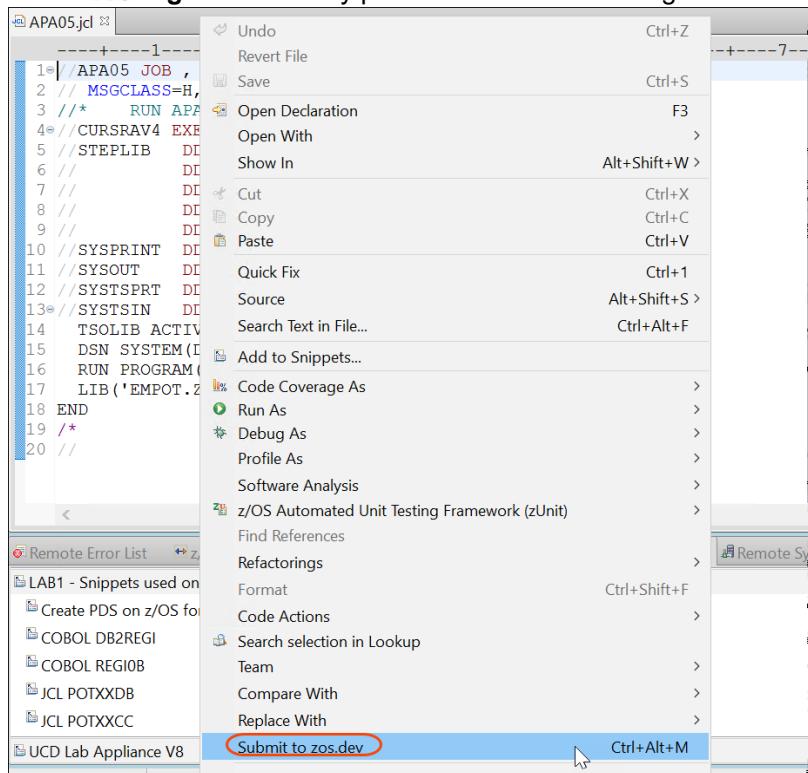
2.3 Submit the JCL to execute the COBOL/DB2 batch program

The member **APA05.jcl** will open and you must submit for execute on z/OS.

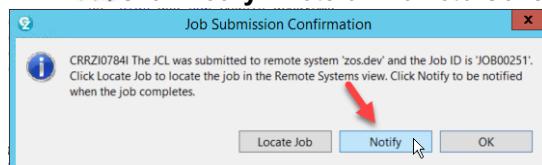
The screenshot shows the IBM i Navigator interface. On the left, a JCL editor displays a job named APA05.jcl. The code includes various DD statements for datasets like IBMUSER.POT.LOAD, DSNSB10.SDSNLOAD, and DSNSB10.DBDBG.RUNLIB.LOAD, along with system output and error definitions. It also contains TSO LIBRARY ACTIVATE and RUN PROGRAM commands. On the right, a 'Remote Systems' browser window is open, showing a tree view of files under the directory EMPOT.ZPICL.JCL, including sub-directories like ...more and numerous JCL files such as APA05.jcl, APA05OPT.jcl, etc.

```
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=IKJEF01,DYNAMNBR=20  
5 //STEPLIB DD DSN=IBMUSER.POT.LOAD,DISP=SHR  
6 // DD DSN=EMPOT.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 //SYSTSIN DD *  
14 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')  
15 DSN SYSTEM(DBDBG)  
16 RUN PROGRAM(APA05) PLAN(APA05) -  
17 LIB('EMPOT.ZPICL.LOAD')  
18 END  
19 /*  
20 //
```

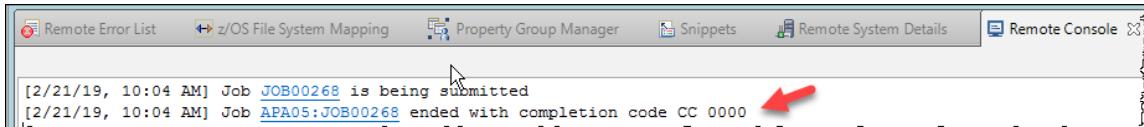
2.2.1 ► Right click in any place of the member being edited and select **Submit to zos.dev**



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.

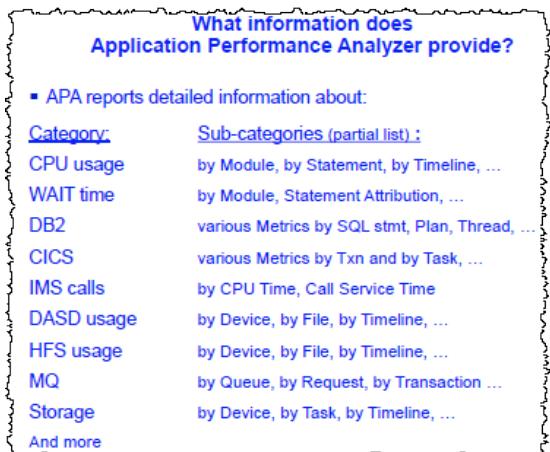


2.2.4 ► Use **Ctrl + Shift + F4** to close all opened editors.

What have you done so far?

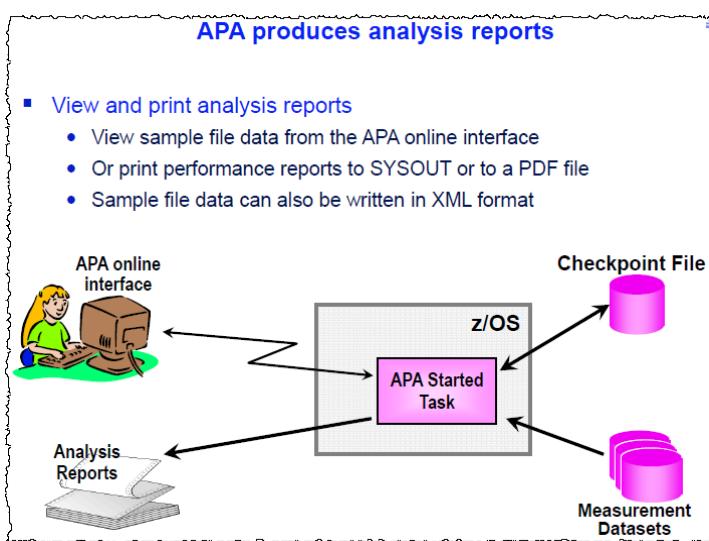


You created an APA observation request for a job that you have submitted. The observation collected less than 5000 samples since it run in less than 1 minute.



Section 3 – Review some of the reports created.

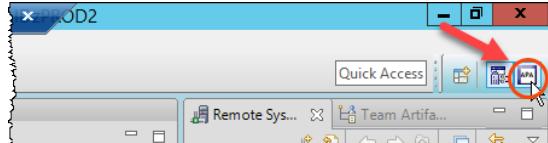
APA produces analysis reports. On this section you will explore some of the reports produced. Notice that you could print, create a PDF or export in XML format.



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

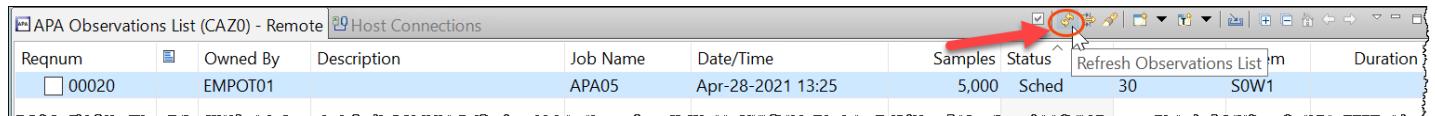
The reports list is a 2-level tree-view. The first level (parent) rows represent the report category while the second-level (child) rows are for the individual reports.



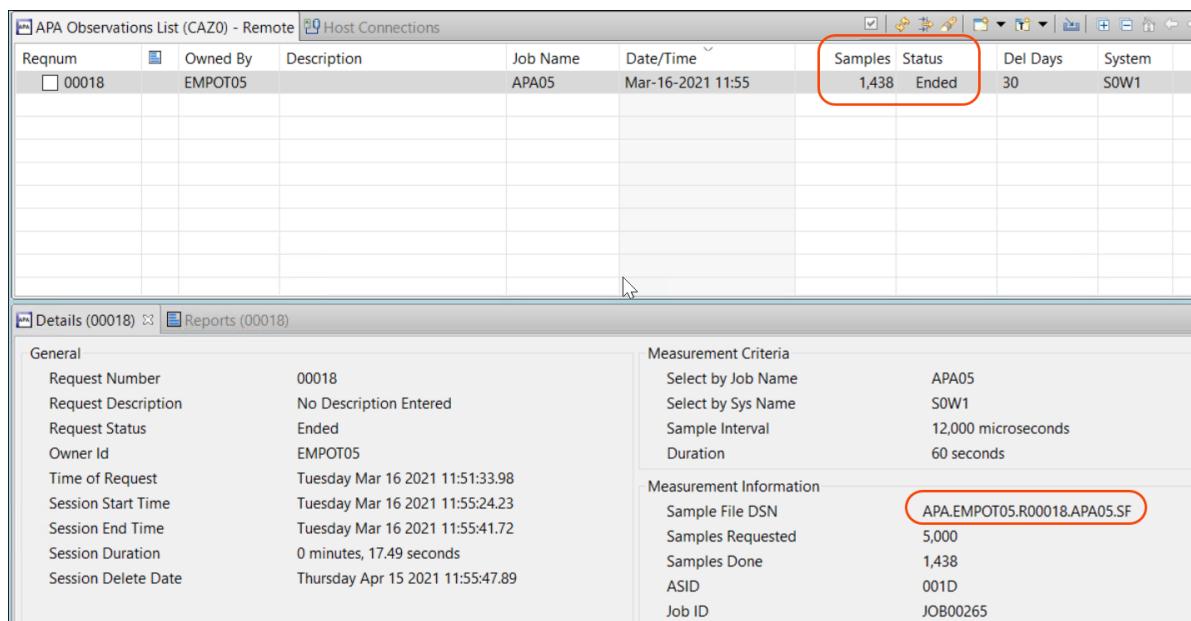
The category rows are for informational/organizational purposes only and result in no action if clicked (other than if the category is expanded). A sticky note icon  is displayed on each report row which has a sticky note.

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..



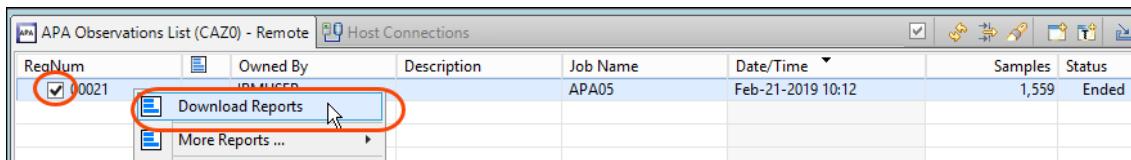
Reqnum	Owned By	Description	Job Name	Date/Time	Samples	Status	Del Days	System
00018	EMPOT05		APA05	Mar-16-2021 11:55	1,438	Ended	30	SOW1

Details (00018)

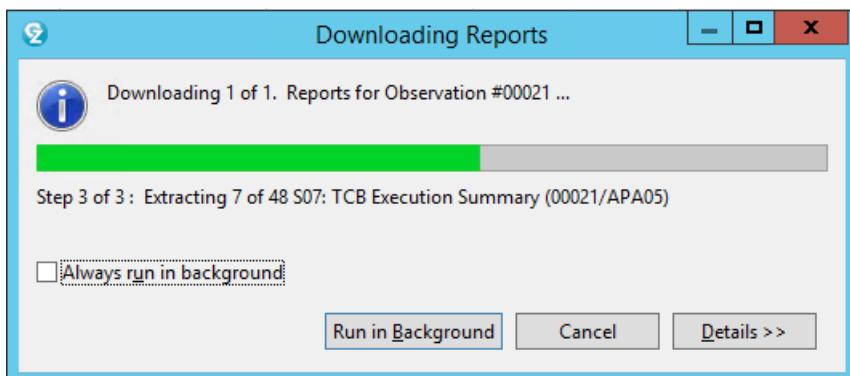
General		Measurement Criteria	
Request Number	00018	Select by Job Name	APA05
Request Description	No Description Entered	Select by Sys Name	SOW1
Request Status	Ended	Sample Interval	12,000 microseconds
Owner Id	EMPOT05	Duration	60 seconds
Time of Request	Tuesday Mar 16 2021 11:51:33.98	Measurement Information	
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).

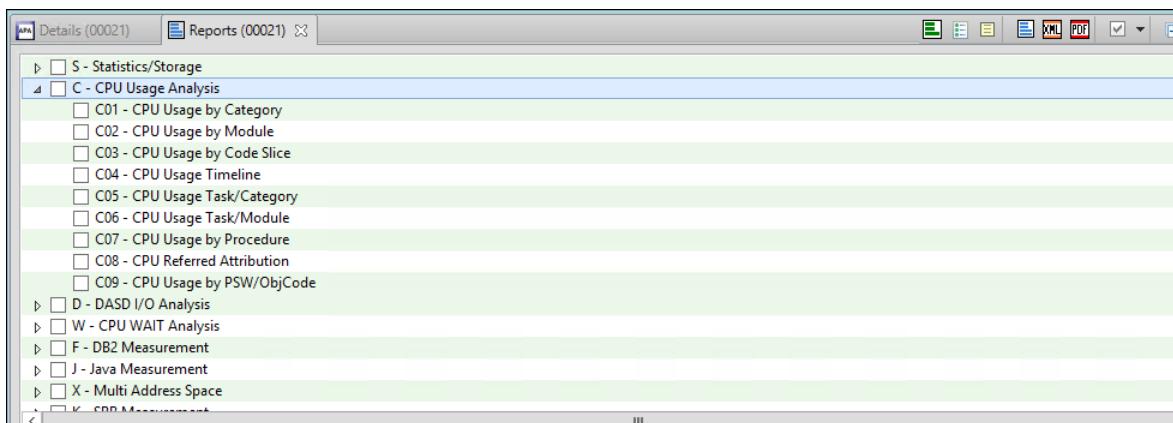
3.1.3 ► Click on the box  (on left) to select the observation, **Right click** on observation and select **Download Reports**. This will download the created reports to your workspace.



This dialog below may or may not be displayed. Notice that this operation may take a while, depending on your system, network, etc..

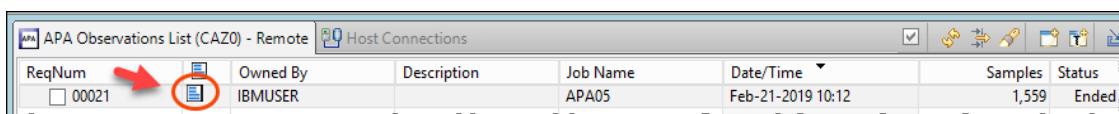


When completed you will have the *Reports* view populated.



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



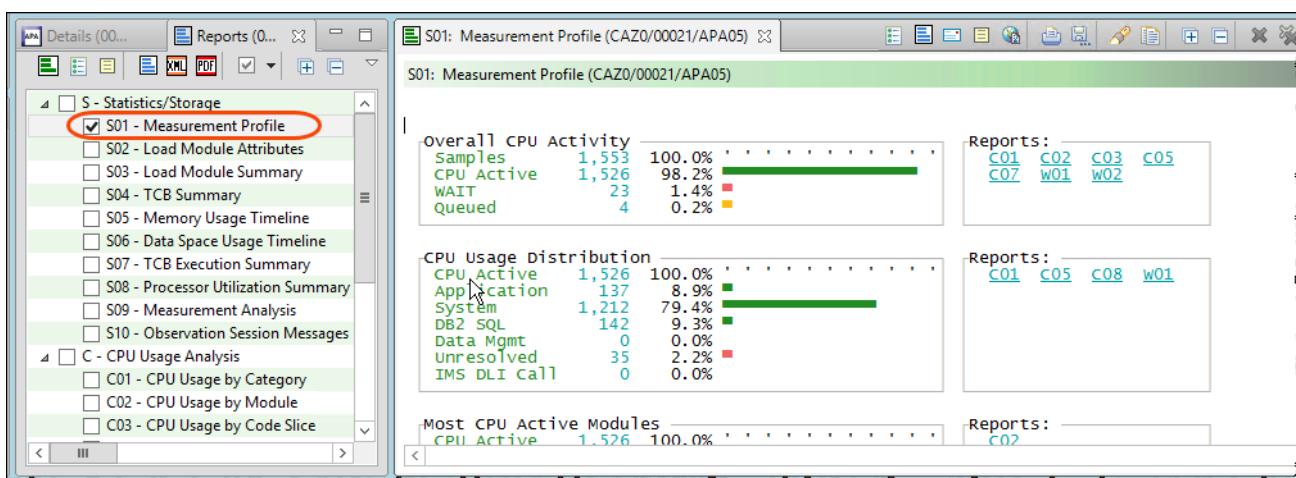
3.2.2 ► Double click on the S01 Measurement profile report. The *Measurement Profile report* is displayed.

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.

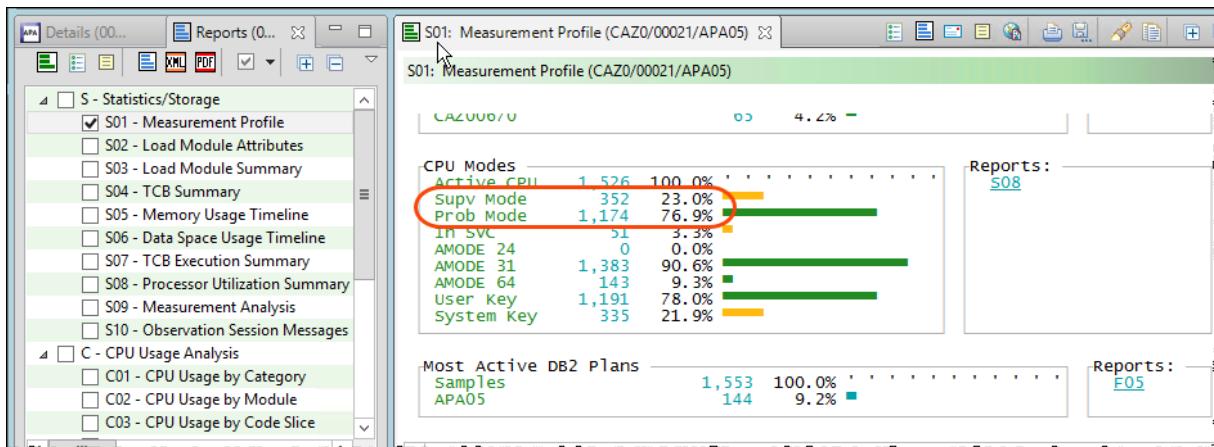
Remember that the actual statistics shown on your reports may be entirely different than the examples shown here. In the example below, the CPU was active for about 98.2% of the samples. An additional 1.4% of the samples show the application in a WAIT state, waiting for resources to become available.

Notice the next 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**.



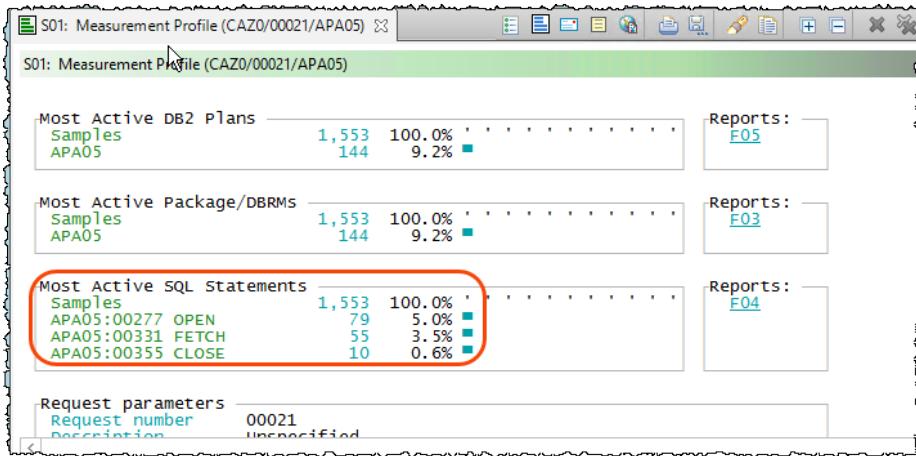
3.2.3 ► Scroll down to the remainder of the S01 report.



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.

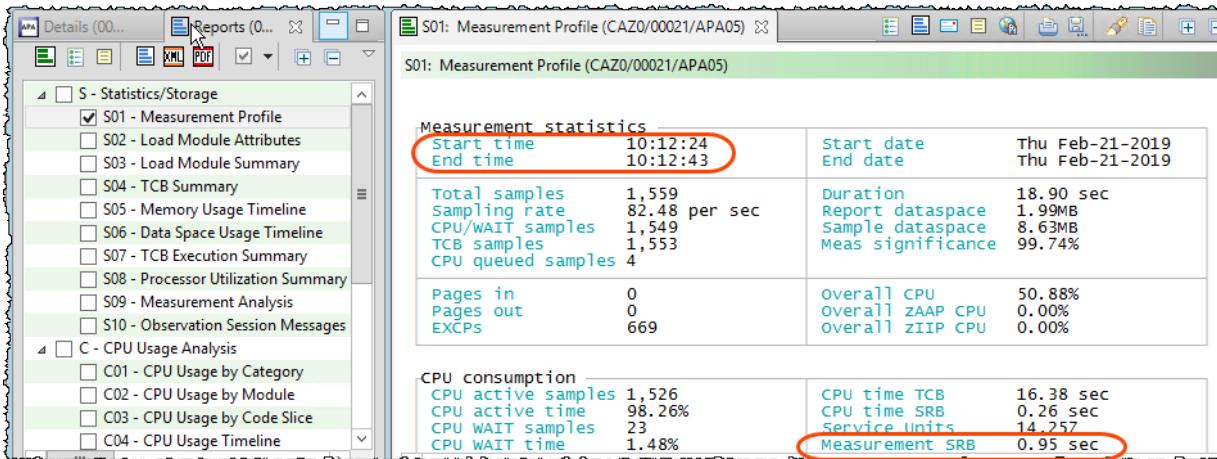
Usage of zIIP and zAAP processors



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		
CAZ00202	CSA	1D763C88	1	4,984	RU RN	STEPLIB	APA.SCAZAUTH
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		
CEEPLPKA	PLPA	05D48000	1	2,182,416			
COFMRETR	JPA	204221D8	1	7,720	RU RN		
COVMRC	JPA	A171A1A	1	648			

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		
CAZ00202	CSA	1D763C88	1	4,984	RU RN	STEPLIB	APA.SCAZAUTH
CAZ00650	CSA	1D73E490	1	2,688	RU RN		
CAZ00670	CSA	1D63BEF0	1	4,368			

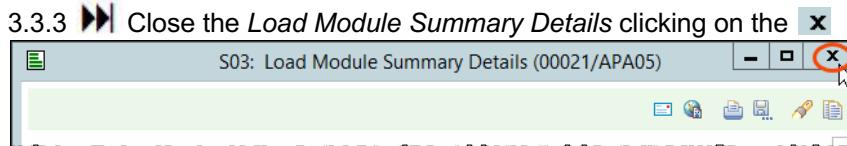
You may find interesting information as seen below, like which COBOL version the program was compiled, the compile options (like NOOPT – NO Optimizer), etc.....

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
+000000 APA05
+000000 APA05

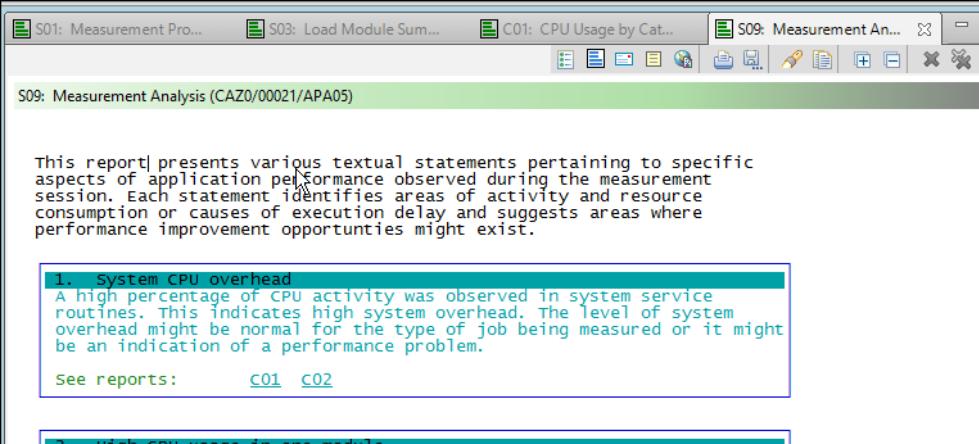
Compile Options:
ADV AFP(VOLATILE) QUOTE ARITH(COMPAT) NOAWO NOCICS NOCMPR2
NOCURR DATA(31) DBCS NODECK DISPSIGN(COMPAT) NODLL DP NODUMP
NODYNAM NOEXP NOFASTRT HGRP(PRESERVE) INTDATE(ANSI) LIB LIST
NOMAP NOMDECK NONAME NONUM NUMCLS(PRIM) NUMPROC(NOPFD) OBJ
NOOFFSET NOOPT NOOPTFILE PGNAME(CO) RENT RMODE(ANY) SEQ SIZE
SOURCE SQL SUCCSID NOSSR NOSTGOPT NOTERM TEST(SOURCE)
NOTHREAD TRUNC(STD) NOVBREF NOWORD XP(XMLSS) XREF ZWB



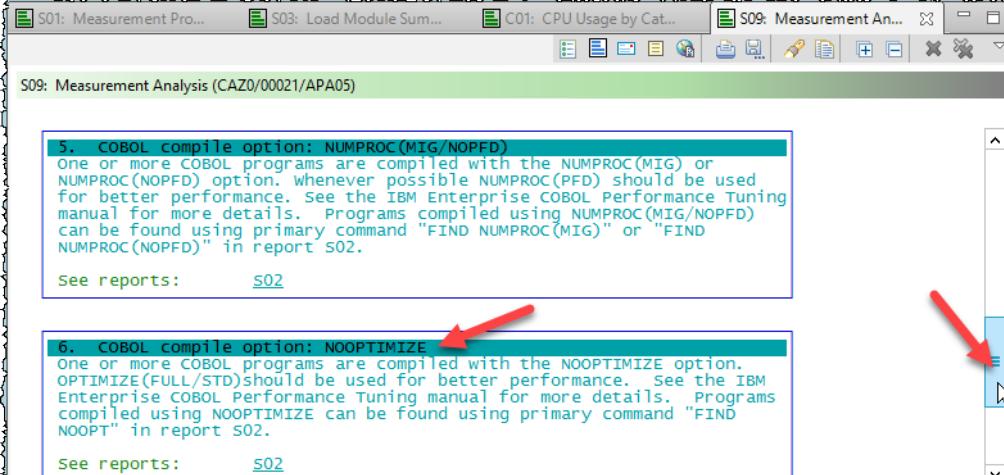
3.4 Measurement Analysis report (S09)

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.



- 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.



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...

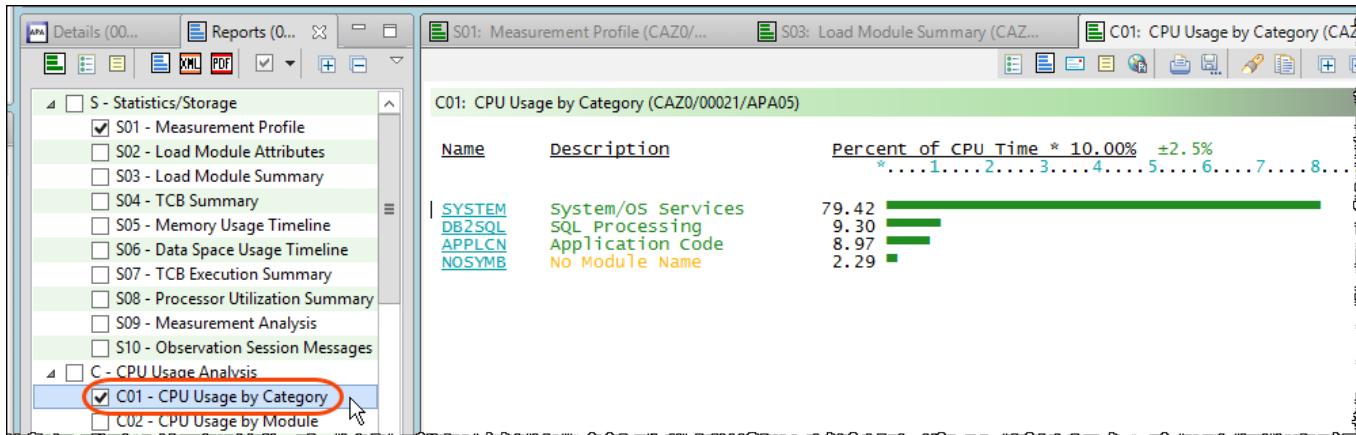
The job used to compile with **OPTIMIZE(2)** is at **EMPOT.ZPICL.JCL(APACL05O)**.

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.

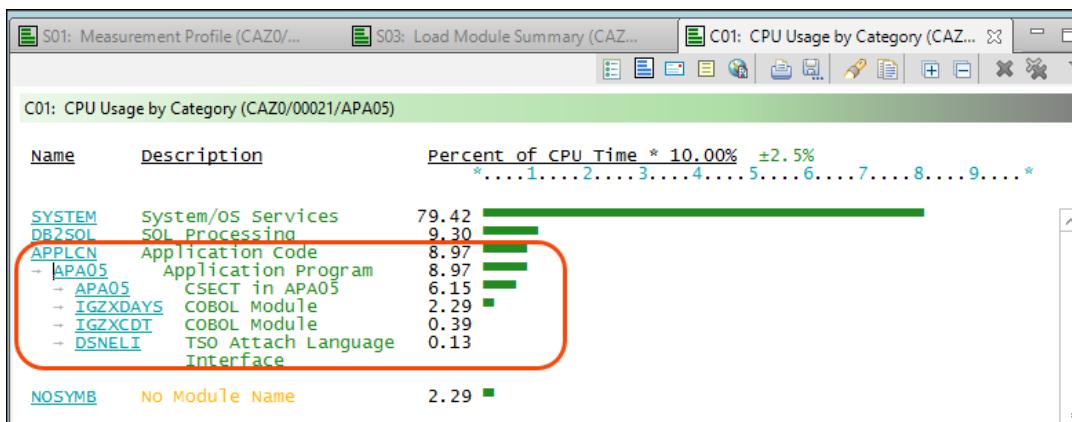


3.5.2 ► Click on the **APPLCN** collection to see the names of the Load Modules which appear in the sampling

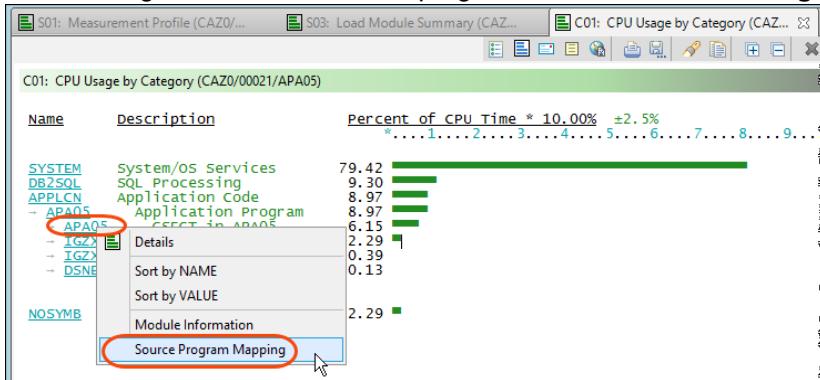
The only Load Module in the **APPLCN** collection is the **APA05** Load Module. In this example **APA05** consumed all of the CPU resource attributed to the **APPLCN** collection.

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.

The screenshot shows the 'C01: CPU Usage by Category Source Program Mapping (00021/APA05)' window. It displays source code lines with their corresponding offsets and counts. Red annotations highlight specific lines, such as line 000279 with a count of 78, which is attributed to the previous statement.

LineNo	Offset	Count	Source Statement
000276			* ROW FETCH PROCESSING.
000277	0003E4		EXEC SQL OPEN C1
000278	000442		END-EXEC.
000279	000468	78	<- CPU time attributed to above statement 150-EXIT. EXIT.
000280			17 line(s) not displayed
000298	000566		MOVE HOURS-TBL-MAX TO HOURS-RPT-MAX
000299	00058E		MOVE HOURS-TBL-MIN TO HOURS-RPT-MIN
000300	000566		DISPLAY *DEPTL DEPT-TBL
000301	000606	1	7 <- CPU time attributed to above statement 1 DISPLAY *PERF-AVG* PERF-TBL-AVG
000302	000656	2	2 <- CPU time attributed to above statement 2 DISPLAY *PERF-MIN* PERF-TBL-MIN
000303	0006A6	7	7 <- CPU time attributed to above statement 7 DISPLAY *PERF-MAX* PERF-TBL-MAX
000304	0006F6	2	2 <- CPU time attributed to above statement 2 DISPLAY *HOURS-AVG* HOURS-TBL-AVG

3.5.5 ► Scroll up to the end

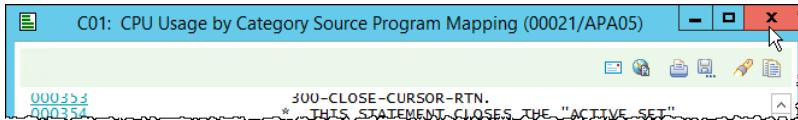
APA also indicates when system level activity is taking place on behalf of a source statement. In this example, the Count column carries a high number and the statement under the **MOVE FUNCTION** statement (727) indicates that the system is executing logic on behalf of the **MOVE FUNCTION CURRENT-DATE** statement.

The screenshot shows the 'C01: CPU Usage by Category Source Program Mapping (00021/APA05)' window. The scroll bar is at the bottom right. A circled '727' is shown next to a MOVE FUNCTION CURRENT-DATE statement, indicating it is being executed on behalf of another statement. A red arrow points to the scroll bar.

LineNo	Offset	Count	Source Statement
000353			300-CLOSE-CURSOR-RTN.
000354			* THIS STATEMENT CLOSES THE "ACTIVE SET"
000355	000CB8	1	EXEC SQL CLOSE C1 END-EXEC.
000356			300-EXIT.
000357	000D3C		EXIT.
000358			350-TERMINATE-RTN.
000359	000D42		MOVE ROW-KTR TO ROW-STAT.
000360	000D6C		DISPLAY ROW-MSG.
000361		1	1 <- CPU time attributed to above statement
000362			* APA added - display # of DB2 open and fetch and close
000363			23 line(s) not displayed
000364			502-LOOP.
000365	00101C		MOVE 1 to IND.
000366	001028	45	PERFORM 504-ADD-COUNTER THRU 506-EXIT
000367			UNTIL IND = 900000.
000368			000390
000369			504-ADD-COUNTER.
000370	00105A	36	MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE.
000371	001080	727	11 <- CPU time attributed to above statement 11 COMPUTE WS-INTEGER-START-DATE =
000372		274	274 <- CPU time attributed to above statement FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
000373			COMPUTE IND = IND + 1.
000374			000395
000375	0011B6		0011B6

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

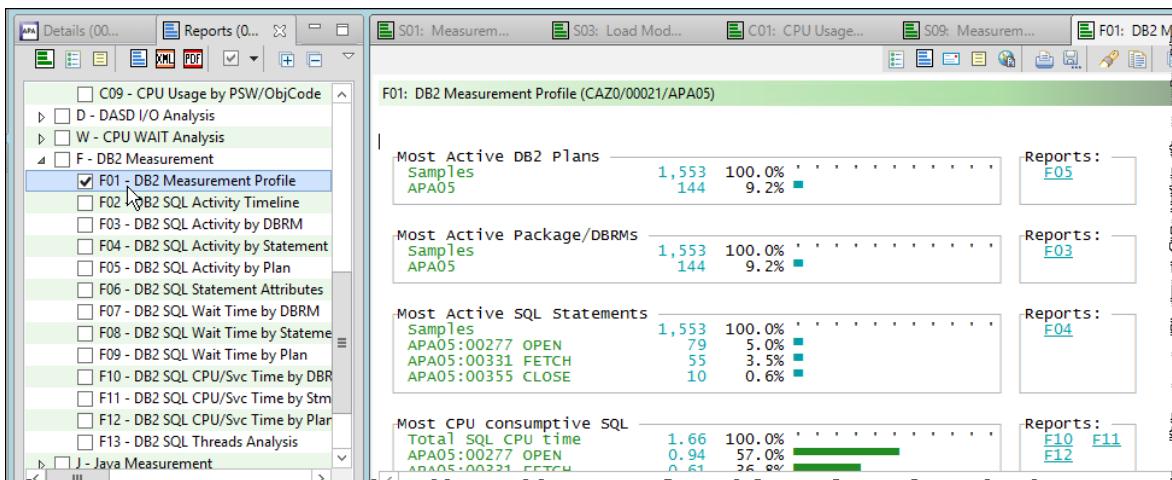


3.6 Analyzing the APA DB2 Measurement Profile report (F01)

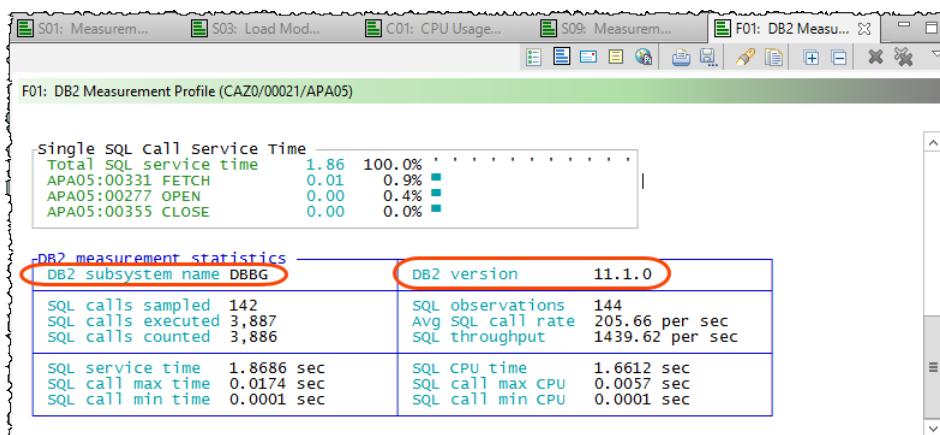
This is a good report to examine first when analyzing DB2 information. It provides an at-a-glance summary of various aspects of the measurement data and helps you choose which other reports to concentrate on. The first section of this report consists of a series of mini performance graphs illustrating 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	±	Time
S00003	APA05	277	OPEN	5.08	.	5.08
S00001	APA05	331	FETCH	3.54	.	3.54
S00002	APA05	355	CLOSE	0.64	.	0.64

3.7.2 Click S00003. You will see the details of this OPEN statement in the program APA05.

```
> 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
```

3.7.3 Right click S0003 and select Source Program Mapping

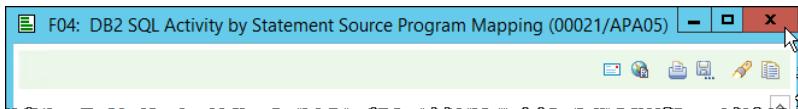
S0003 APA05 277 OPEN 5.08

- Details
- DB2 Explain SQL
- Module Information
- Source Program Mapping**

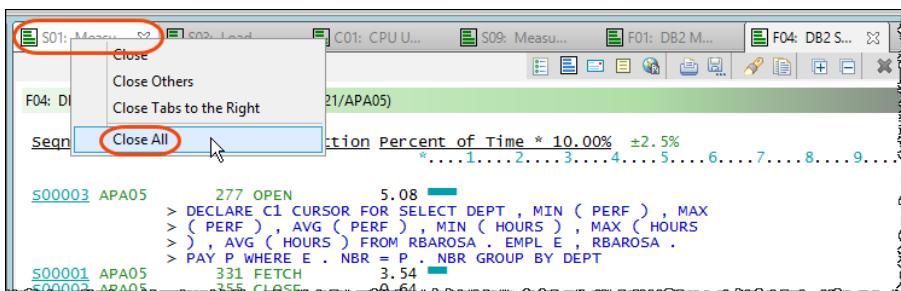
3.7.4 The reports shows where the Open is used in the program.

```
Load Module: APA05 LIB: 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  
000268 WHERE E.NBR = P.NBR  
000269 * AND PERF >:PERF  
000270 GROUP BY DEPT  
000271 END-EXEC.  
000272 100-EXIT.  
000273 EXIT.  
000274 150-OPEN-CURSOR-RTN.  
000275 * THIS STATEMENT OPENS THE "ACTIVE SET" IN PREPARATION OF  
000276 * ROW FETCH PROCESSING.  
000277 EXEC SQL OPEN C1  
000278 79 END-EXEC.  
000279 150-EXIT.  
000280 EXIT.  
000281 200-FETCH-RTN.  
000282 * THIS PARAGRAPH SETS UP THE SQL PARAMETERS, PERFORMS THE  
000283 * PARAGRAPH TO FETCH THE ROW, AND DISPLAYS THE RESULTS.  
000284 * ---> HINT <--- USE ISPF EXCLUDE (xx) OR BLOCK COPY  
000285 FROM YOUR CURSOR DECLARE STATEMENT TO VERIFY PROPER
```

3.7.5 ► Close the panel clicking on the 



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 10.



© 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.



Please Recycle
