Z Open Development
Proof of Technology (PoT)

Exercises Using zDT on Cloud

Updated August 26, 2019

Contents

Overview

- Discover and understand your Mainframe applications (ADDI)
- 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)
- 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 2019**
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 August 12 2019 (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

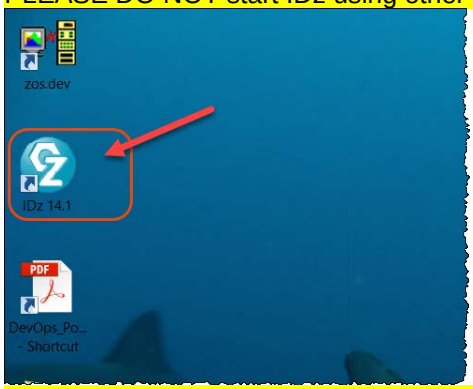
You will connect to the z/OS. This section is like LOGON to a TSO using a provided userid and password.

___ 1.1 Working with a defined z/OS Remote system connection

1.1.1 Start *IBM Developer for z Systems*.

⏭ Using the desktop double click on **IDz 14.1** icon.

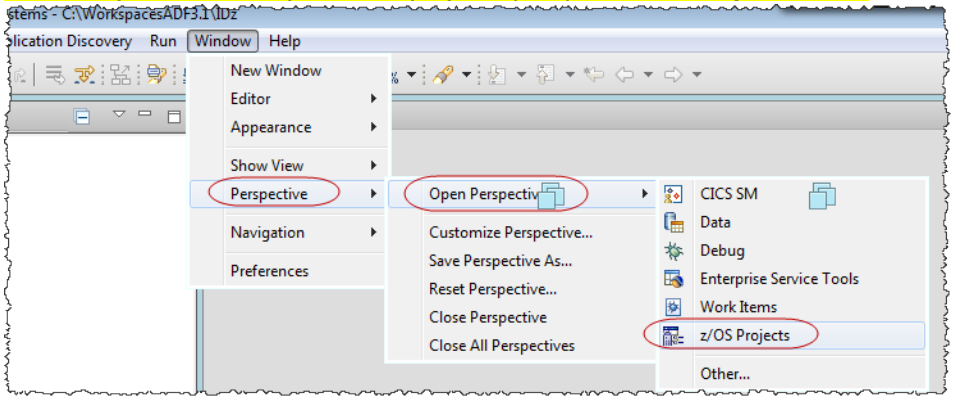
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.



Be patient, since you are using a cloud instance the first startup may take few minutes…

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

Notice: If you are already on *z/OS projects* perspective nothing will occur.



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.

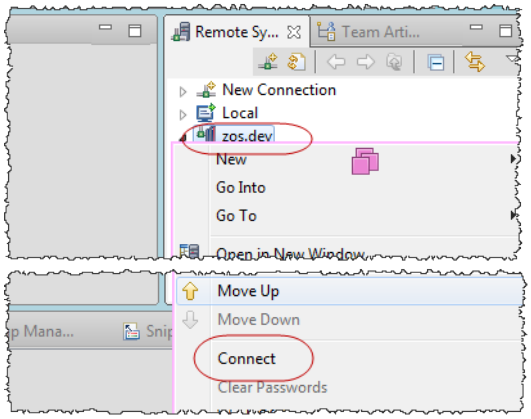
The z/OS Projects perspective contains the many views like Remote Systems view, z/OS Projects view, Properties view, Outline view, Remote Error List view, z/OS File System Mapping view, Remote System Details view, Team view, Property Group Manager view and Snippets view

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.

___ 1.2 Connecting to the z/OS Remote system

1.2.1 To Connect to the z/OS system:

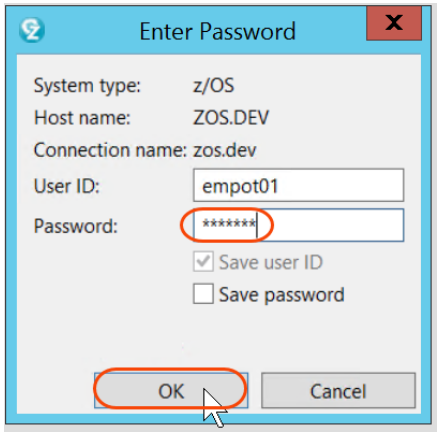
⏭ Using the *Remote Systems view* on the top and right side of the screen:
Right-click on **zos.dev** and select **Connect**
Notice: Since you are using a cloud instance the response to the right click may be delayed first time.



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

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

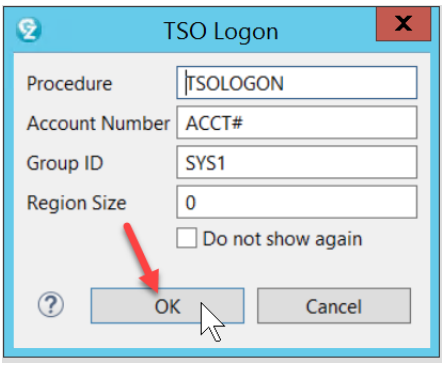
1.2.3 ⏭ Click **OK** to connect to z/OS.



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.
*Try to open a Windows command prompt and type **ping zos.dev**.*
If you do not receive a reply, Contact the instructor. Your connection is broken.

1.2.4 Click **OK** for the *TSO logon* dialog



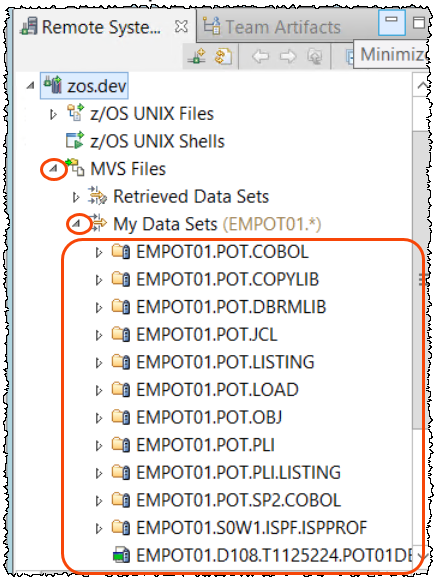
Be Patient! The connection could take a while depending on the network condition.

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

BEFORE…

AFTER:

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



Remote Systems view

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

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

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

What is a z/OS project?

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

Creating a new MVS subproject

MVS subprojects contain the development resources that reside on an MVS system. You can create multiple subprojects in a z/OS project.

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

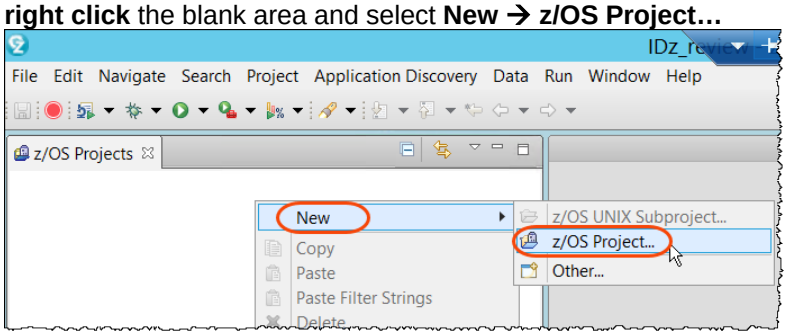
→ Connecting to a remote system

→ Creating a z/OS project

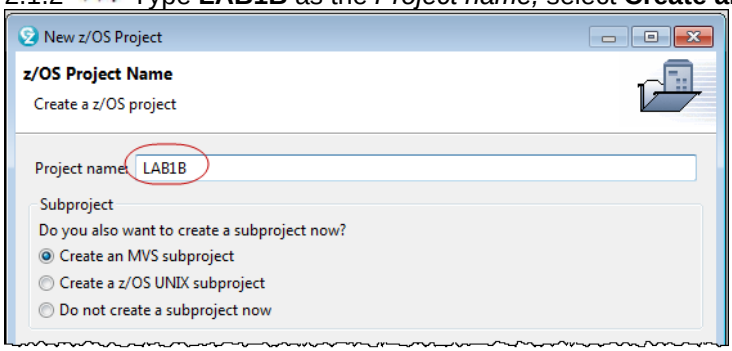
___ 2.1 Submit a COBOL/DB2 batch to execute

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

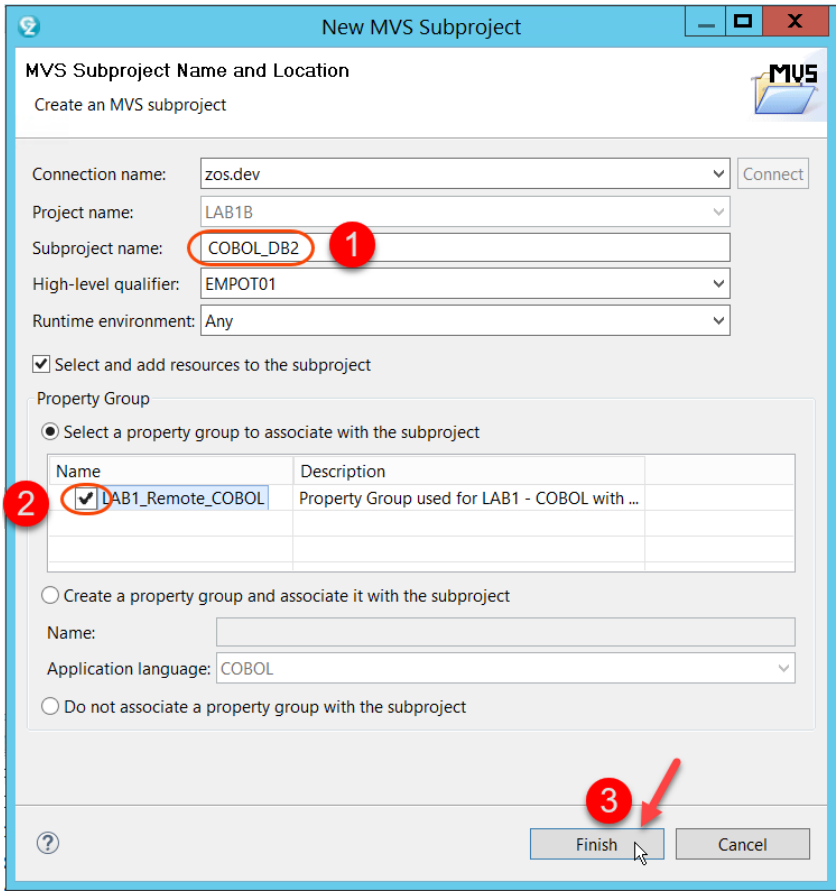
2.1.1 ⏭ Using the *z/OS Projects* view (on the top and left), **right click** the blank area and select **New → z/OS Project…**



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**



Can't you find the property group above?

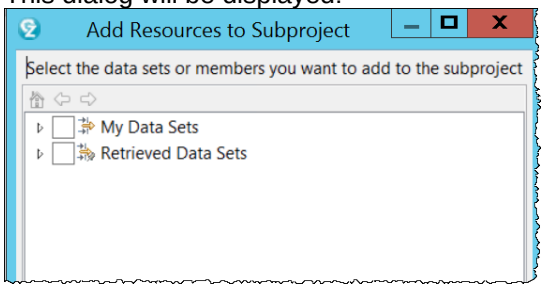
It is because you are using a wrong workspace. Close IDz, go back to the step 1.1.1 and be sure you start IDz from the icon that is on the windows desktop.

– PROPERTY GROUP

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

You can, for example, define a property group that points to copybooks. This is something like `//STEPLIB`; otherwise, some of the variables used in the program are not resolved.

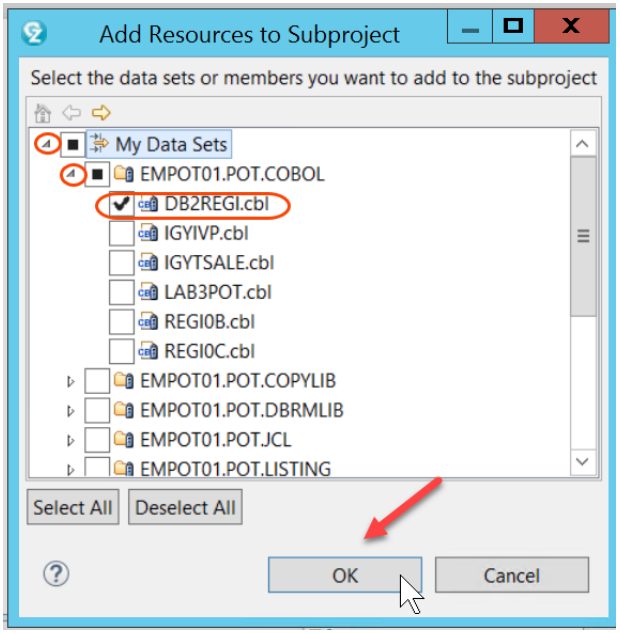
This dialog will be displayed:



___ 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 two members..

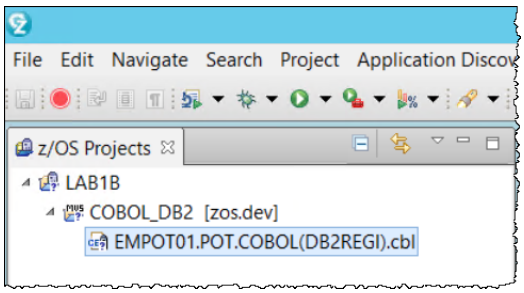
2.2.1 ⏭ On the dialog below, expand **My Data Sets, EMPOT01.POT.COBOL ,** select **DB2REGI.cbl,** and click **OK**



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..

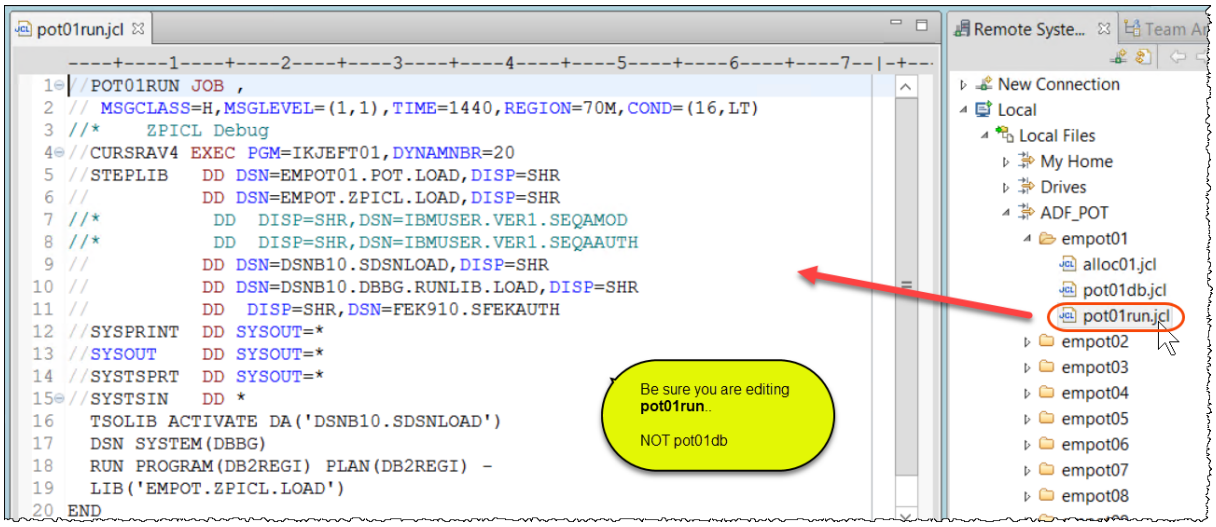


___ 2.3 Submit a JCL to execute the COBOL program

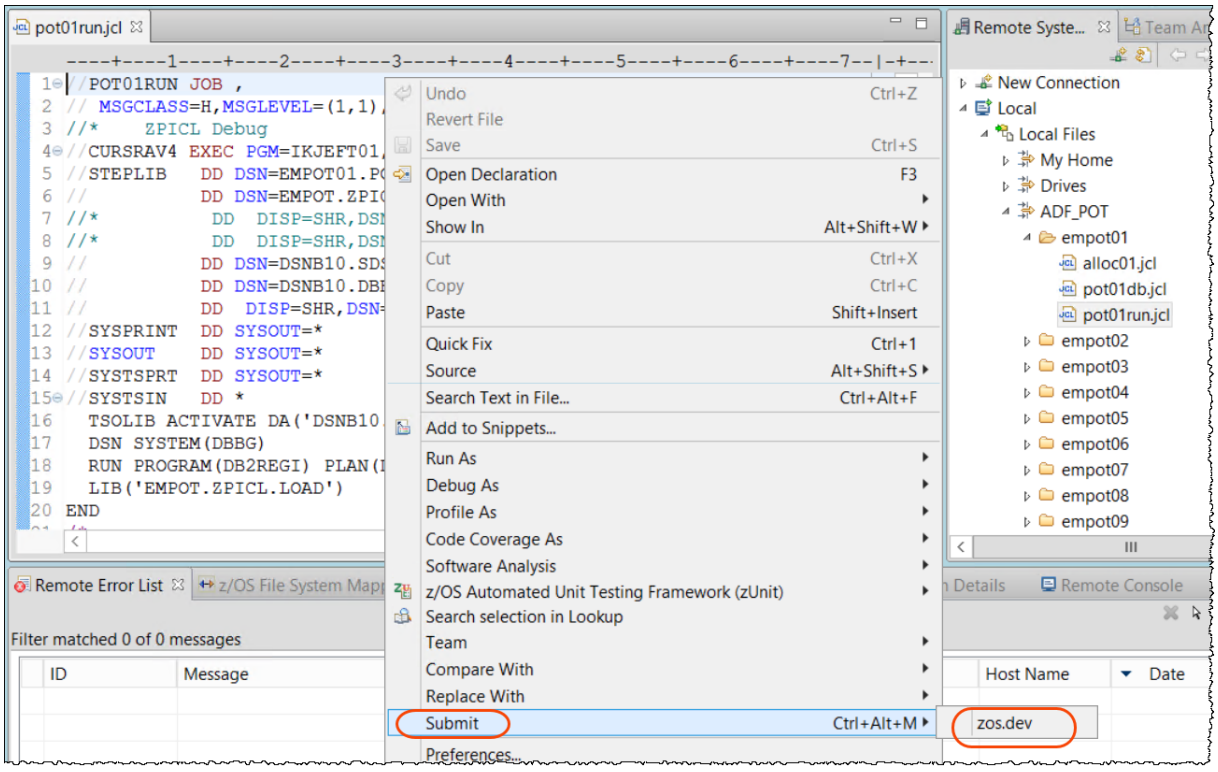
The JCL to execute the program is available on the local windows folder: "`C:\ADF_POT\empot01`". You will use this JCL to execute a batch COBOL/DB2 on z/OS.

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.

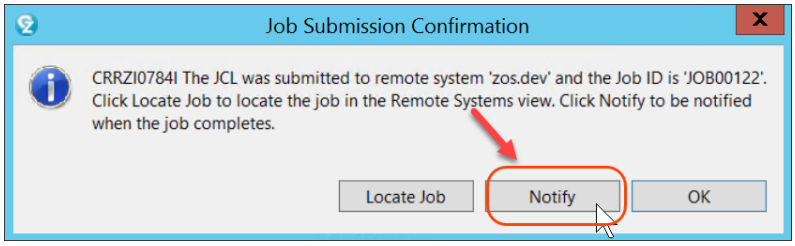
.



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

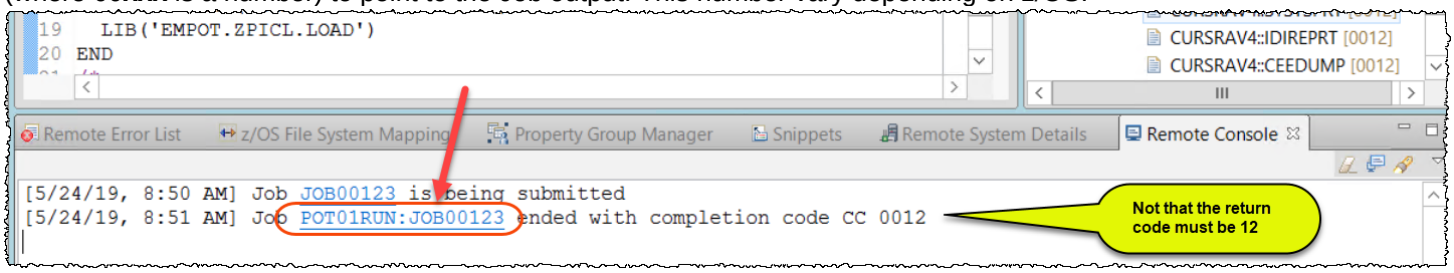


2.3.3 ⏭ When the dialog pop up appears, click **Notify.** This allows you to be notified when the execution is ended.

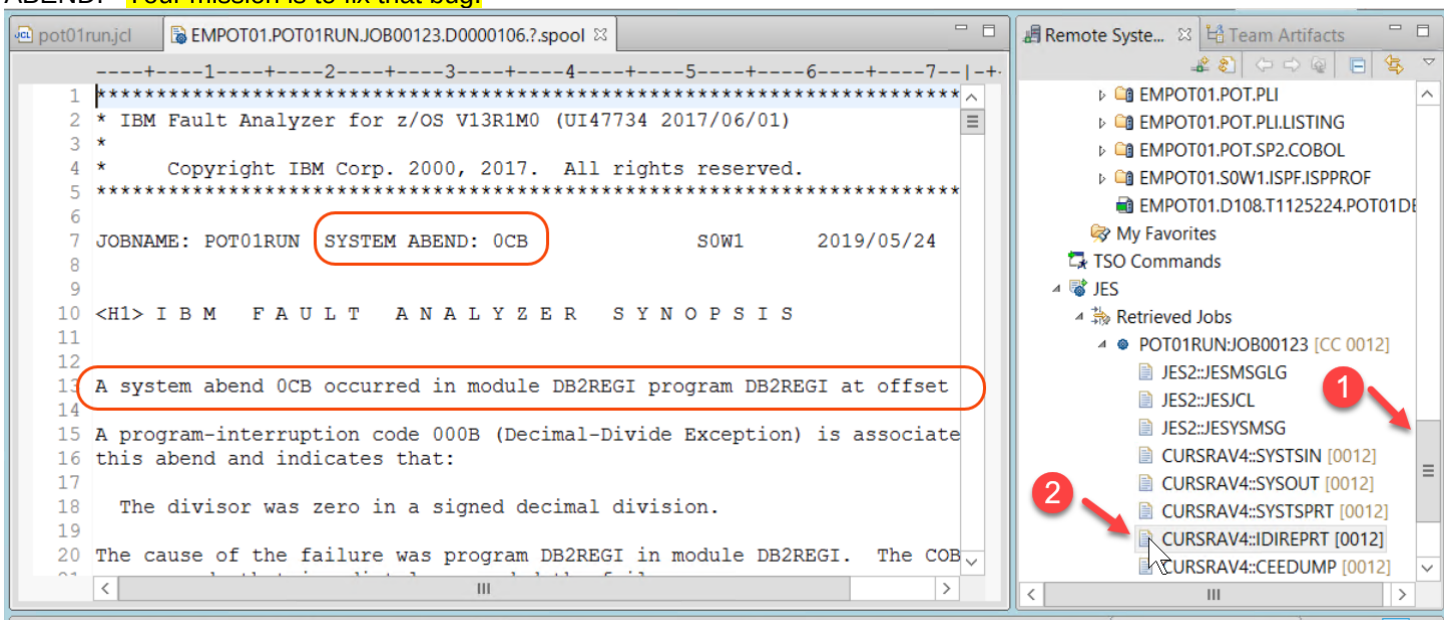


2.3.4 Under *Remote Console*, you will be notified when execution is completed.

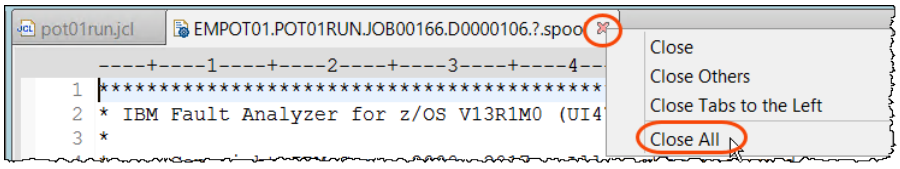
⏭ Once the execution ends, **click** on the link **POT01RUN:JOB00xxx**
(where 00**xxx** is a number) to point to the Job output. This number vary depending on z/OS.



2.3.5 ⏭ Under *Remote Systems* view **scroll down**, and expand JES > Retrieved Jobs > **POT01RUN:JOB00xxx** and **double click CURSRAV4::IDIREPRT [0012]** step and you will see the *Fault Analyzer* report showing an ***OCB*** ABEND. Your mission is to fix that bug.



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



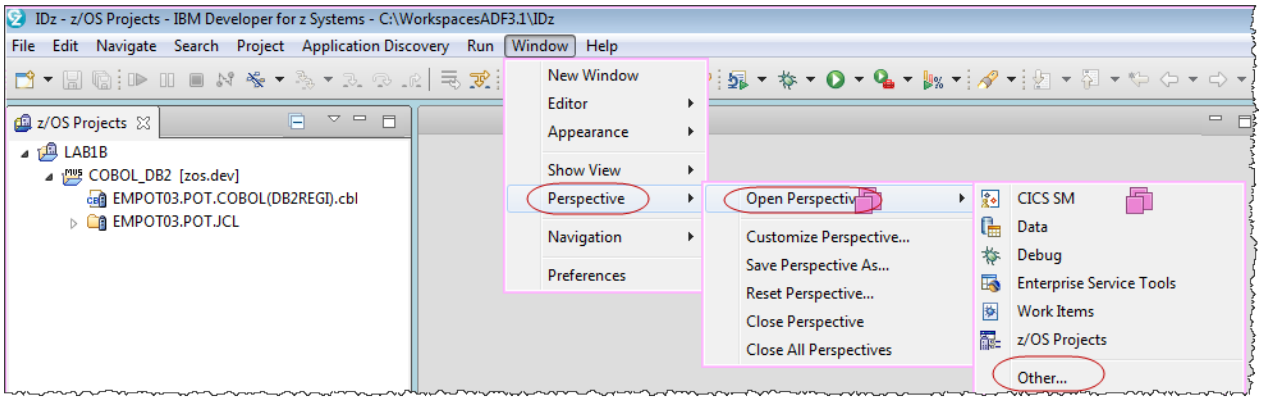
Section 3. Use Fault Analyzer to identify the cause of the ABEND

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

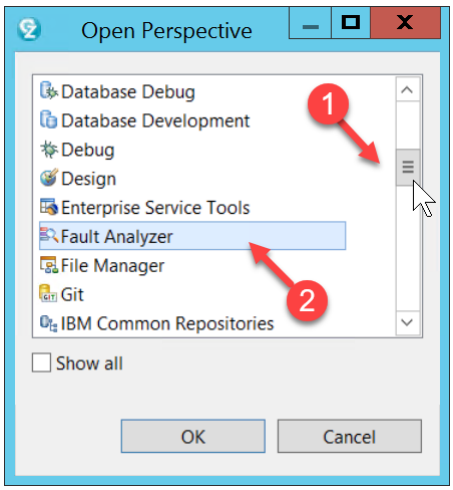
What is IBM Fault Analyzer for z/OS?
IBM Fault Analyzer for z/OS helps developers analyze and fix system and application failures for CICS, WebSphere MQ, IMS and DB2 environments. When an application ends abnormally, Fault Analyzer is automatically engaged, gathering real-time information about the event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

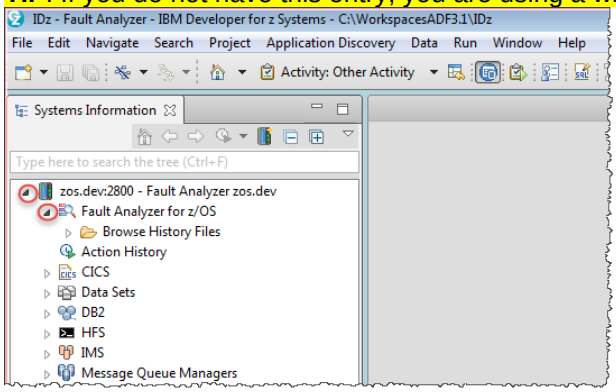


3.1.2 Scroll down, select **Fault Analyzer** and click **OK**



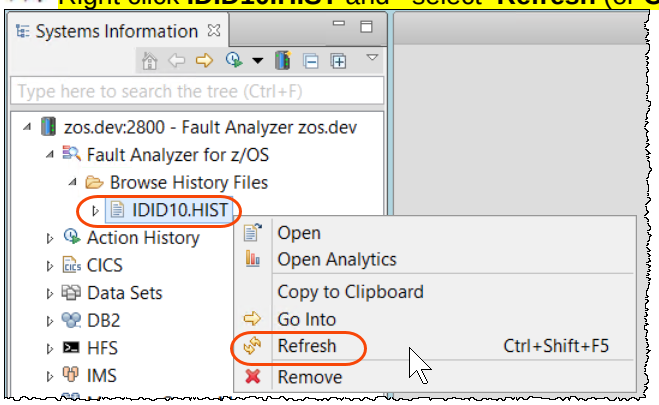
3.1.3 Under *Systems Information* view (on left) expand **Fault Analyzer for z/OS**

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



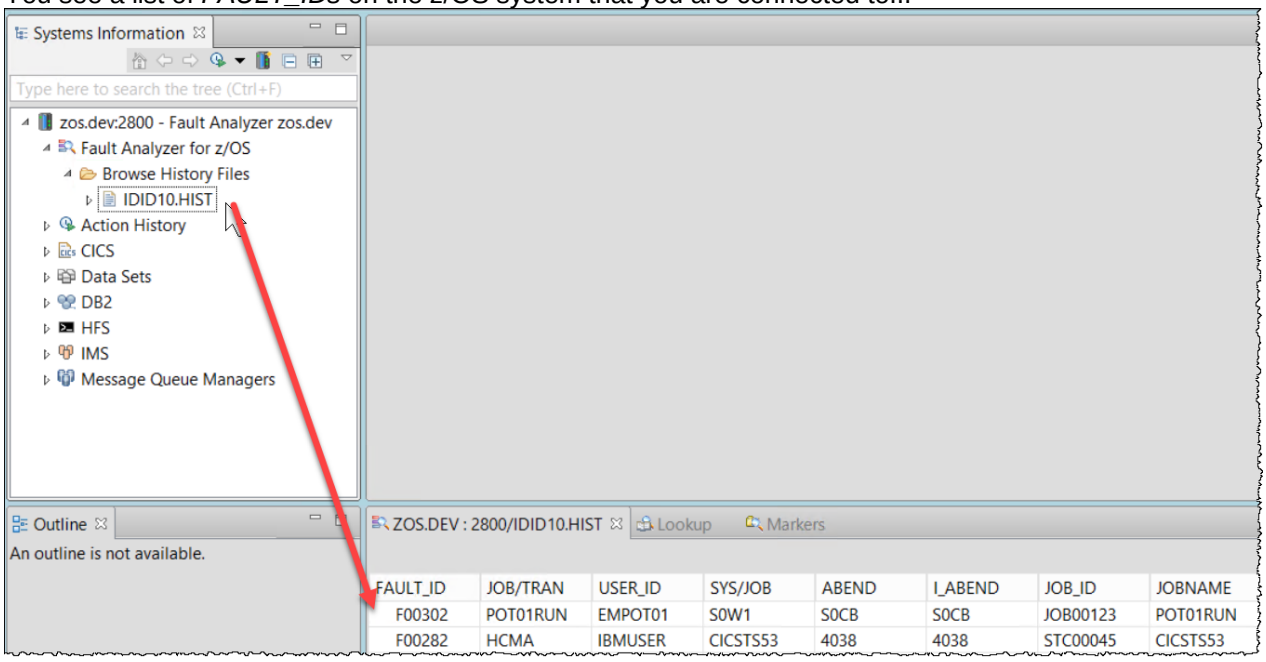
3.1.4 Expand **Browse History File**. You will see the folder ***IDID10.HIST***

Right click **IDID10.HIST** and select **Refresh** (or **Ctrl + Shift + F5**)

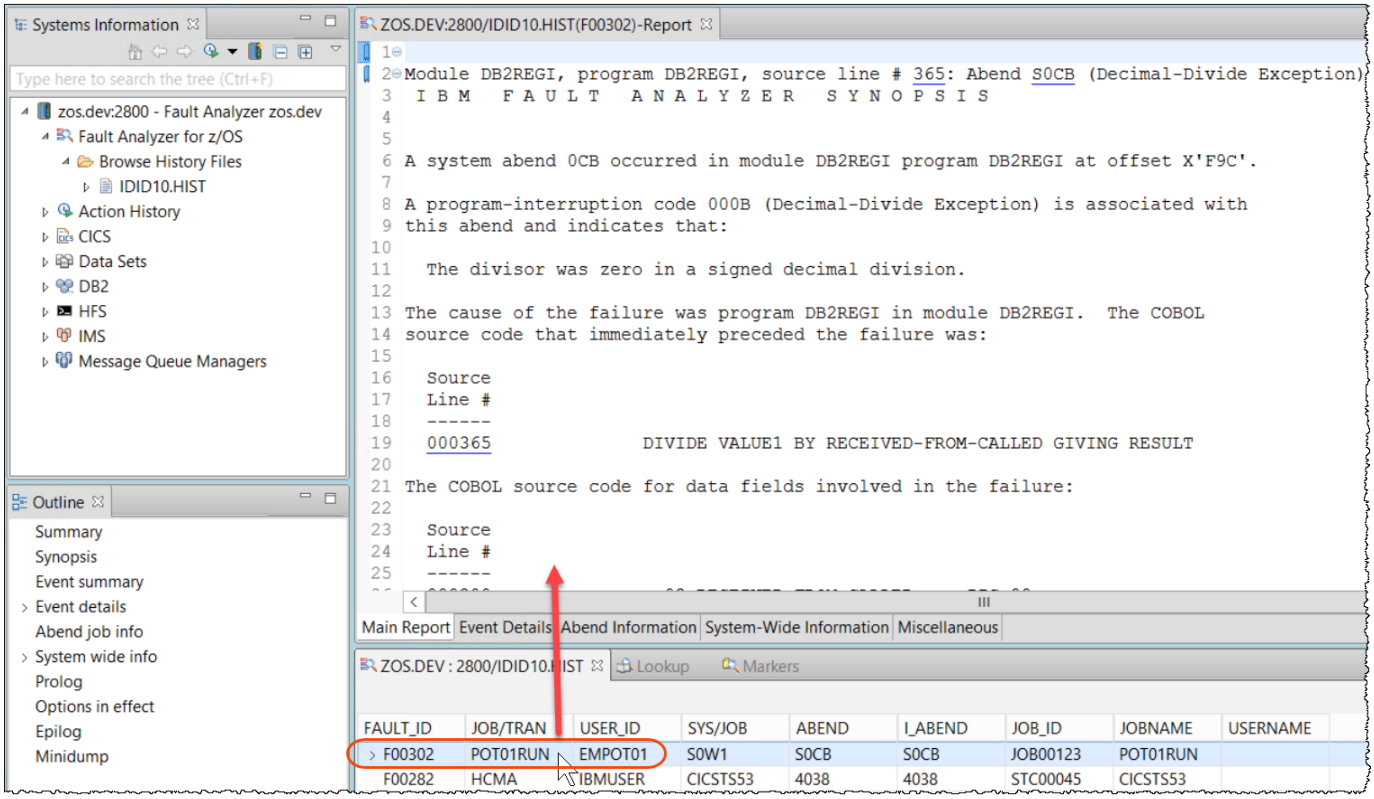


3.1.5 If you get a *Sign on* dialog for *Fault Analyzer* use empot01 credentials and click **OK**

You see a list of *FAULT_IDs* on the z/OS system that you are connected to...

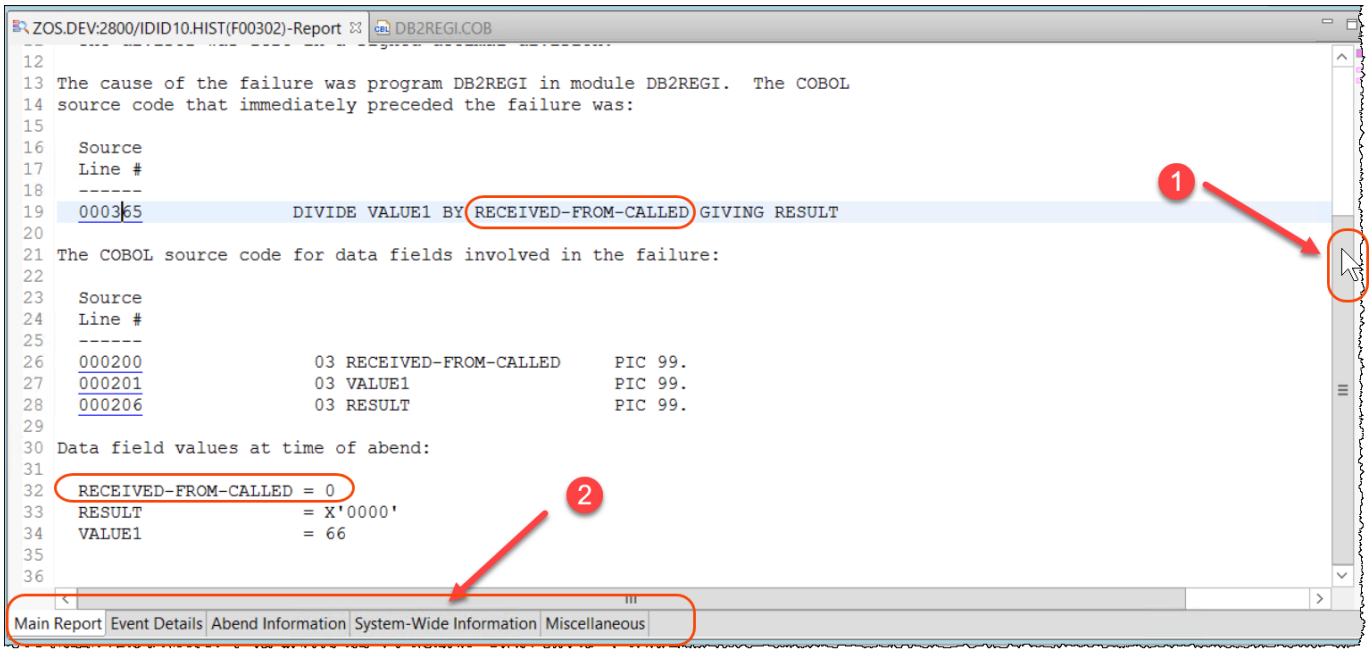


3.1.6 ⏭ On the view ***ZOS.DEV:2800/IDID10.HIST*** locate the latest job name **POT01RUN** and **double click** on it. You will see the report being downloaded from the z/OS to your Windows client. The report below will be displayed

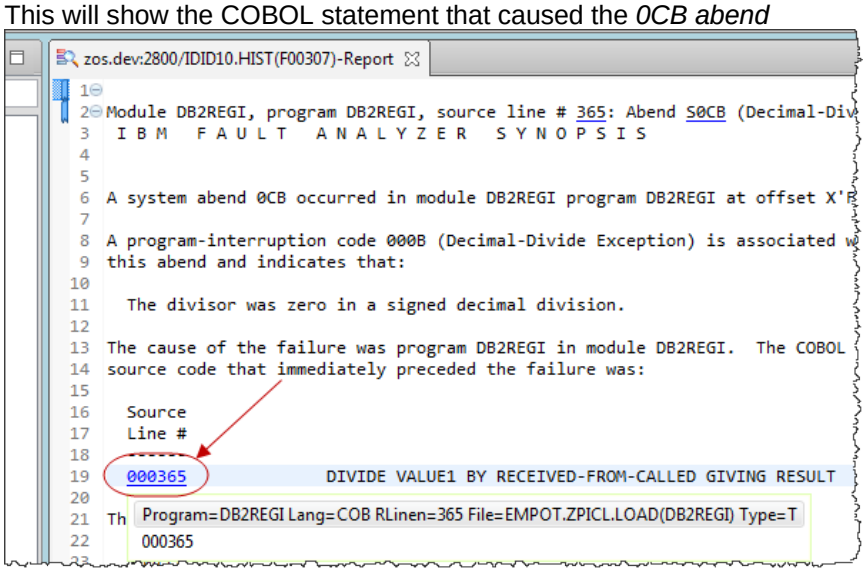


3.1.7 ⏭ Scroll the report down and you will see that the field **RECEIVED-FROM-CALLED** used on the division has “0”. So the abend is divide by zero.

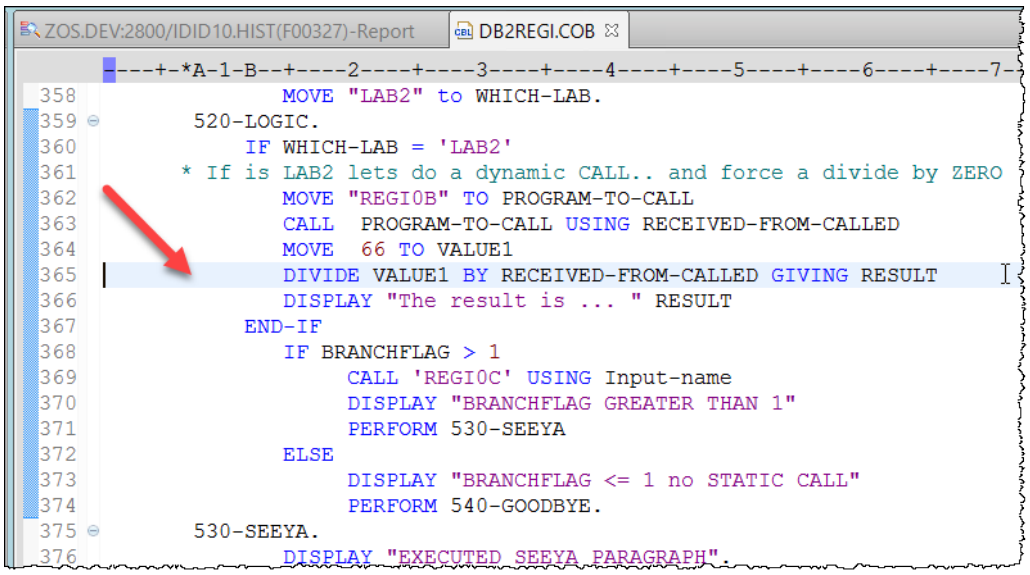
⏭ Also notice that there are other tabs on this panel (like Event Details, Abend Information, etc.). You may try those tabs to get more information about the abend.



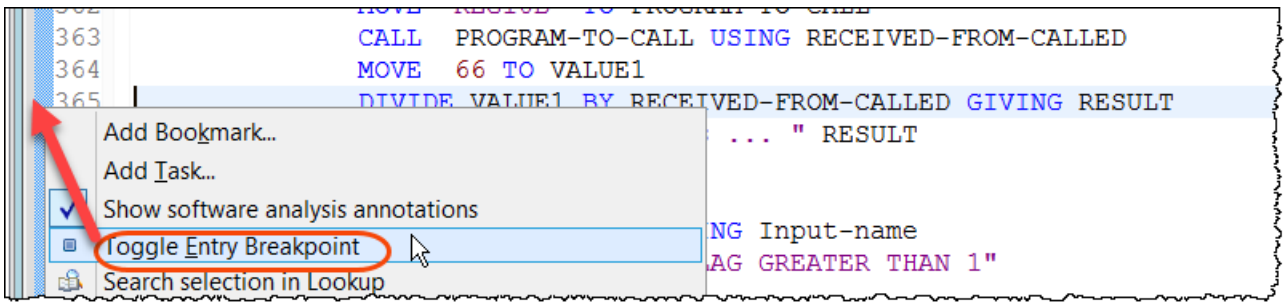
3.1.8 Using the *Main Report* view Click on the link **000365**
This will show the COBOL statement that caused the *0CB abend*



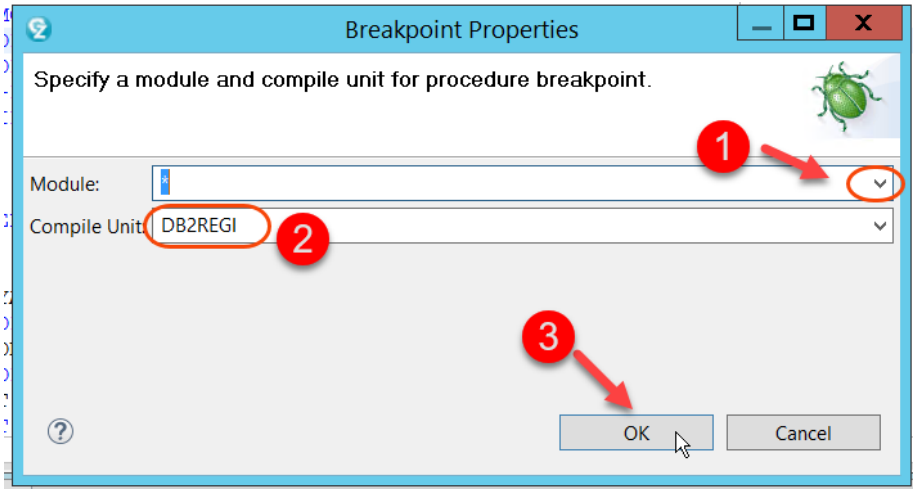
3.1.9 The program editor shows the line with the statement that is causing the abend.
Notice that this is NOT the COBOL source code. This what is on the “minidump” downloaded by Fault Analyzer. There is no sense to make changes here. But you will see what caused the abend.



3.1.10 On line **365**, right-click in the ruler area (left most) and select **Toggle Entry Breakpoint**.



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



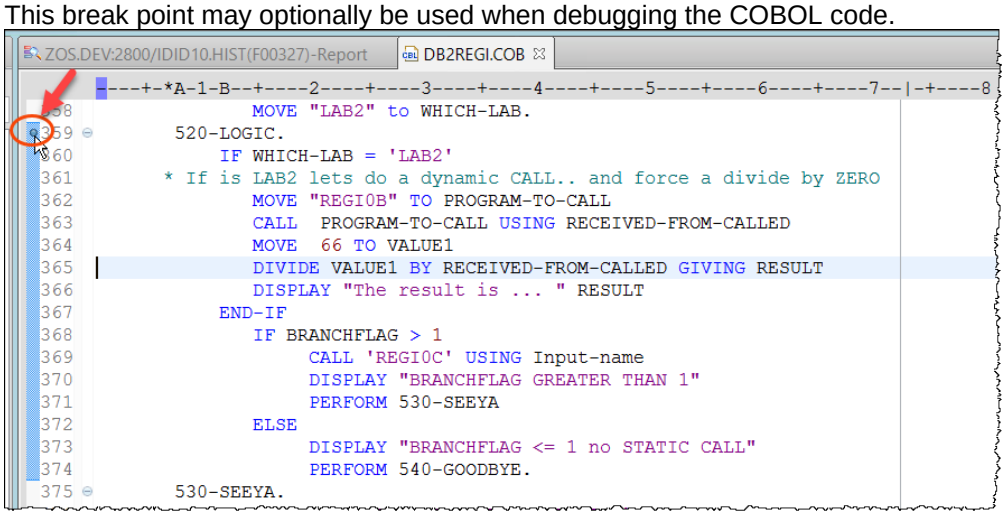
What is *Toggle Entry Breakpoint*?

Toggle Entry 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 allows a more natural edit, compile,
debug workflow.

3.1.12 The Toggle Entry Breakpoint is set.
This break point may optionally be used when debugging the COBOL code.



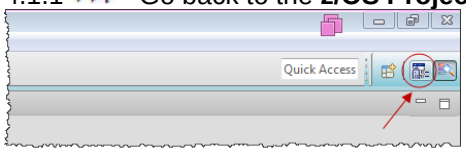
3.1.13 Close all opened editors using **CTRL+ Shift + F4**.
Or just click on the . of each opened editor.

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

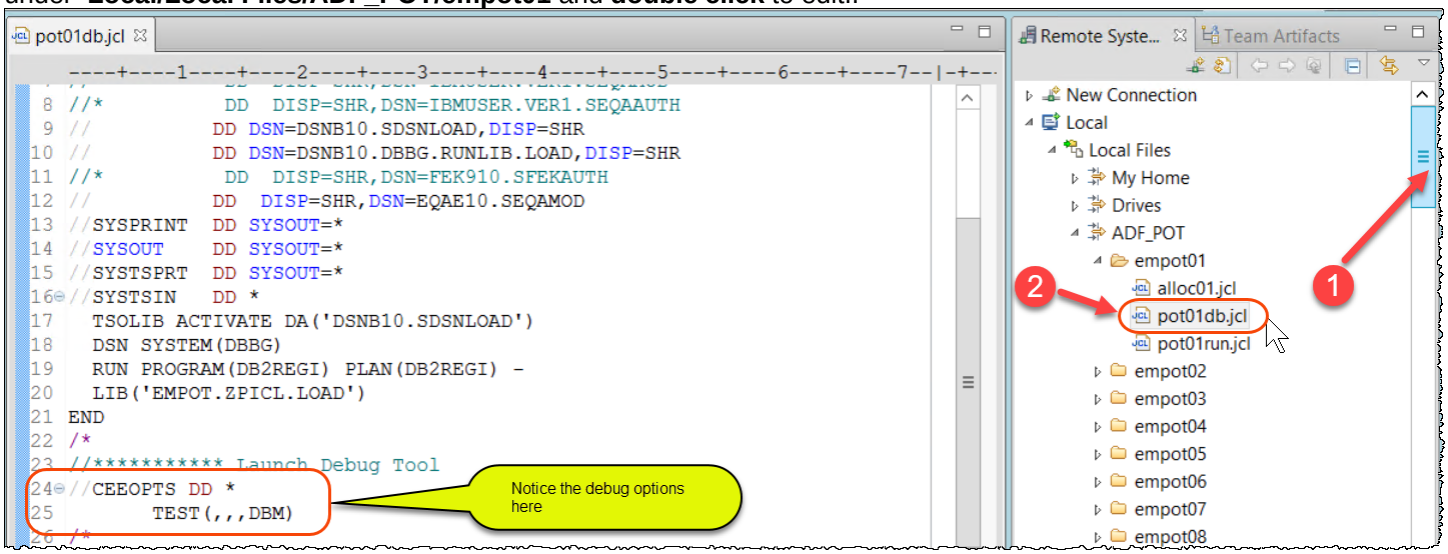
You will use the Debug to verify the ABEND and make a runtime fix.

___ 4.1 Submit the JCL to invoke the Debug

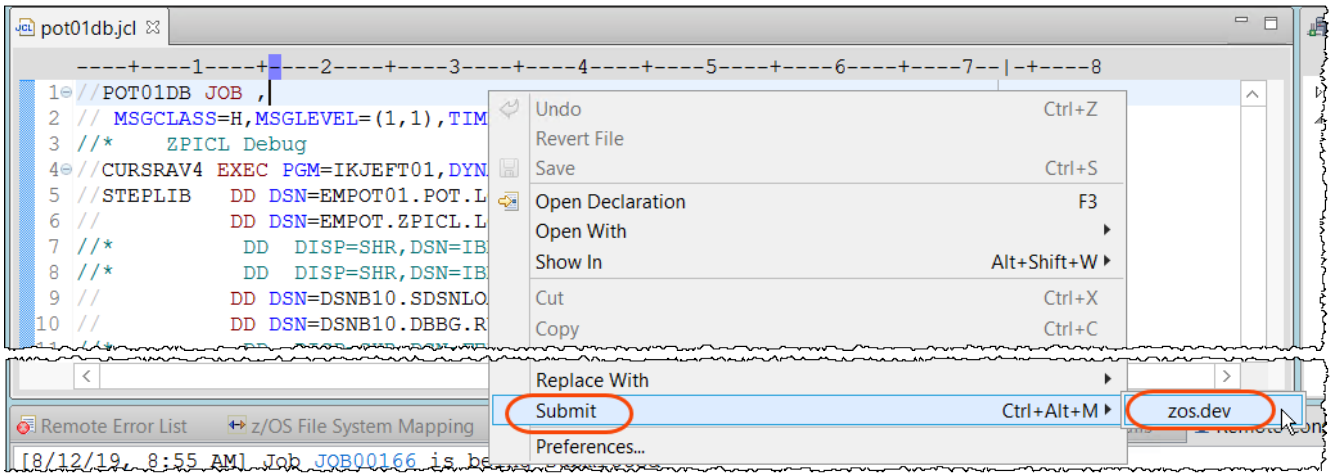
4.1.1 Go back to the **z/OS Projects Perspective** by clicking on the icon below on the right corner.



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..



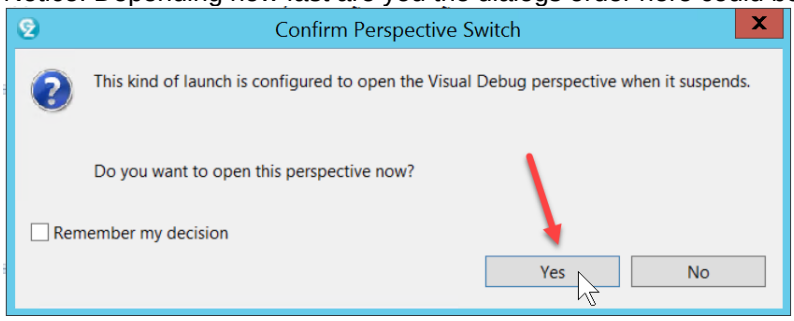
4.1.3 **Right click** on the JCL being edited and select **Submit** > **zos.dev**



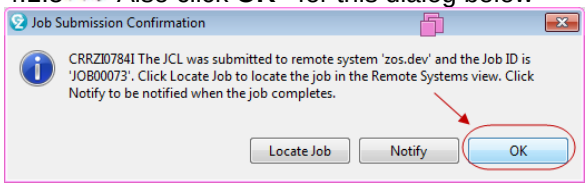
4.1.4 Once the job starts the z/OS debug will "talk" with you. Notice that the communication will be via the IDz connection (RSE) , you don't need to specify IP addresses. This may work even when firewalls are in place.

⏭ Click **Yes** to open the *Visual Debug* Perspective

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

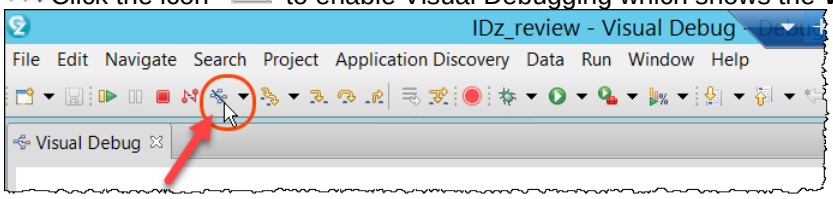


4.1.5 ⏭ Also click **OK** for this dialog below



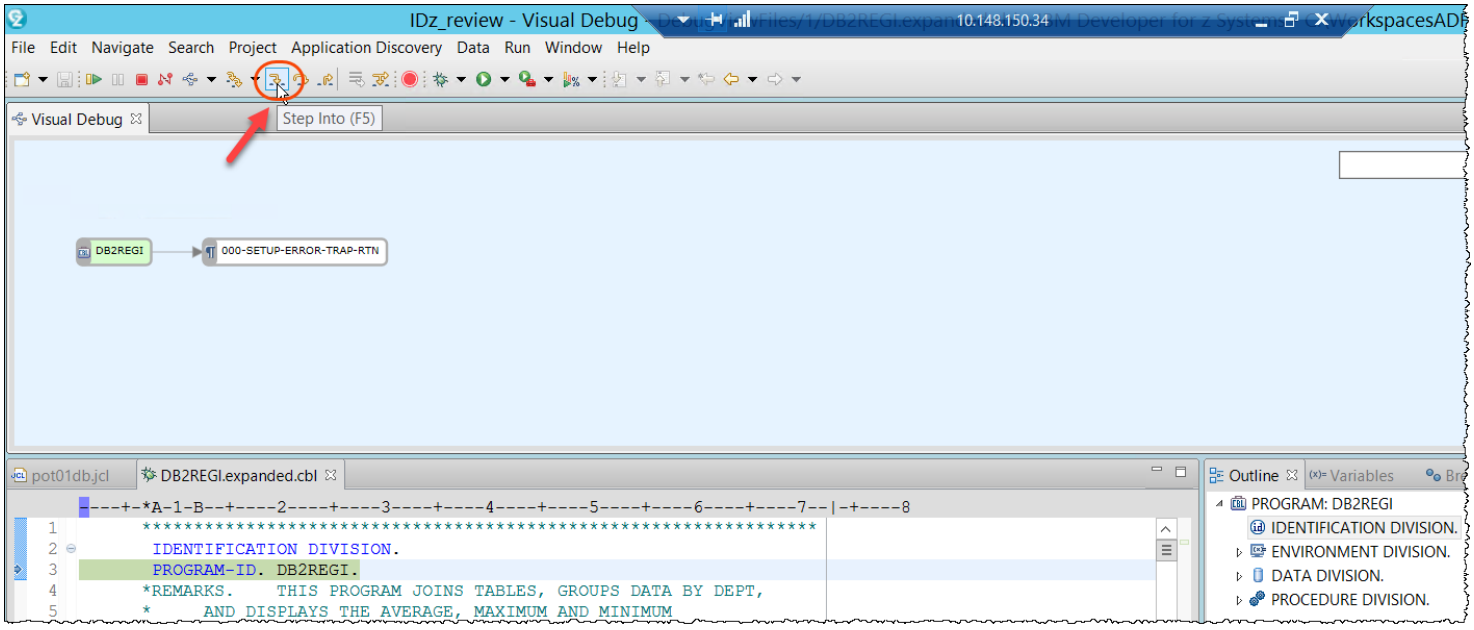
4.1.6 Using the *Visual Debug* perspective:

⏭ Click the icon  to enable Visual Debugging which shows the ***Visual Debug*** view.

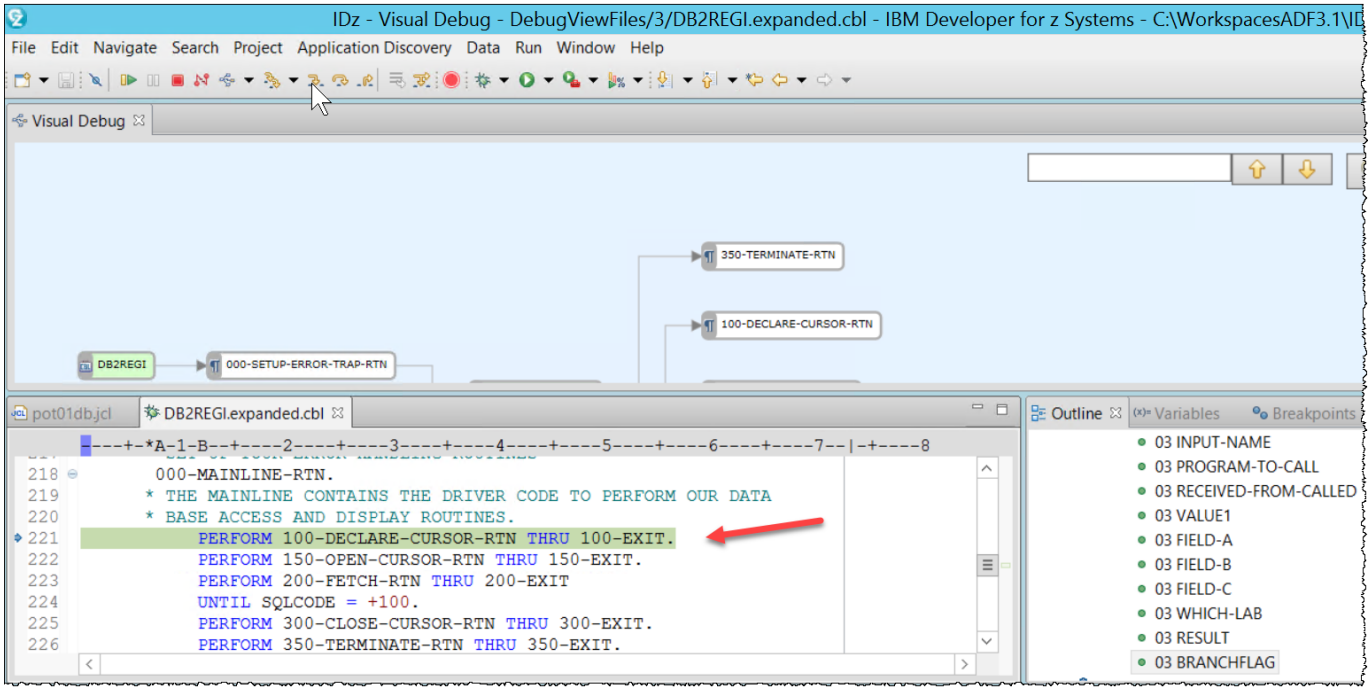


4.1.7 The *Visual Debug* view show the paragraphs being executed (in the top)

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

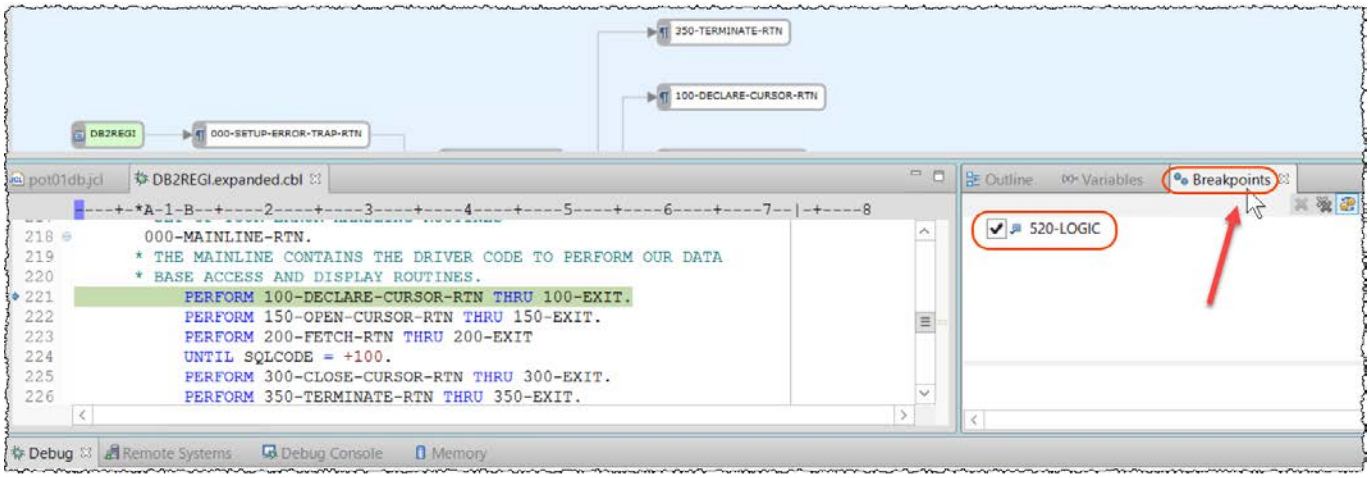


4.1.8 The execution will stop at the statement that will execute
`PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT`



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

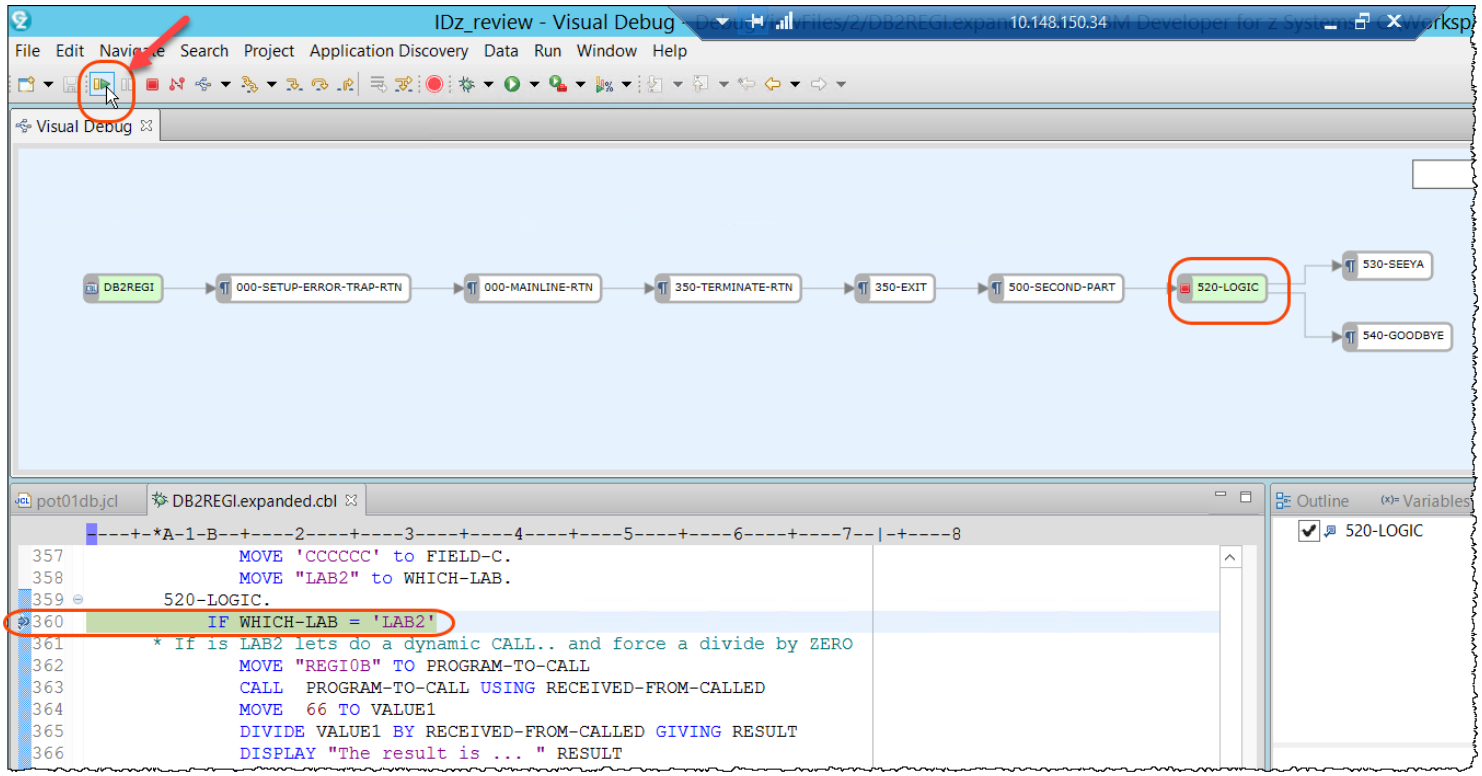
⏭ Click on **Breakpoints** tab to see this breakpoint created before on step 3.1.10.



What is the Visual Debugging?

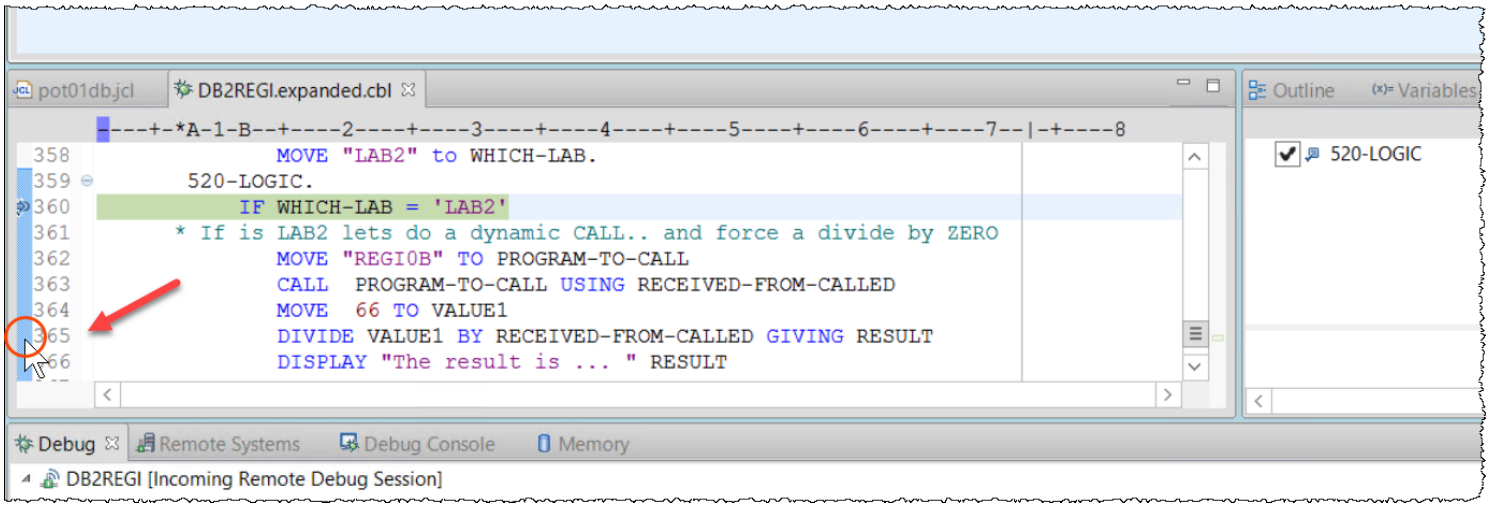
Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.
With this diagram, you can visualize the stack trace, set breakpoints, and run to a selected call path. In COBOL, the stack trace represents the paragraph call chain.
In PL/I, the stack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

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

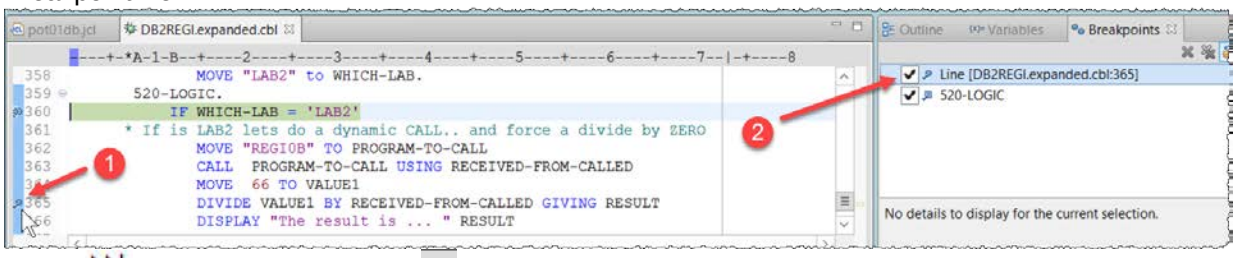


4.1.11. As you verified using Fault Analyzer, the abend occurred on line 365 where you had a divide by zero (see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.

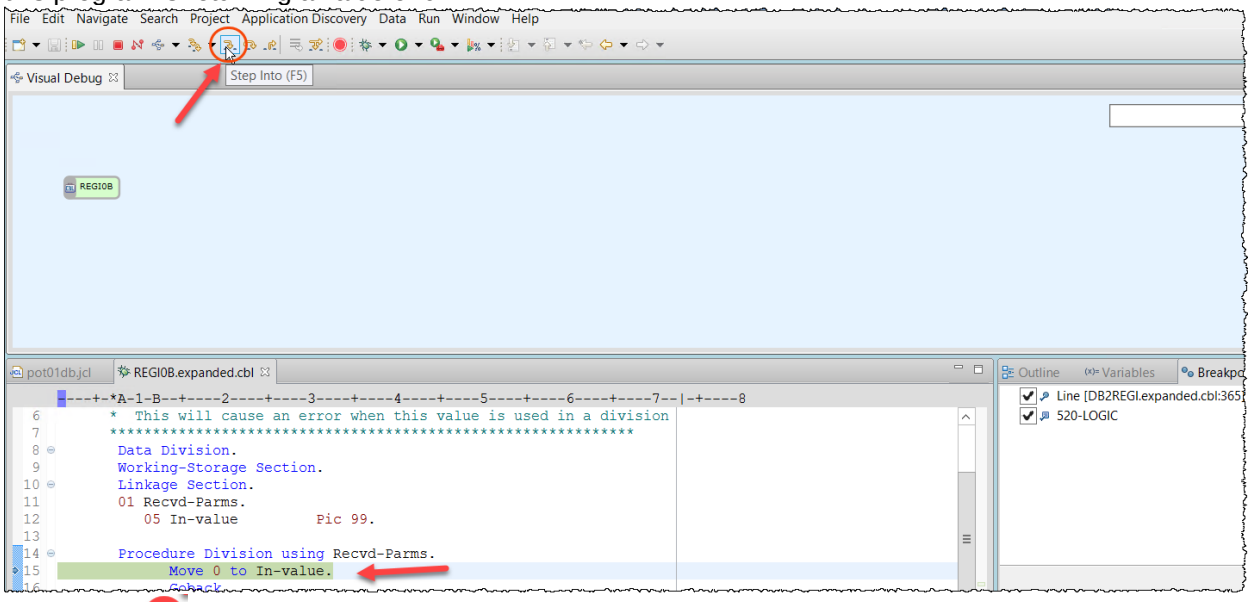
On the COBOL editor move the mouse to the **blue column** on left and **double click** on the line 365 `DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT` to create a breakpoint.



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



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

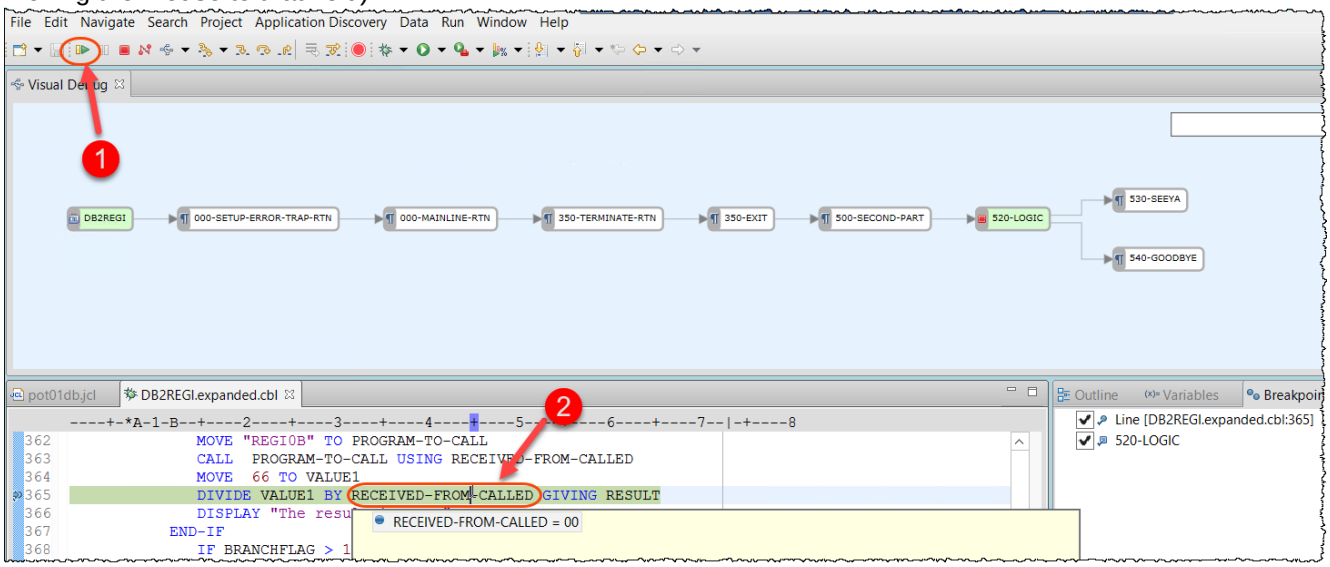


4.1.14 Click on or press **F8.**

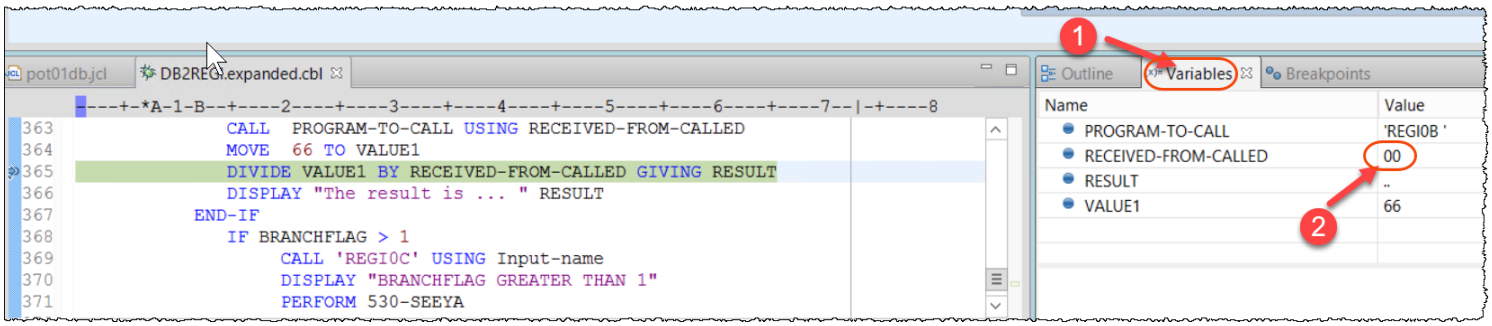
The execution will stop at the breakpoint that you created (line 365). .

Move the cursor to **RECEIVED-FROM-CALLED** variable and click to see the **00** value. .

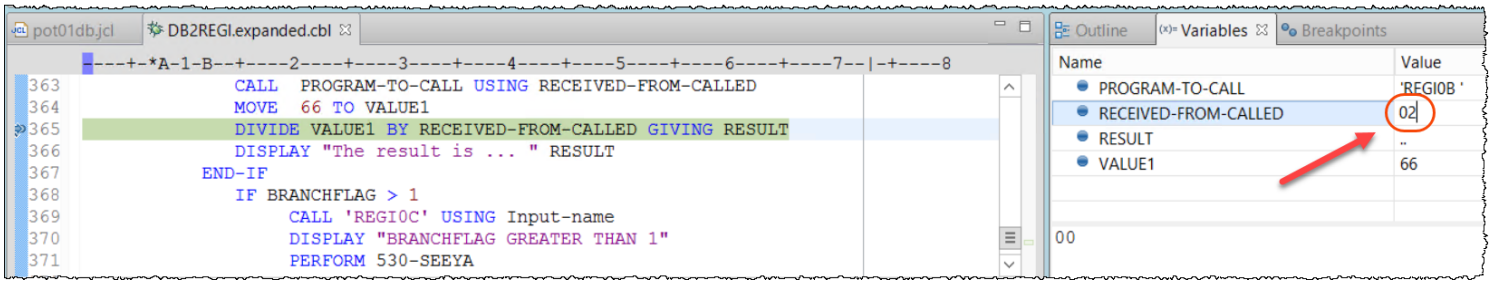
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



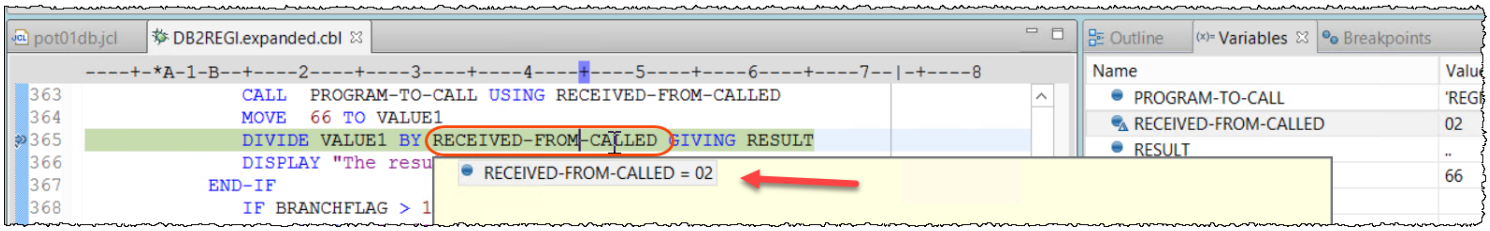
4.1.15 ⏭ (1) Left click the **Variables** tab (right) and (2) verify that **RECEIVED-FROM-CALLED** is **00**.



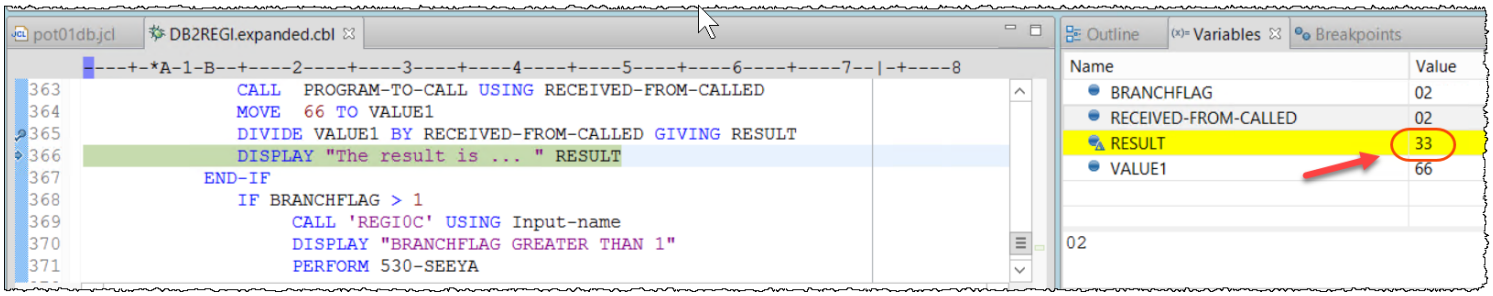
4.1.16 ⏭ This is the abend cause. You must change the value to something different than 0.
⏭ Click on **RECEIVED-FROM-CALLED** value of **00** and modify to **2**, just overtyping and press **enter.**



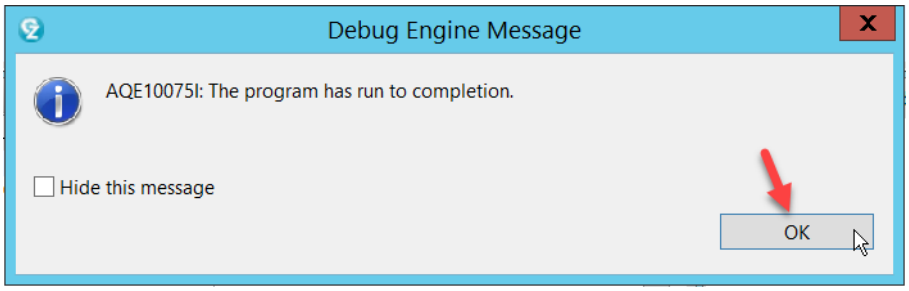
4.1.17 ⏭ Again, move the cursor to **RECEIVED-FROM-CALLED** variable and now see the value **02**.
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



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



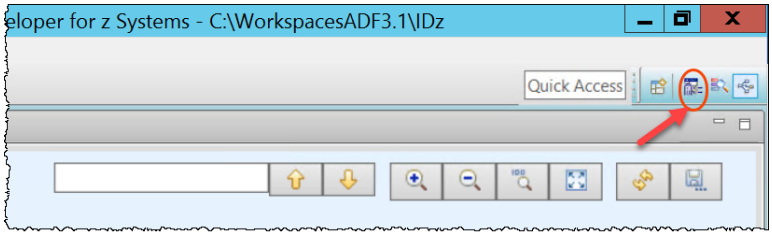
4.1.19 Click on **(or F5)** few times and **Resume** (**F8**) when you are satisfied so the program will execute until the end.



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

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

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



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

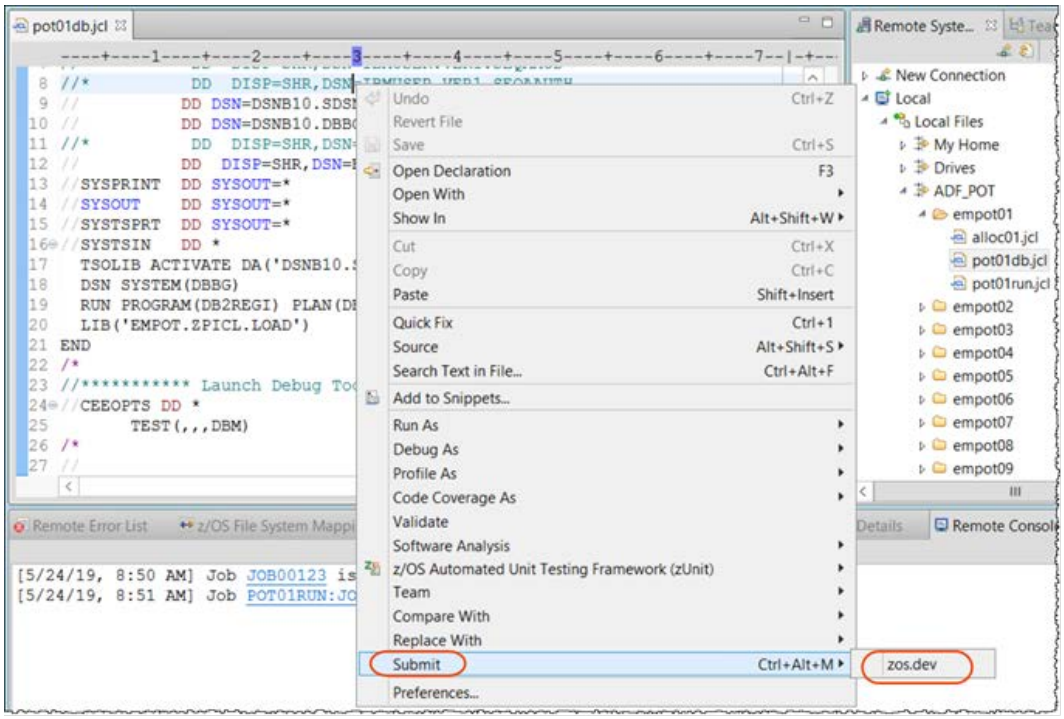
If you are not running late and interested on this subject, execute the steps below, **otherwise jump to 4.3**

Stack pattern breakpoint

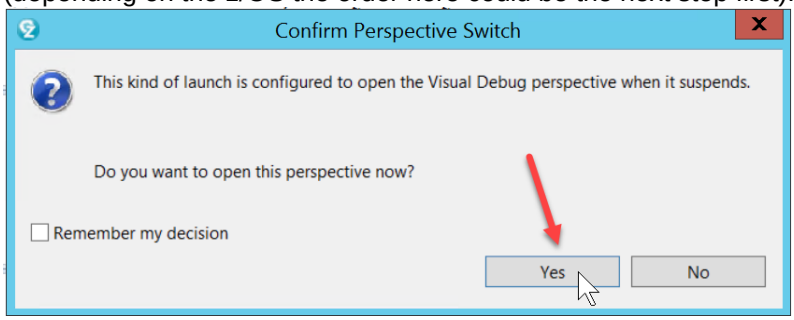
A stack pattern breakpoint is a special type of conditional breakpoint. With this feature, you can specify that you only want to stop at a location when the current stack trace contains a predefined stack pattern.

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

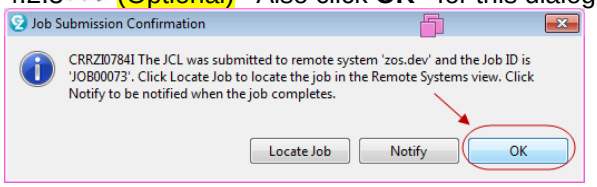
4.2.1 (Optional) **Right click** on the JCL being edited and select **Submit** > **zos.dev**



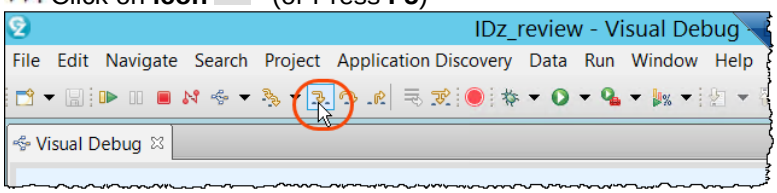
4.2.2 (Optional) Once the job starts the z/OS debug will "talk" with you.
Click **Yes** to open the *Visual Debug* Perspective
(depending on the z/OS the order here could be the next step first).



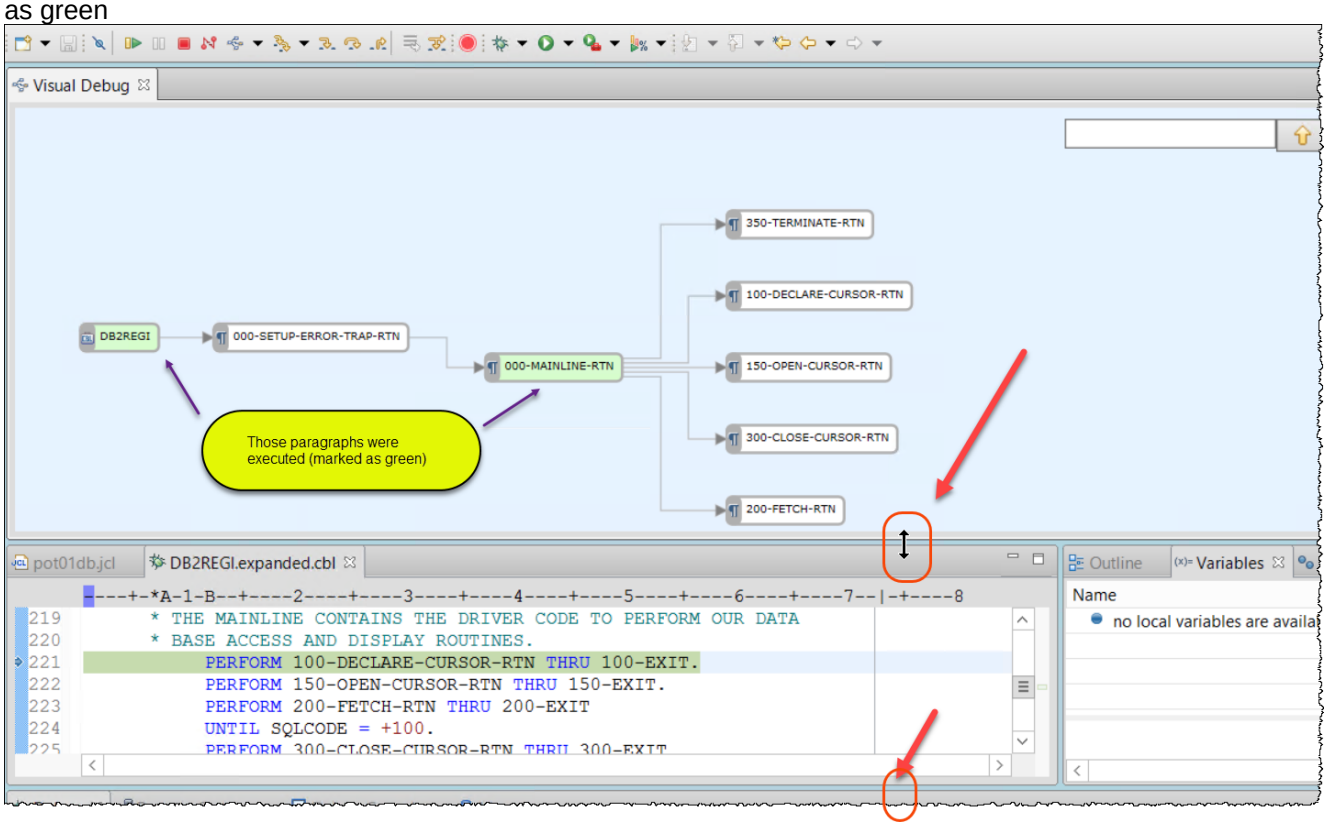
4.2.3 (Optional) Also click **OK** for this dialog below



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

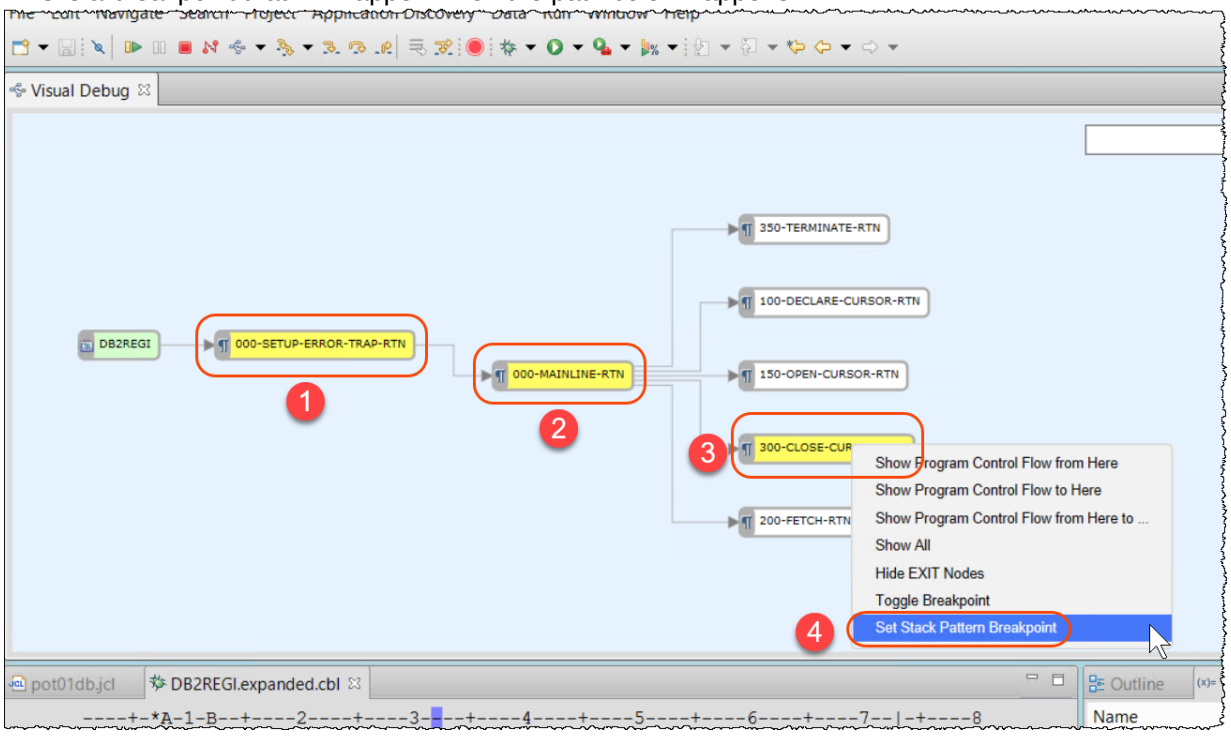


4.2.5 (Optional) Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green

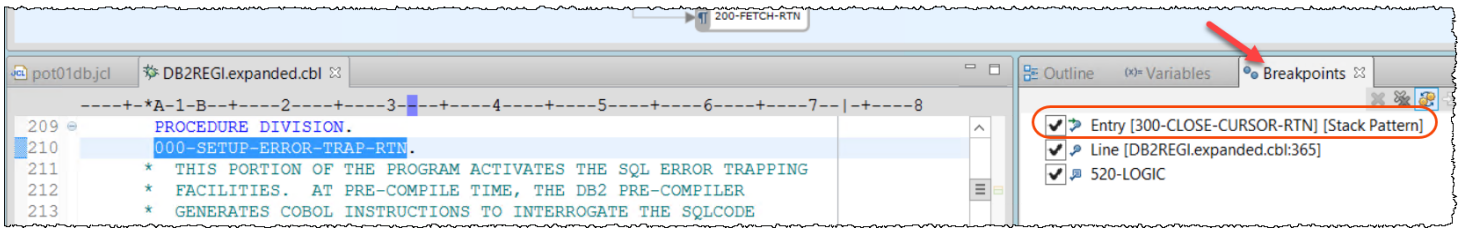


4.2.6 (Optional) Using the *Visual Debug* view, **scroll down**

Press **CTRL** Key and select the path below, right click and choose **Set Stack Pattern Breakpoint**
This is a breakpoint that will happen when the path below happens.

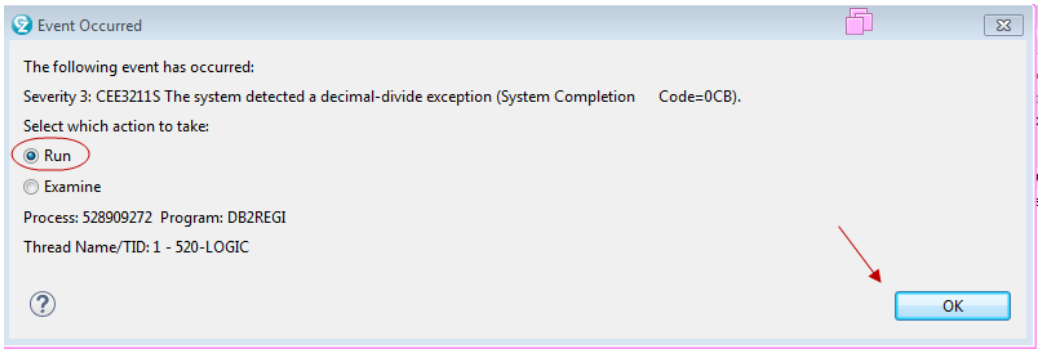


4.2.7 (Optional) Click on **Breakpoints** view and you will see this breakpoint created.



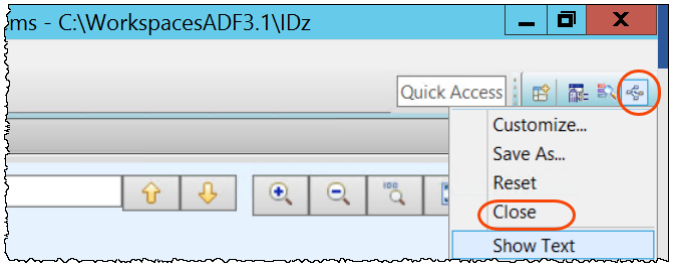
4.2.8 (Optional) Click on  (or press **F8**) few times to resume the execution

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

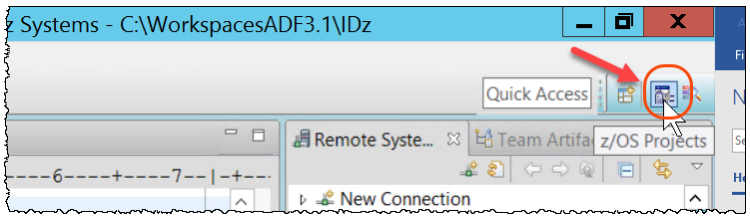


Notice – This breakpoint did not work since you did not have this path.

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



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

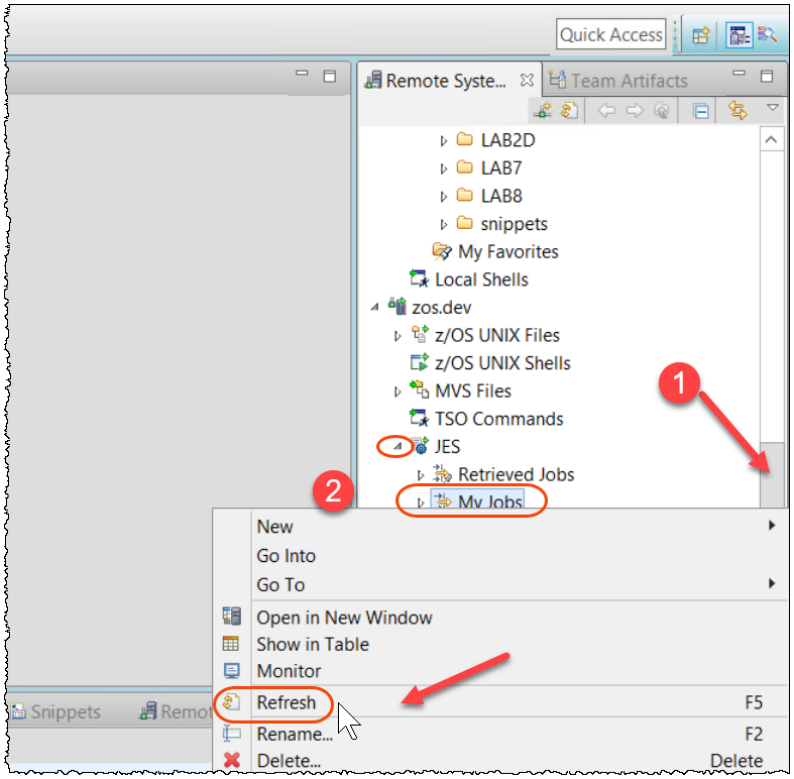


___ 4.3 Accessing the z/OS JES

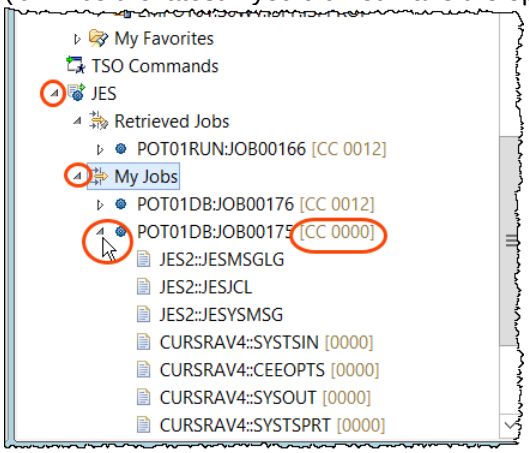
On ISPF many people use *SDSF* for this activity.
You will use the Job Monitor subsystem part of the host components.

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

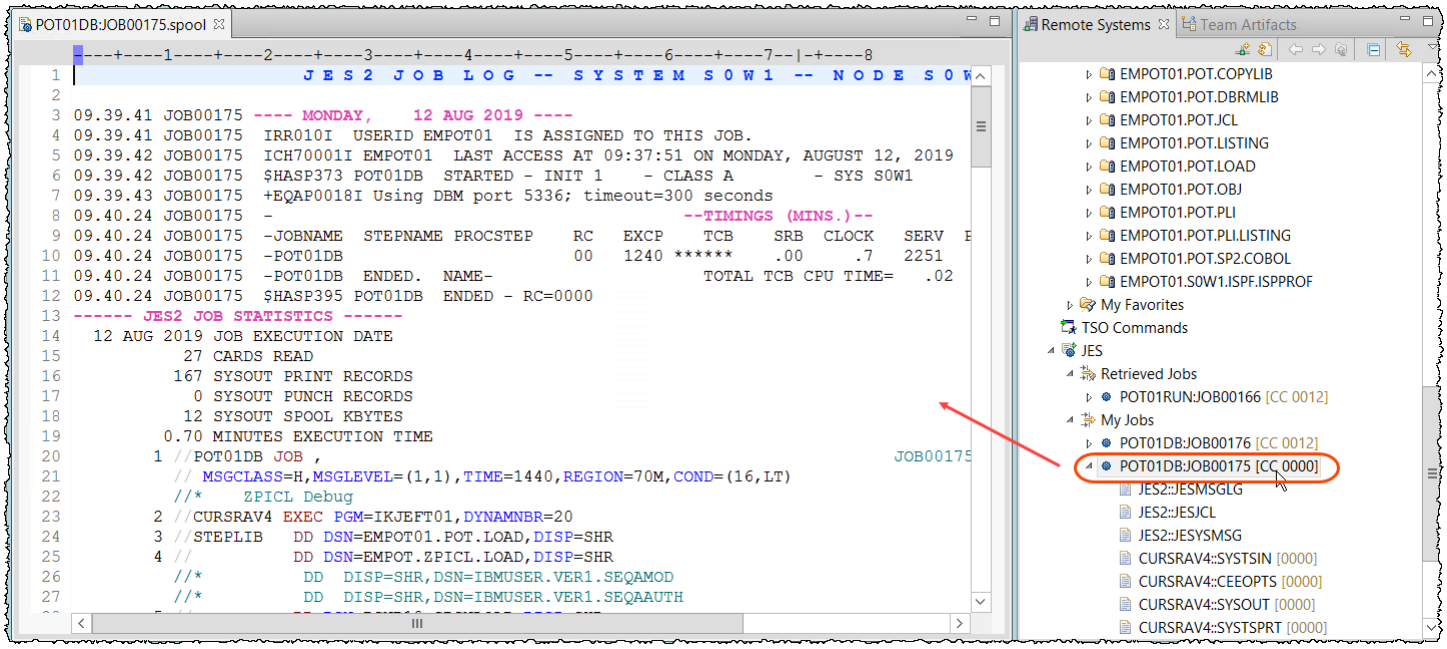
4.3.2 Using the *Remote Systems* view scroll down, expand JES right click on **My Jobs** and select **Refresh**



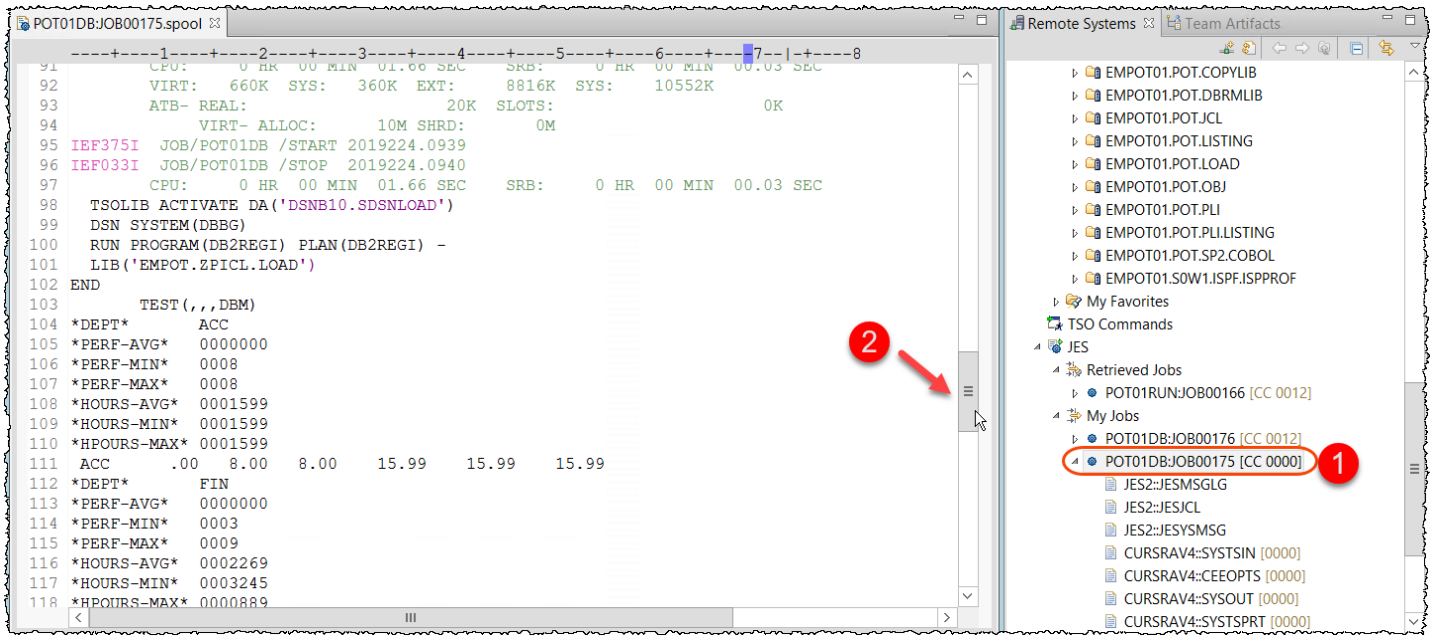
4.3.3 Expand **My Jobs** and the Job **POT01DB:JOB00xxx** that has the CC 0000
(it will be the latest if you did not make the optional step)



4.3.4 On *Remote Systems* view, **double click** on the **POT01DB:JOB00xxx**.



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



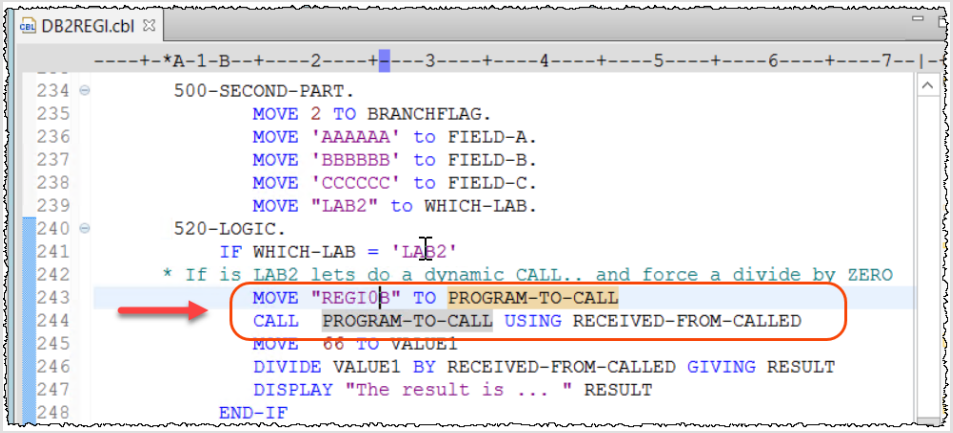
4.3.6 **Close** all opened editors. (**Ctrl + Shift + F4**). Click **No** if asked to save change changes.
Or just click on the . of each opened editor.

Section 5. Modify the COBOL code to fix the bug

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

What z/OS remote assets you will work with?

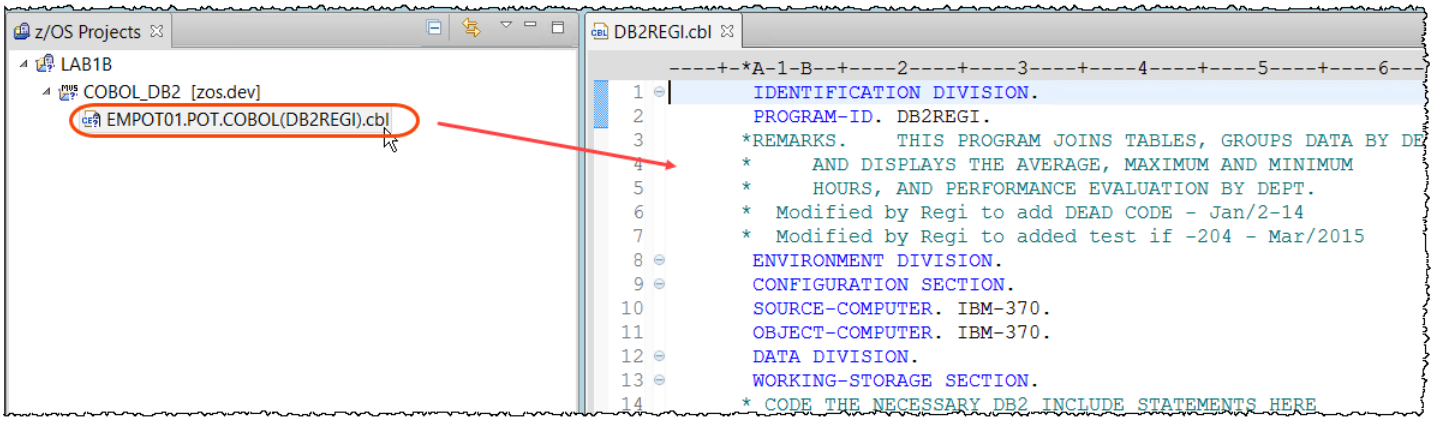
This is a batch program that reads DB2. In addition, this program does a Dynamic call to another COBOL program named **REGI0B**.



___ 5.1 Using the Editor with z/OS COBOL programs

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

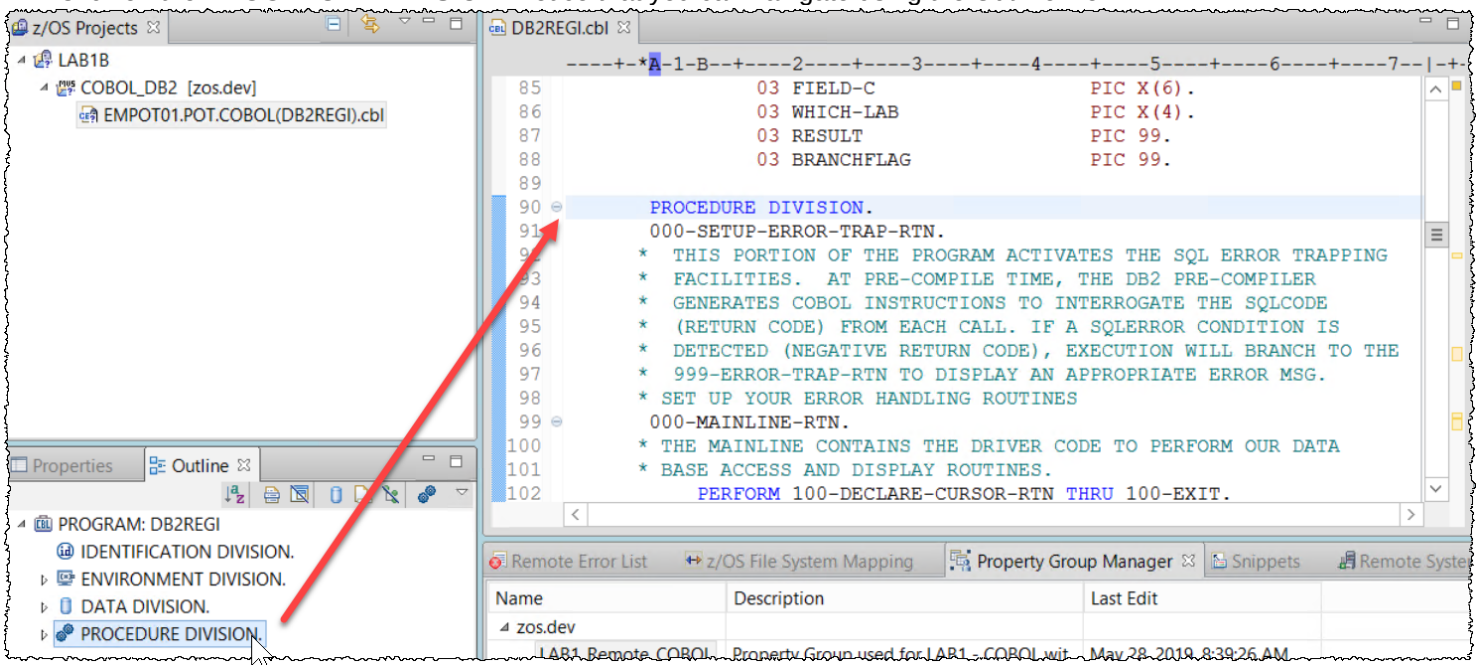
5.1.1 Using the *z/OS Projects* perspective and the *z/OS Projects* view, **double click** on **EMPOT01.POT.COBOL(DB2REGI)** to open the file using the editor.



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

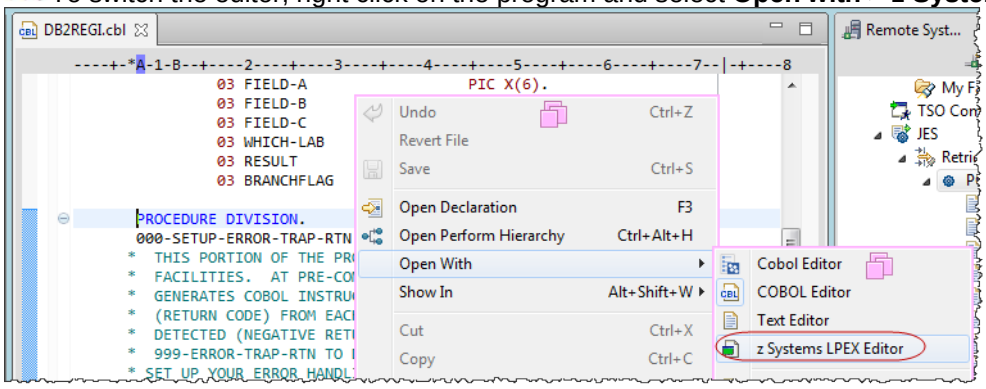
⏭ Using the editor, browse the program and notice that the contents of the *Outline view* are synchronized with the COBOL source code and vice versa.

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

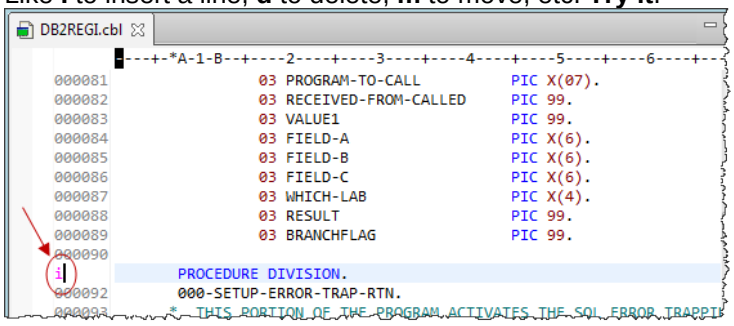


5.1.3 If you prefer the ISPF editor, you might use another IDz editor called “LPEX editor”.

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

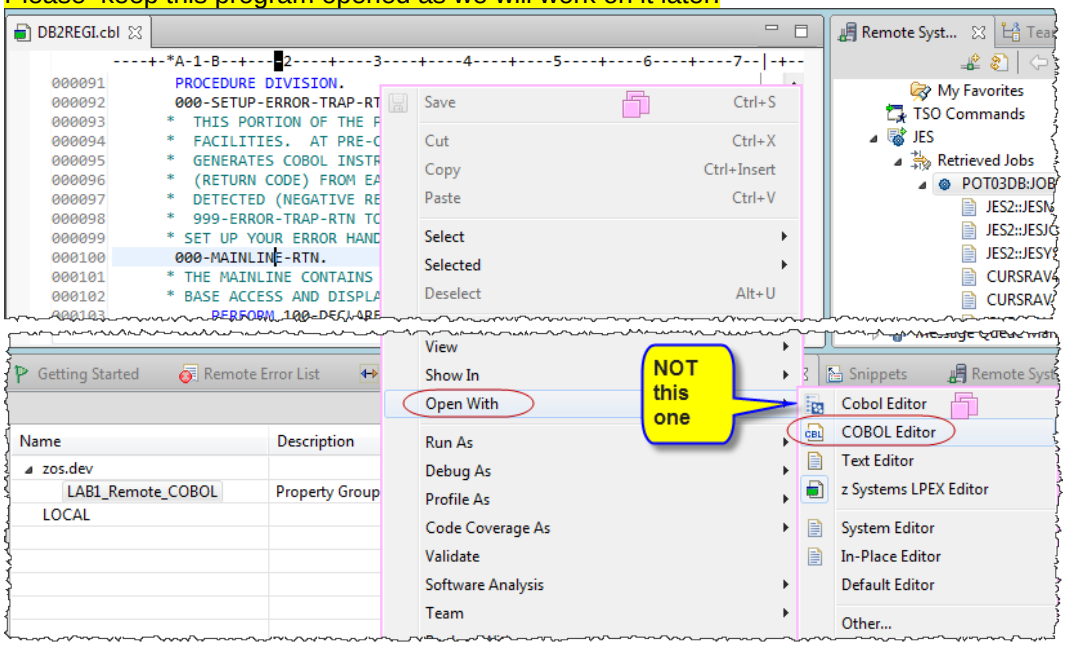


5.1.4 ⏭ Note that the column on left are similar as the prefix area of the ISPF editor..
You may type commands to add lines etc..
Like **I** to insert a line, **d** to delete, **m** to move, etc. **Try it**.



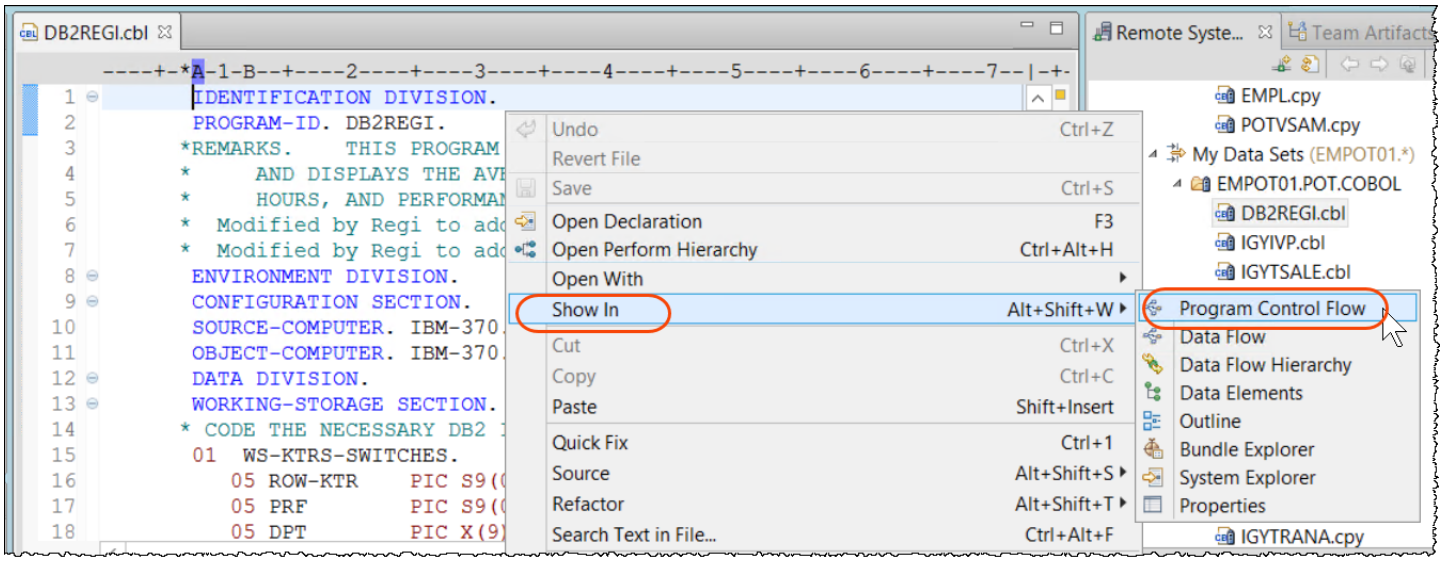
5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)

⏭ Right click on the program and select **Open with > COBOL Editor (the SECOND choice)**
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). Application Discovery uses the first editor.
Please keep this program opened as we will work on it later.



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

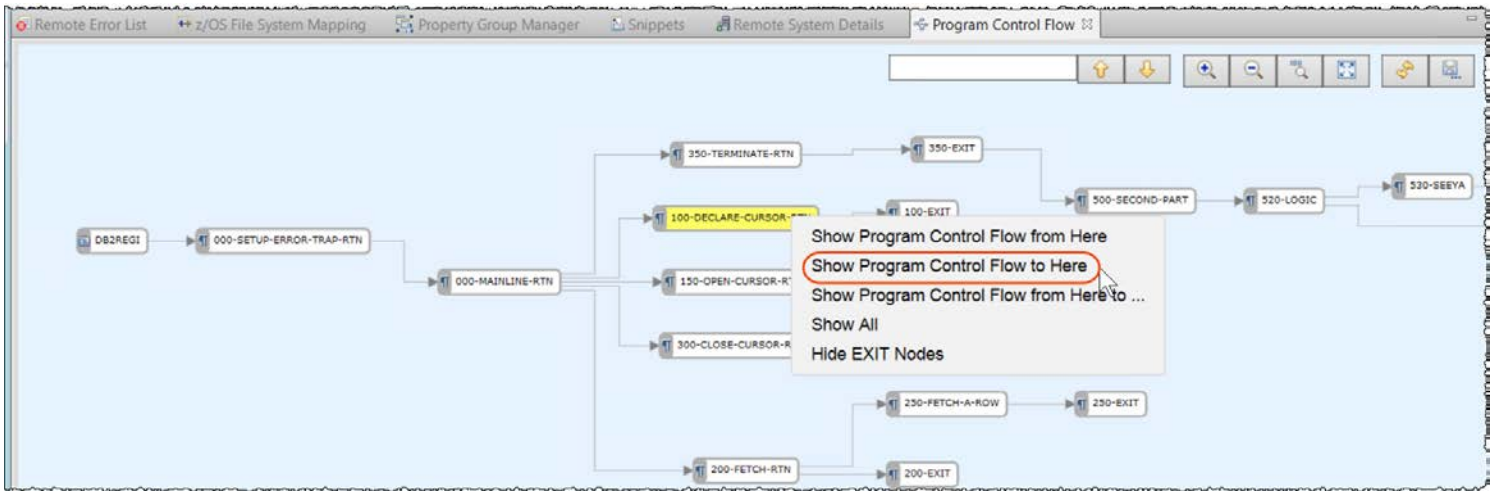
⏭ Right click on the editor and select **Show In → Program Control Flow**



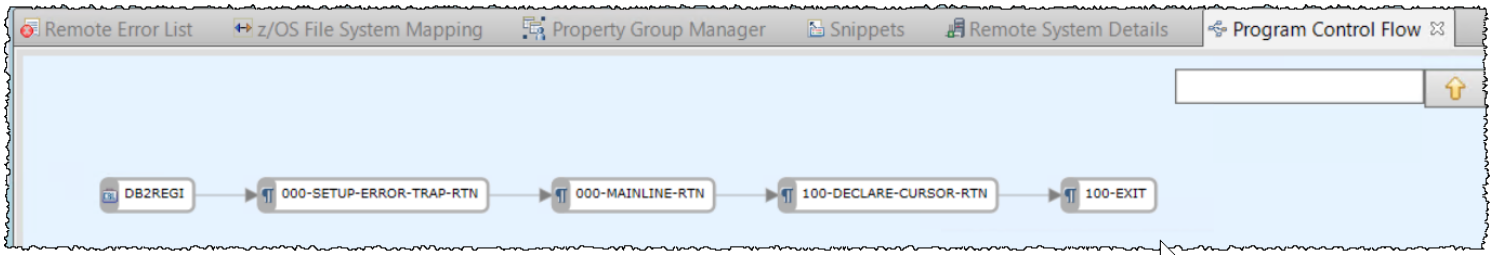
5.1.7 See the results under *Program Control Flow* view.

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

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



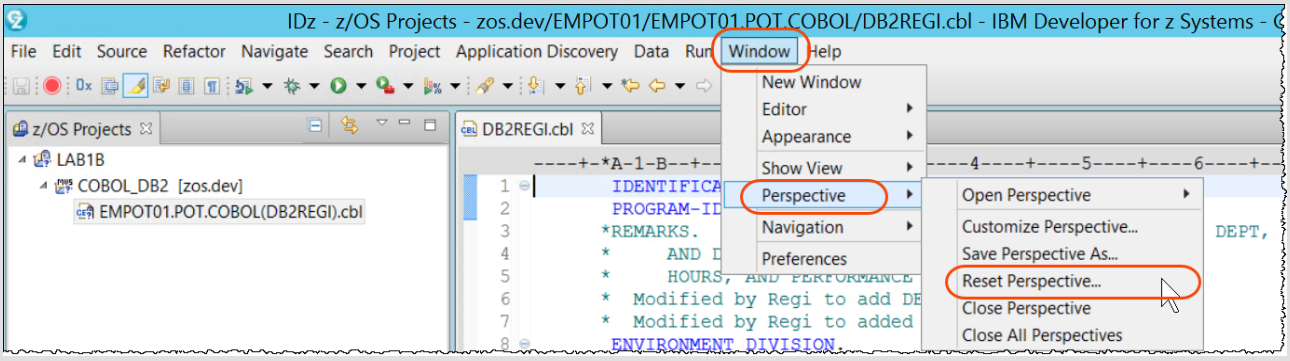
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



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

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

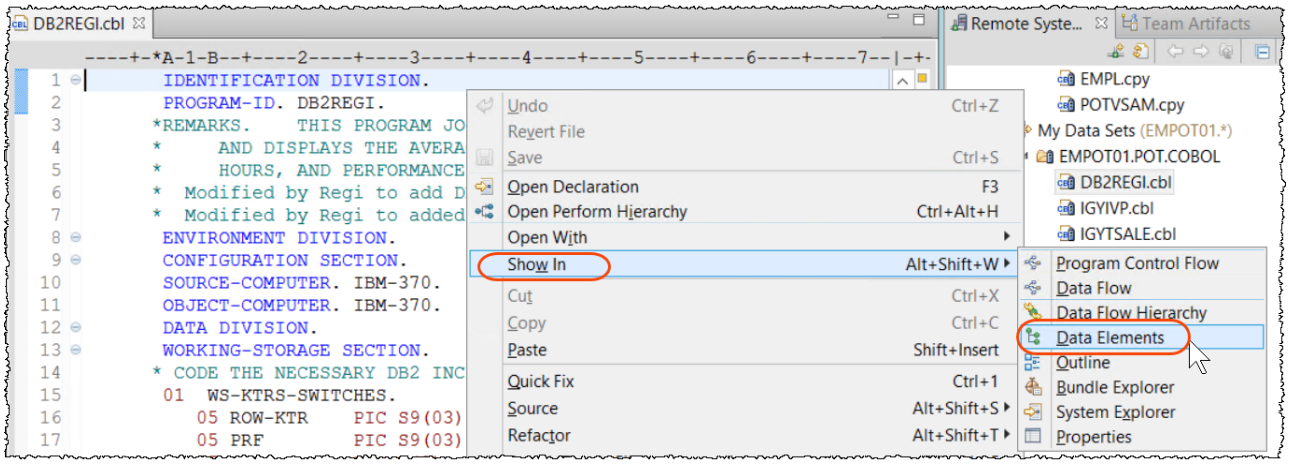
Just select **Windows > Perspective > Reset Perspective** as below:



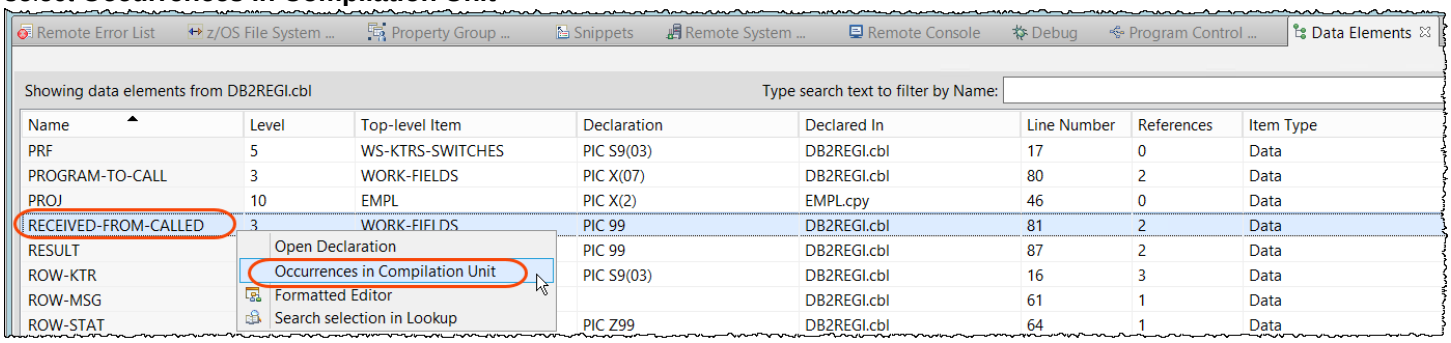
5.1.9 ⏭ Double click "**Program Control Flow"** title to have the diagram smaller.

⏭ Right click again on the editor and select

Show In → Data Elements

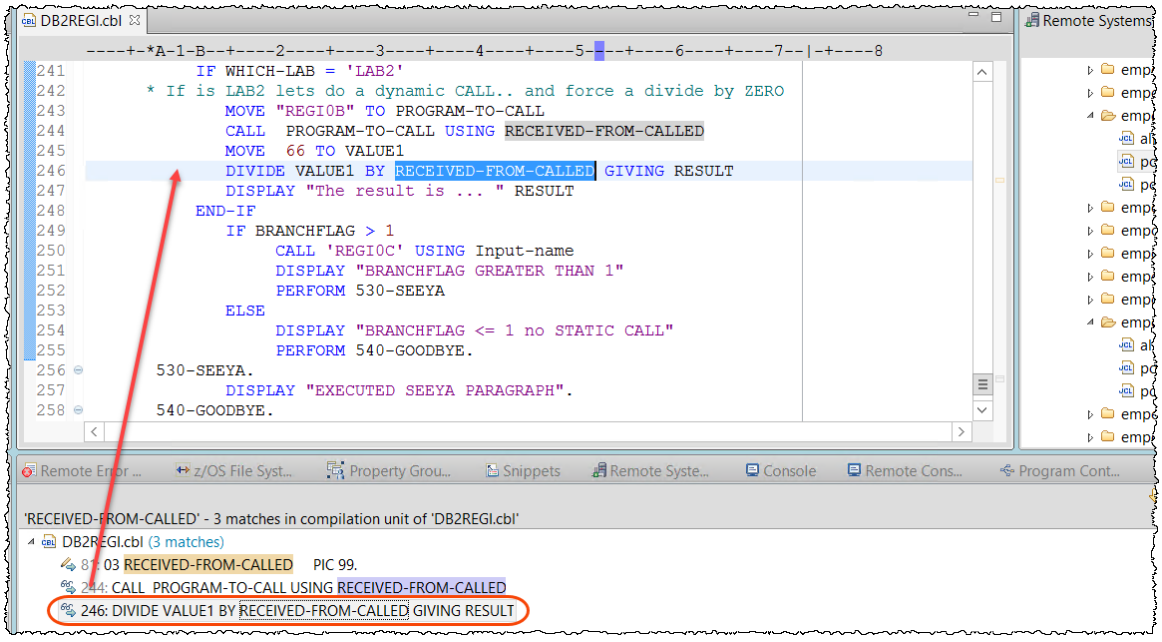


5.1.10 ⏭ On *Data Elements* view, resize the view, scroll down and right click on **RECEIVED-FROM-CALLED** and select **Occurrences in Compilation Unit**



5.1.11 ⏭ Double click on line **246**. It will show in the line that is abending.

Notice the colors. When the variable is referenced, it is blue and when it is modified is brown.

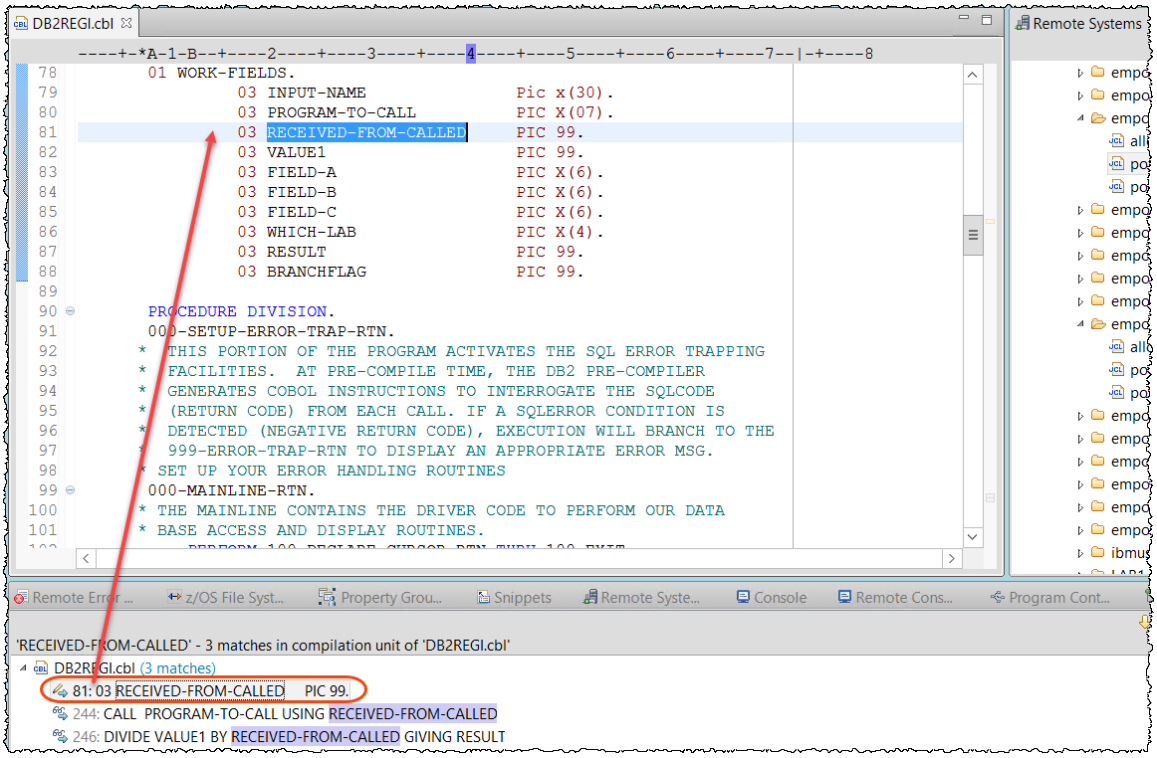


You need to fix the called program named **REGI0B** that is returning 0 on this variable as seen before with Fault Analyzer.

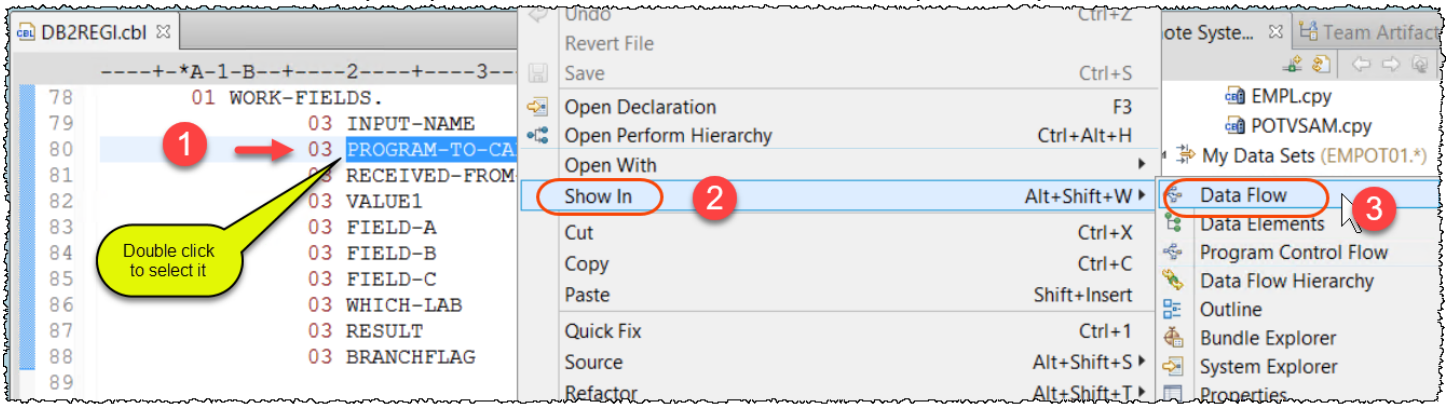
___ Exploring Data Flow

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

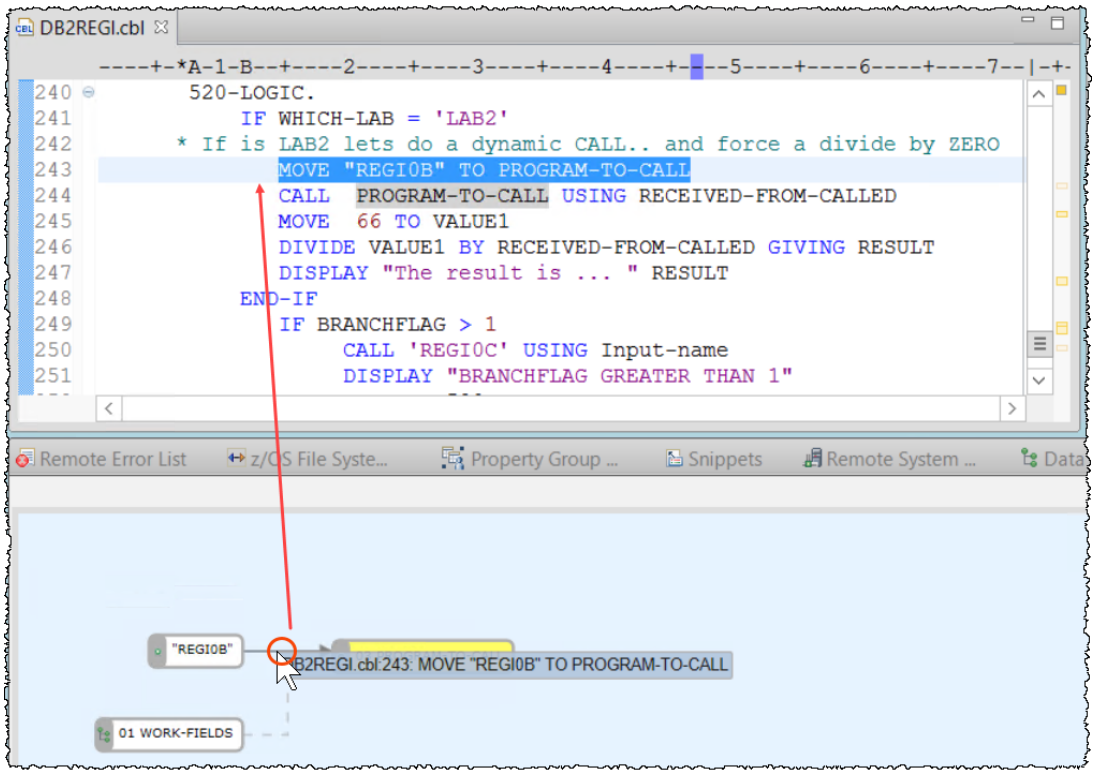
5.1.12 ⏭ Double click on line **81**. It will show where *RECEIVED-FROM-CALLED* is defined.



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



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



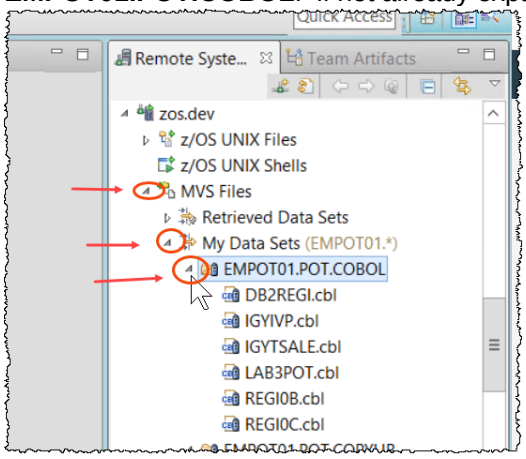
5.1.15 **Close** all opened editors if still opened. (**CTRL + Shift + F4**)
Or just click on the . of each opened editor**.**

___ 5.2 Fixing the program REGI0B

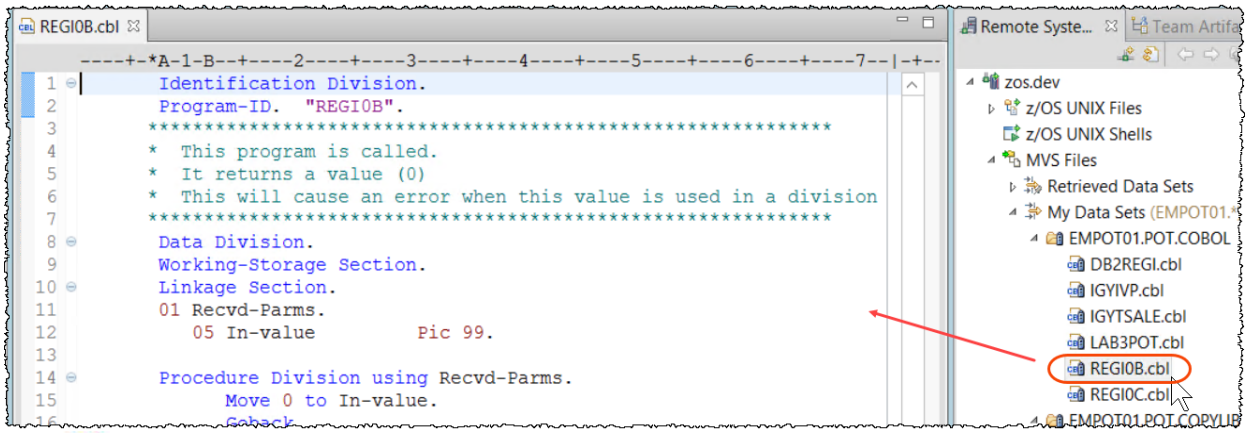
You need to fix the called program named REGI0B that is returning 0 on this variable as seen before with Fault Analyzer and the debugger.

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

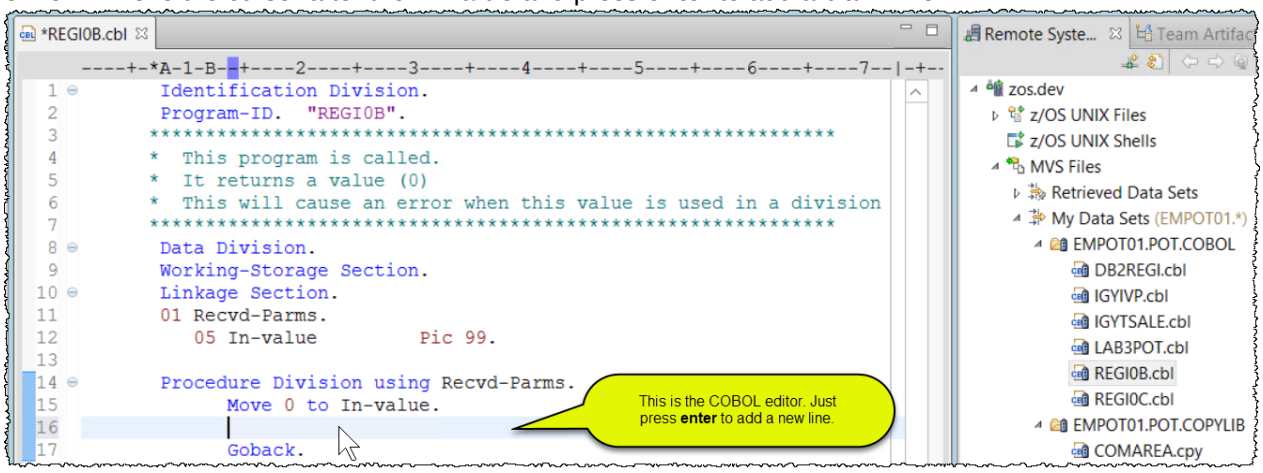
5.2.1 Using *Remote Systems* view **scroll back**, expand **MVS Files, My Data Sets** and **EMPOT01.POT.COBOL**. if not already expanded.



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



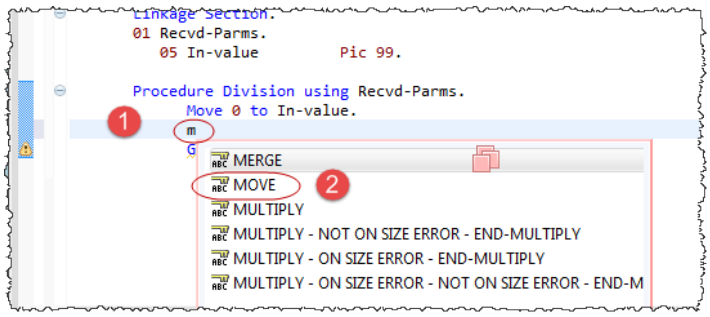
5.2.5 Move the cursor after the **In-value** and press **enter** to add a blank line



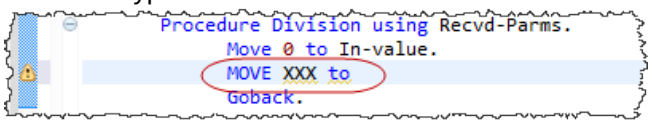
5.2.6 You can now take advantage of the *editor content assist...*

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

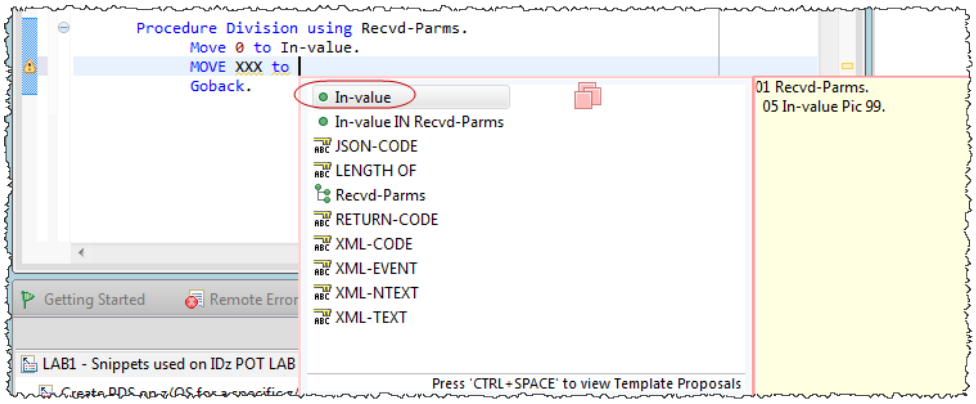
5.2.7 Type **m** and select the statement **MOVE** (can use the mouse double click or select and press enter)



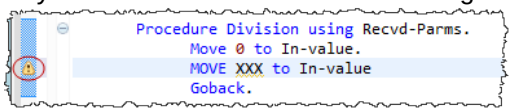
5.2.8 Type "**XXX to**"



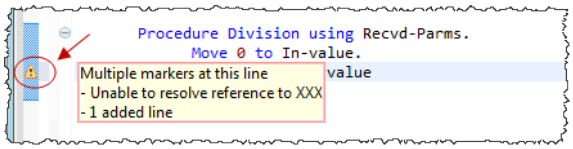
5.2.9 Press **Ctrl + Space** to list the variables and select **In-value**.
Notice that variables could be defined in copybooks.



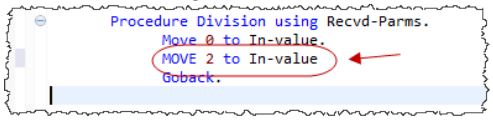
A yellow mark shows that something is wrong.



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



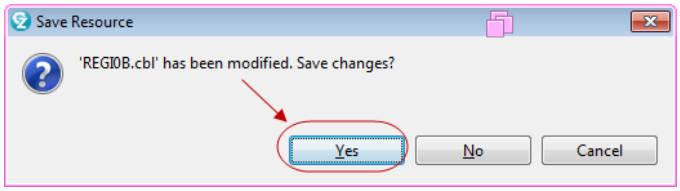
5.2.11 Change from **XXX** to **2** as below. Note that the icon will go away.



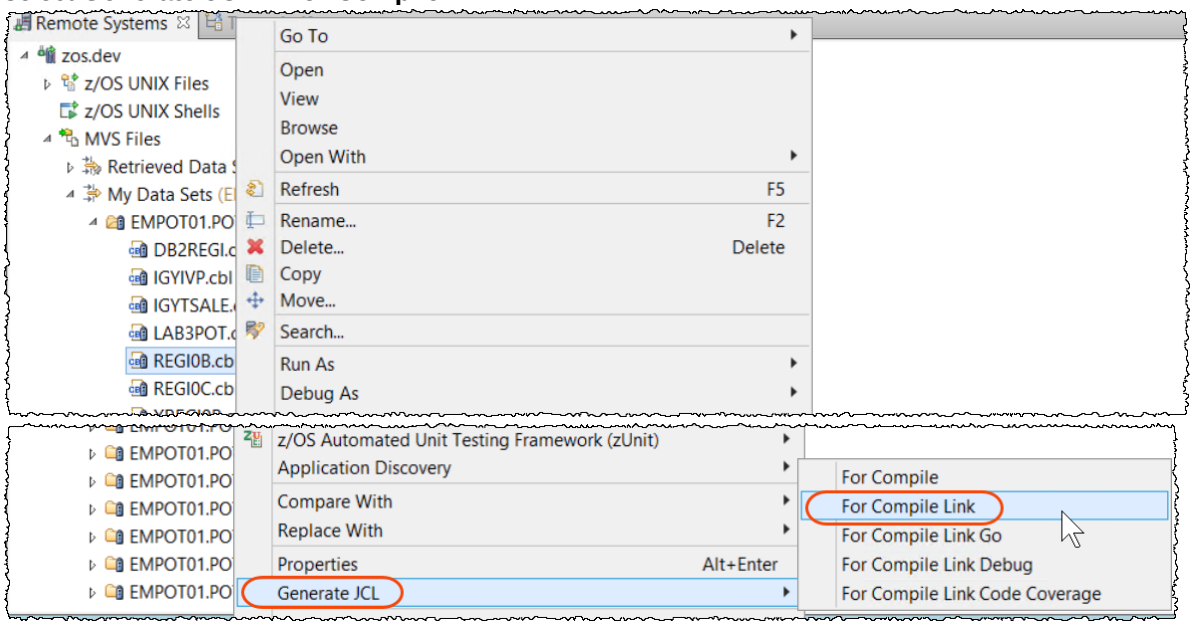
___ 5.3 Compile and link REGI0B.

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

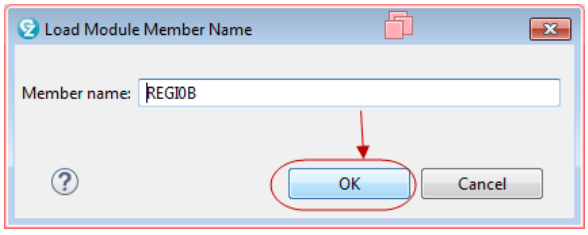
5.3.1 Click **Ctrl + Shift + F4** to close the editors. Or just left click on the . of each opened editor.
Click **Yes** to Save Changes.



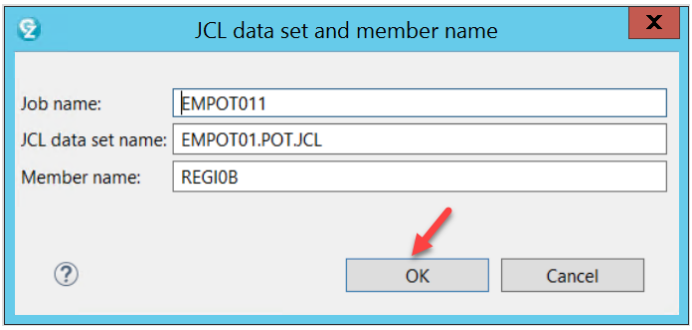
5.3.2 Using the Remote Systems view **right click** on **REGI0B.cbl** and select **Generate JCL > For Compile Link** .



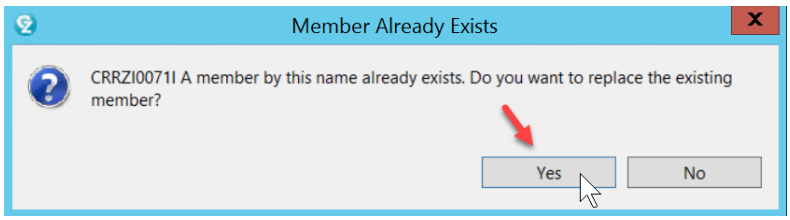
5.3.3 Accept the default member name and click **OK**



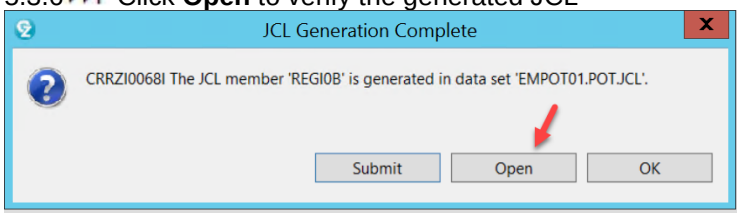
5.3.4 Accept the default and click **OK**



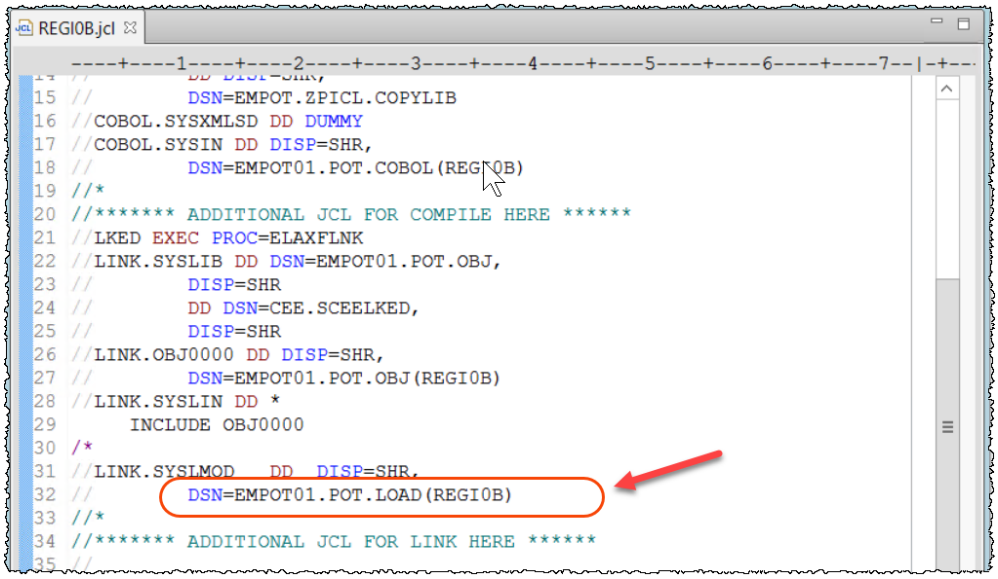
5.3.5 Click **Yes** if the member already exists



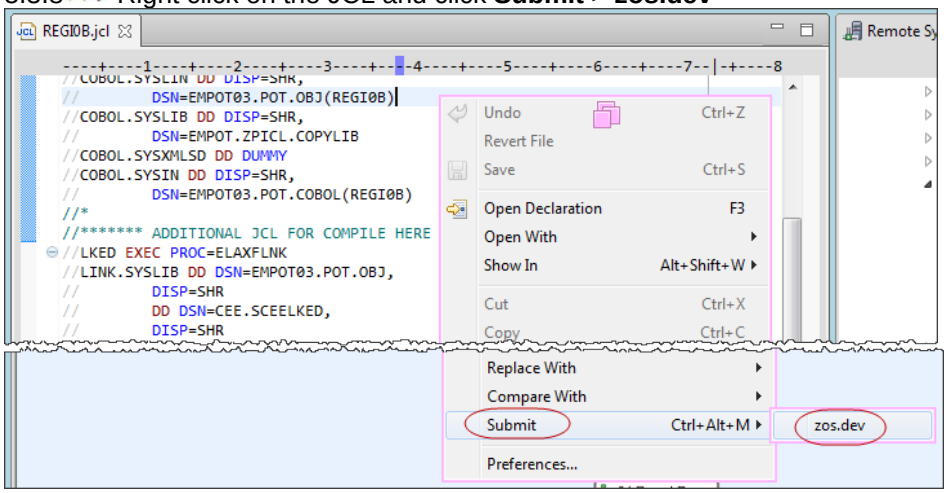
5.3.6 Click **Open** to verify the generated JCL



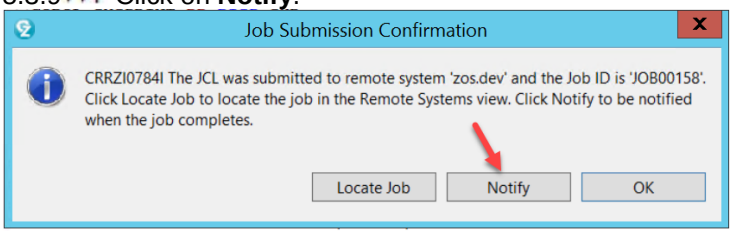
5.3.7 **Scroll down** and notice that the *Load module* will be created on the PDS: **EMPOT01.POT.LOAD.**



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

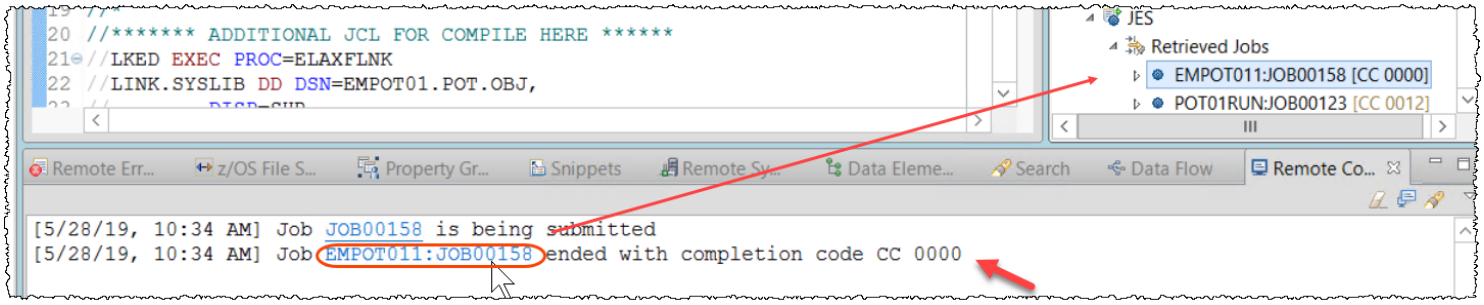


5.3.9 Click on **Notify**.



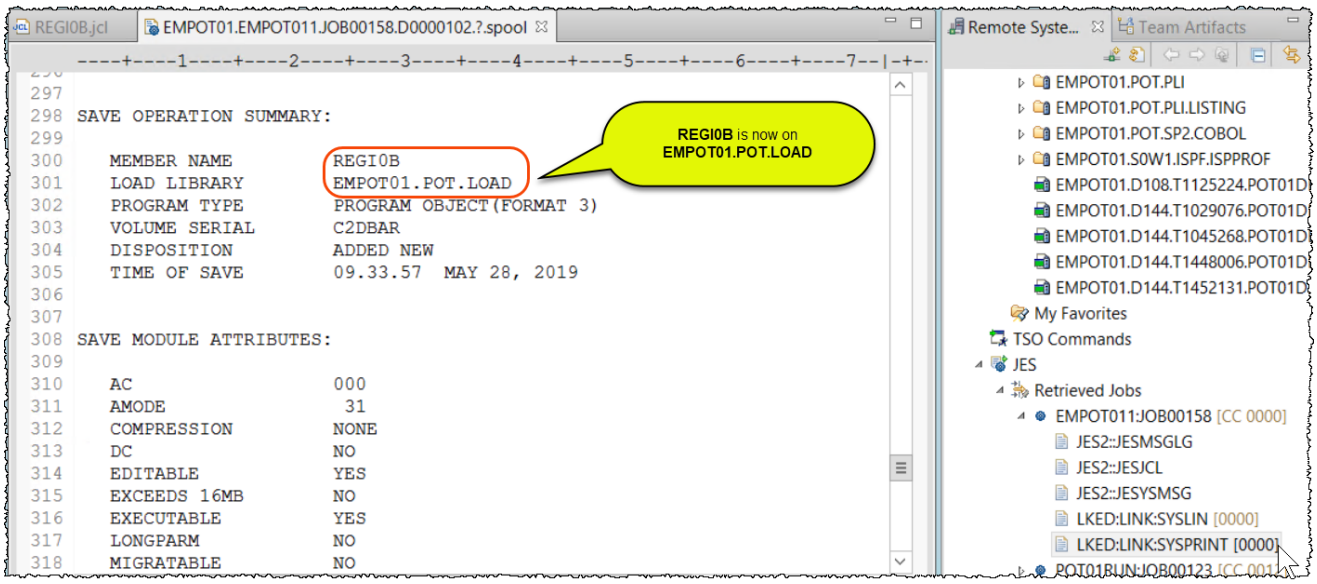
5.3.10 You MUST have **000** as return code.

⏭ Click on **EMPOT011:JOB00xxx**



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

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

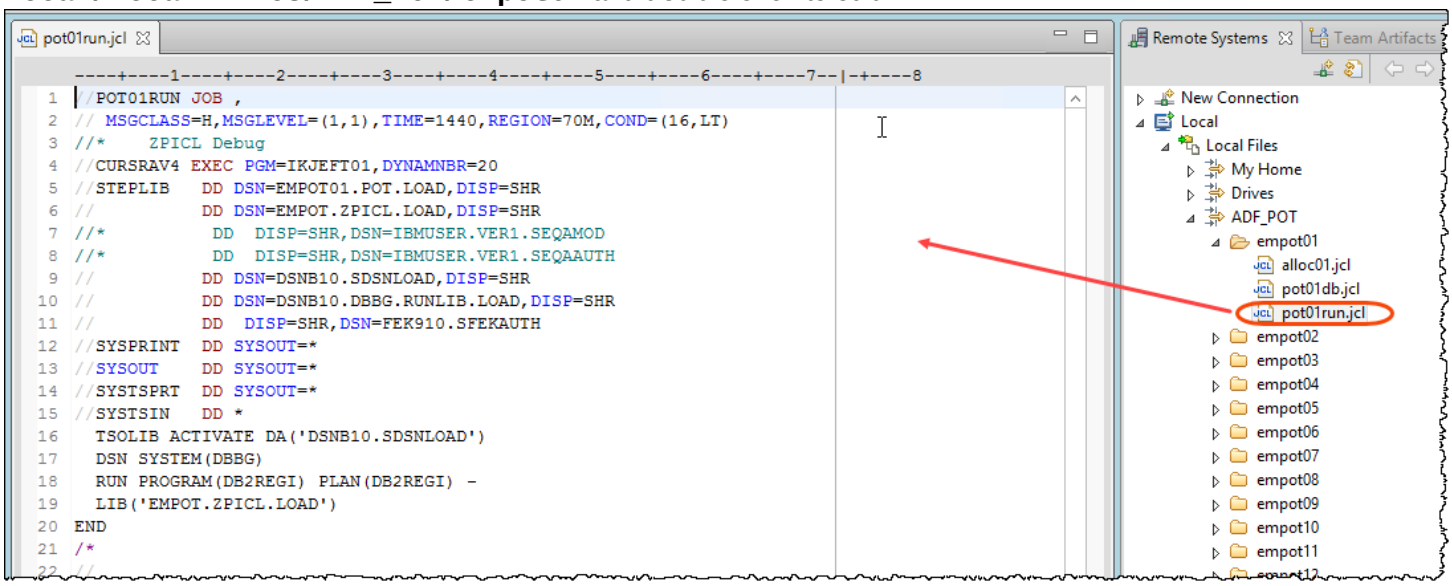


5.3.10 ⏭ Use **Ctrl + Shift + F4** to close all editors.
Or just click on the ⊠. of each opened editor

___ 5.4 Execute the COBOL/DB2 program

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

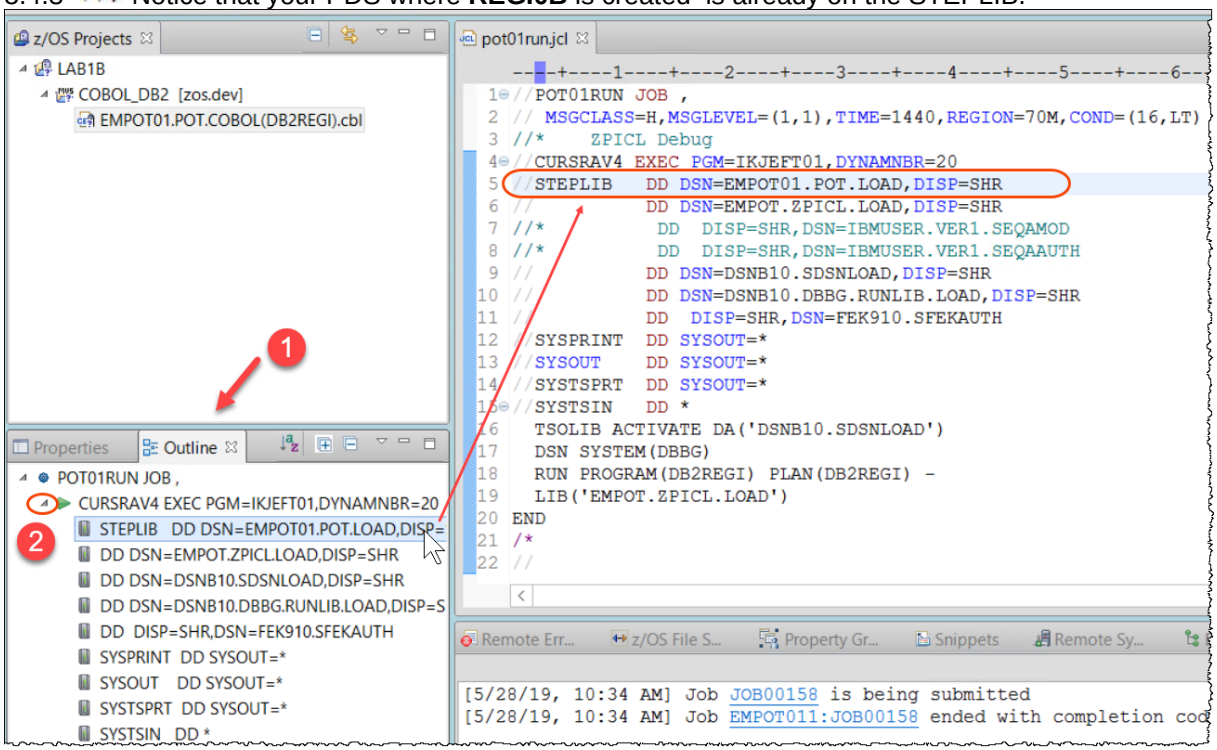
5.4.1 ⏭ Using *Remote System View*, scroll back to locate the file **`pot01run.jcl`** under the folder **`Local/Local Files/ADF_POT/empot01`** and **double click** to edit..



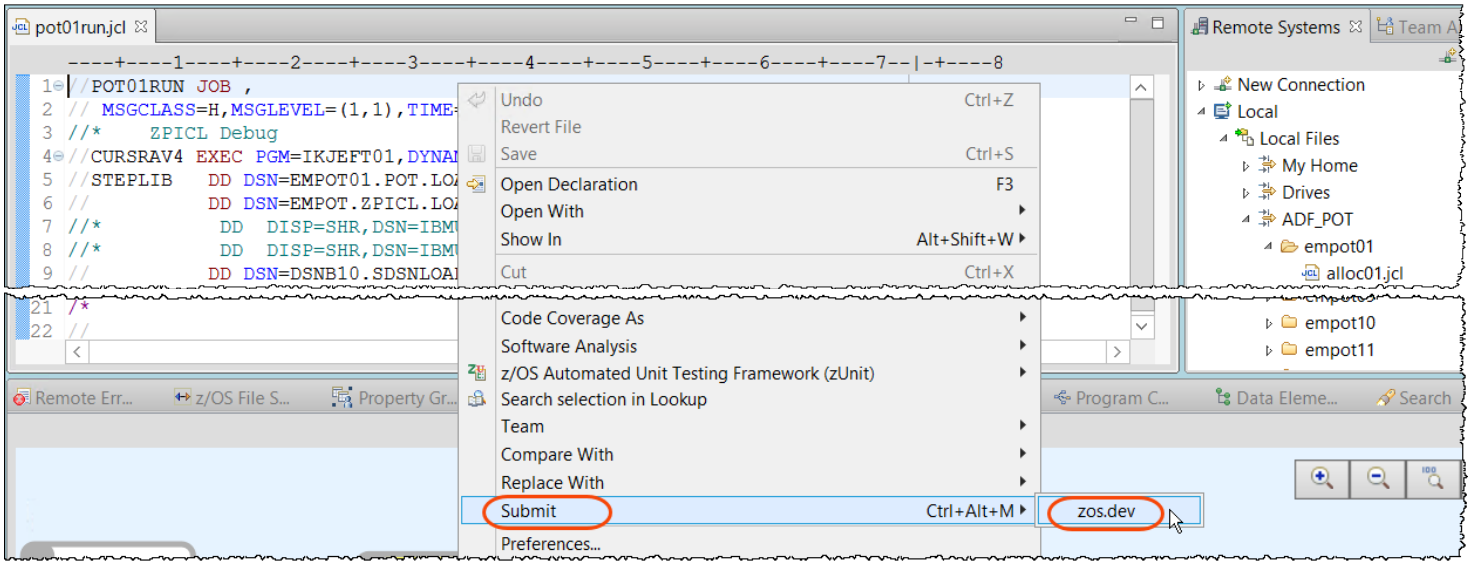
5.4.2 Notice on bottom left corner the same *Outline* view that you have for the COBOL program. It will help to navigate on large JCL members.

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

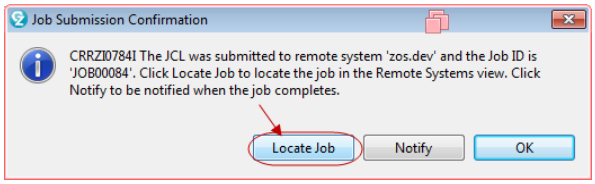
5.4.3 ⏭ Notice that your PDS where **REGI0B** is created is already on the STEPLIB.



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



5.4.5 ⏭ Click on **Locate Job**.



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

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

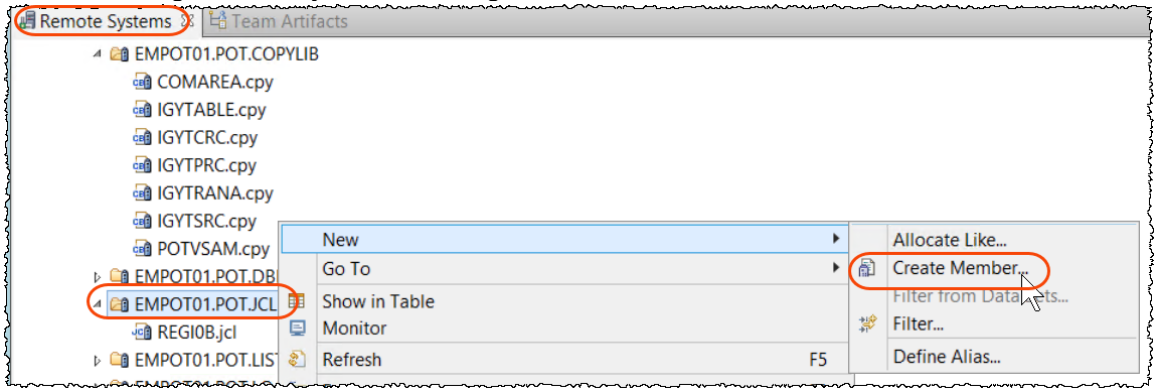
Code coverage analyzes a running program and generates a report of statements that were executed, compared to the total number of executable lines.
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode, A sample JCL procedure, to process a single program run,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

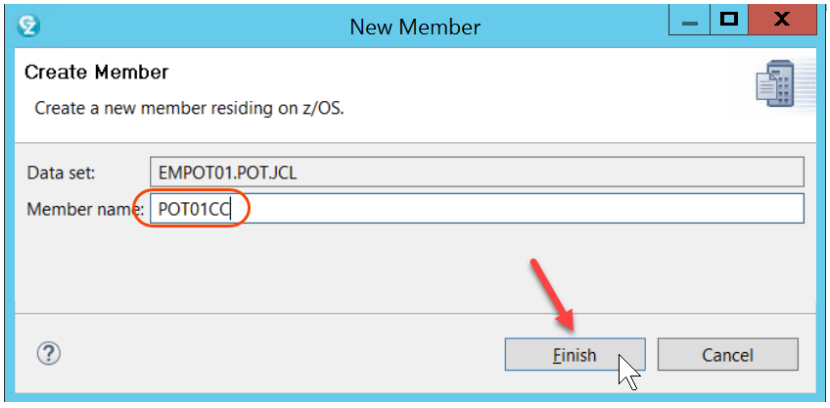
___ 6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

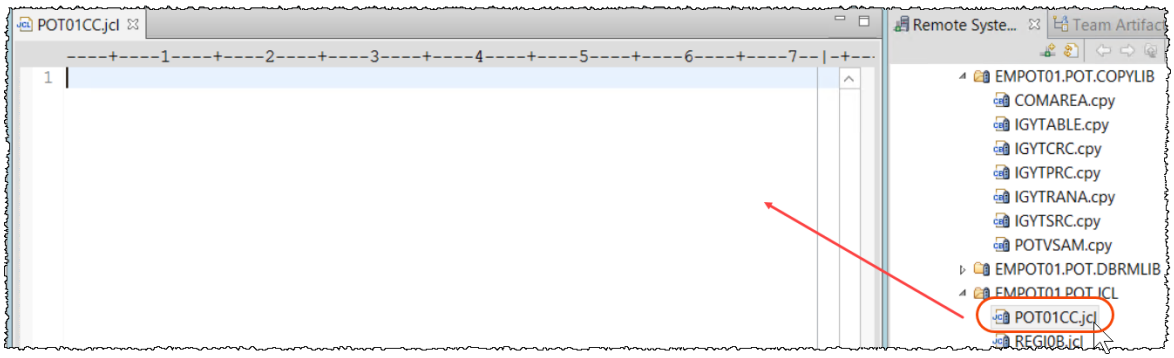
6.1.1 Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member…**



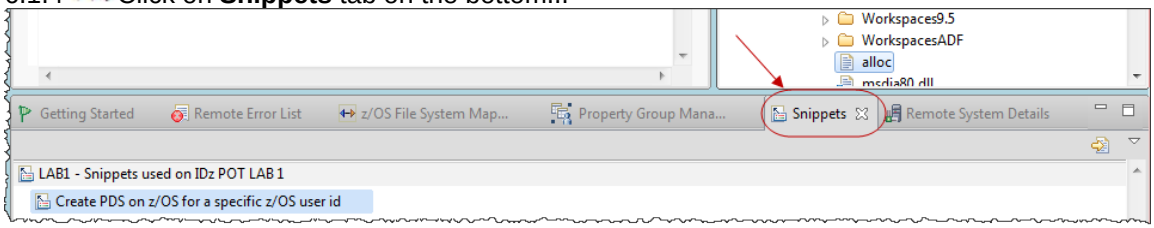
6.1.2 Name it as **POT01CC** and click **Finish**



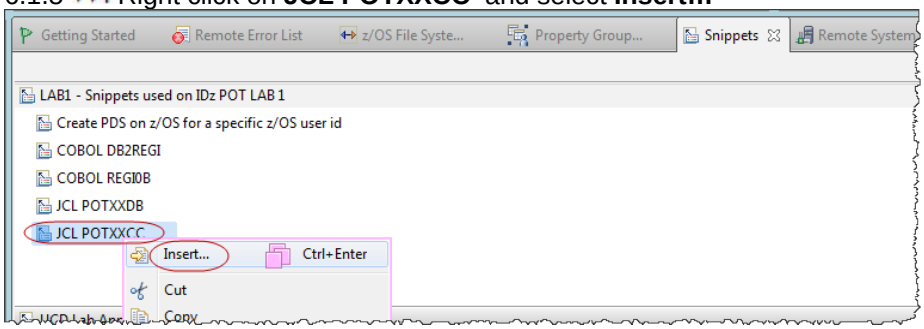
6.1.3 Double click on **POT01CC** to edit it



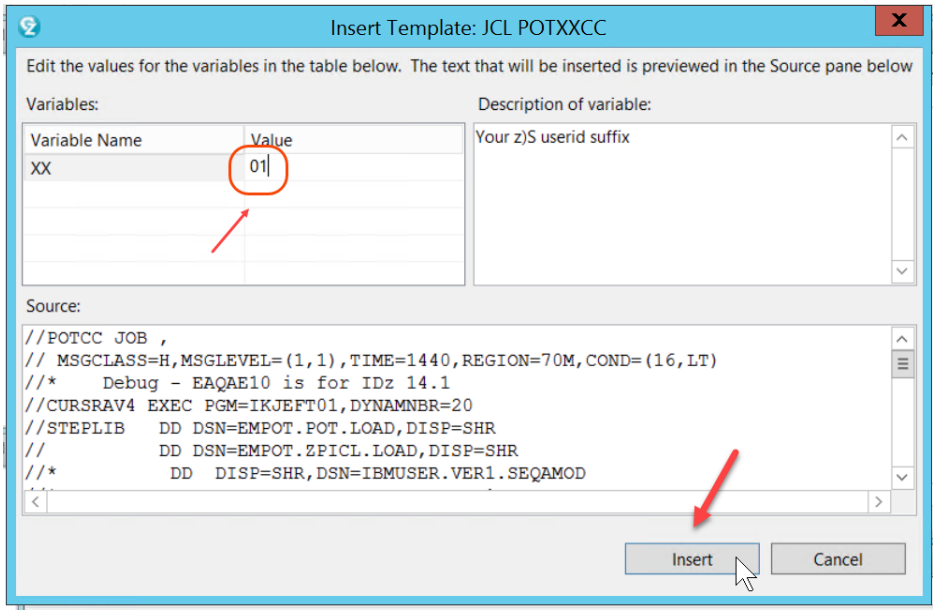
6.1.4 Click on **Snippets** tab on the bottom...



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

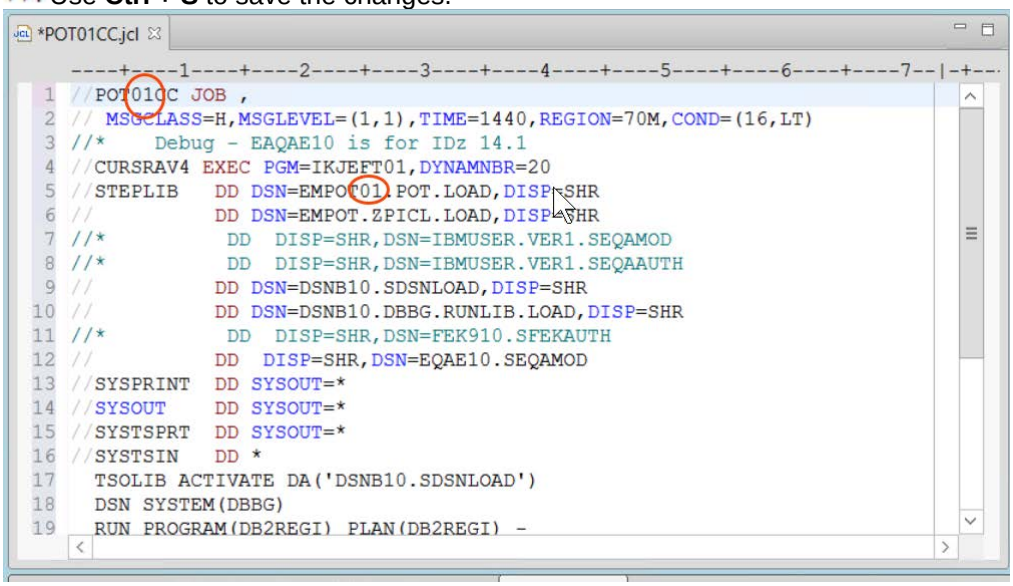


6.1.6 When the *Insert* dialog opens, type **01** in the *Value* and click **Insert**



6.1.7 . Notice that the Snippets variables will be replaced.

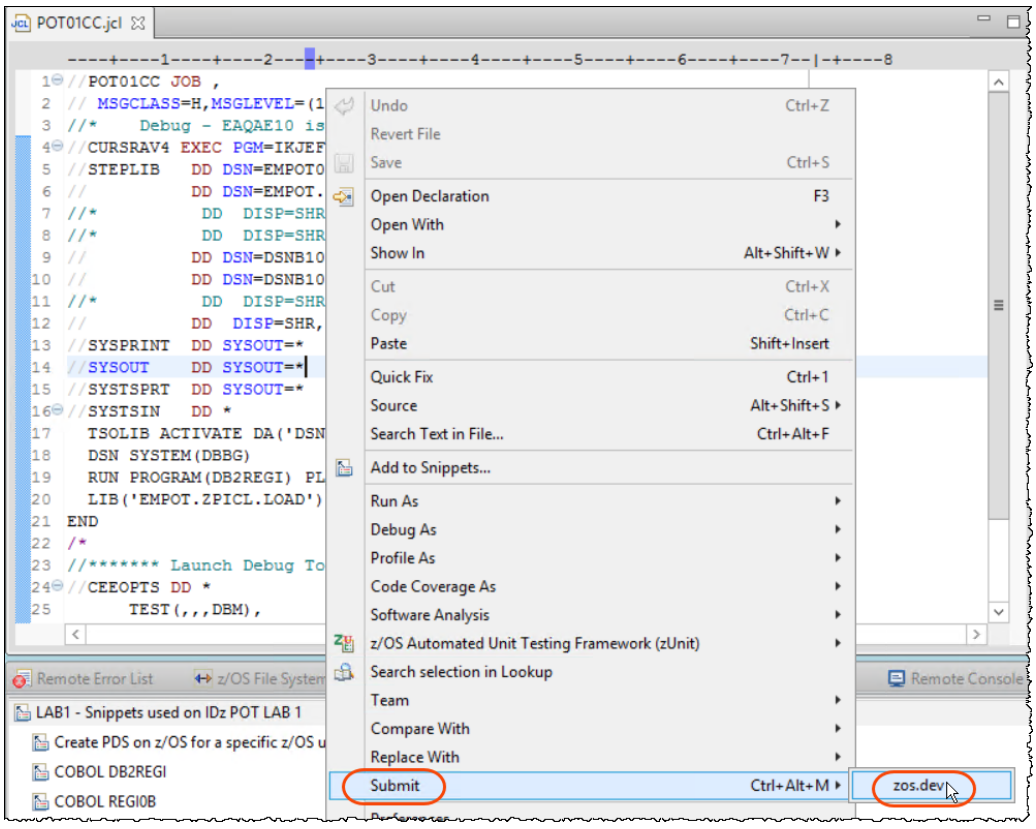
⏭ Use **Ctrl + S** to save the changes.



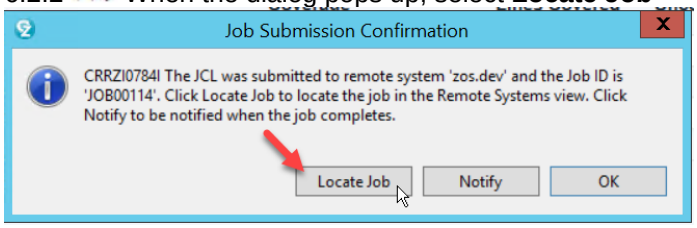
___ 6.2 Submit the JCL for z/OS execution

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

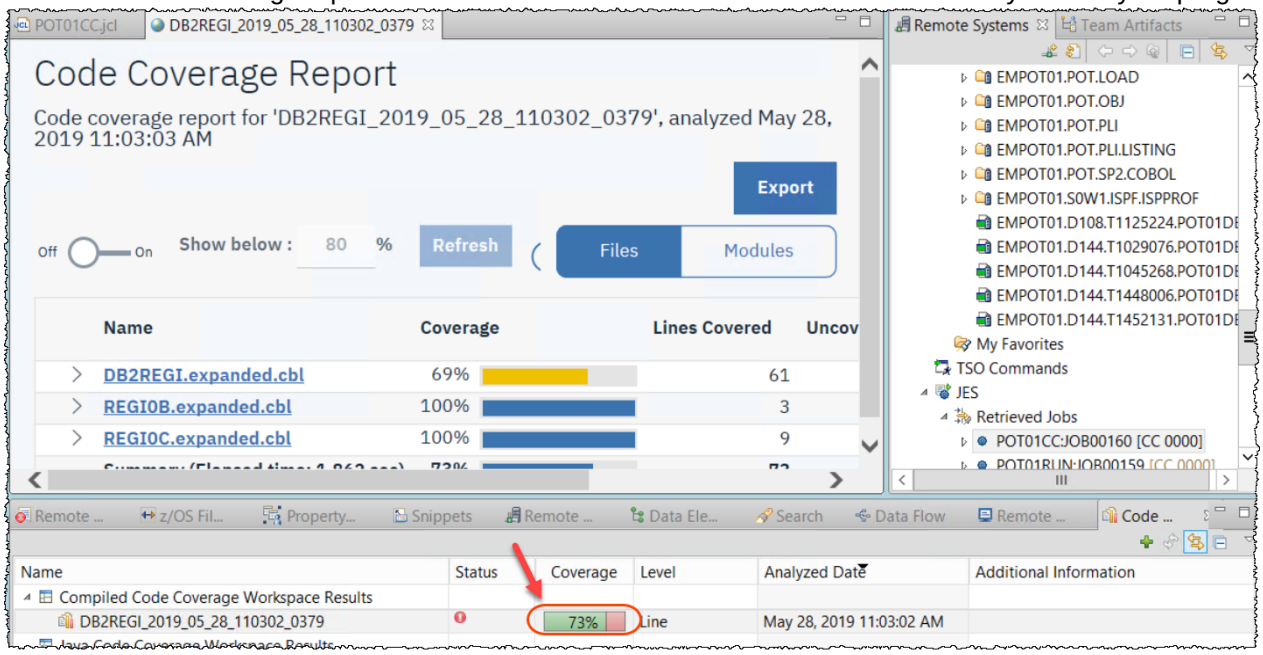
6.2.1 ⏭ **Right click** on the JCL being edited and select **Submit** > **zos.dev**



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

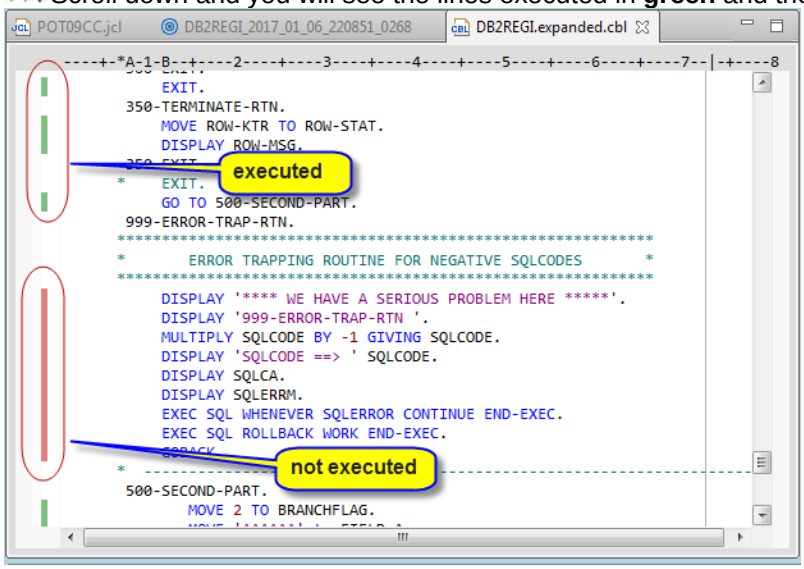


6.2.3 The code coverage report will be created.You can see that this test covered only **73%** of your program



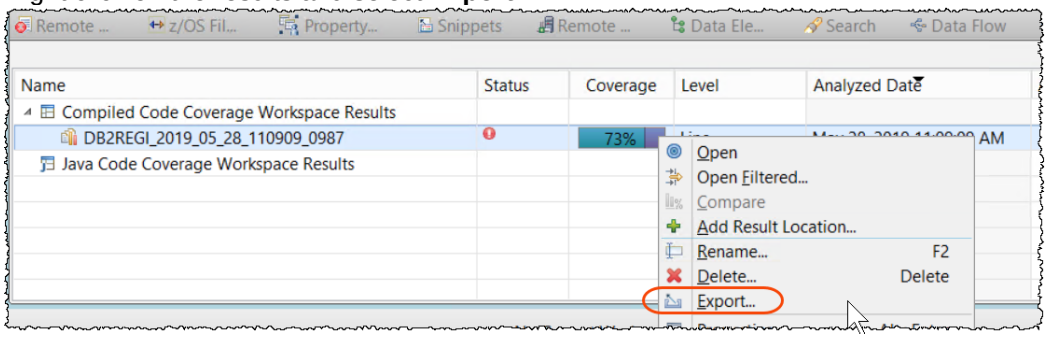
6.2.4 ⏭ Double click on **DB2REGI.expanded.cbl**

⏭ Scroll down and you will see the lines executed in **green** and the lines not executed in **Red**.

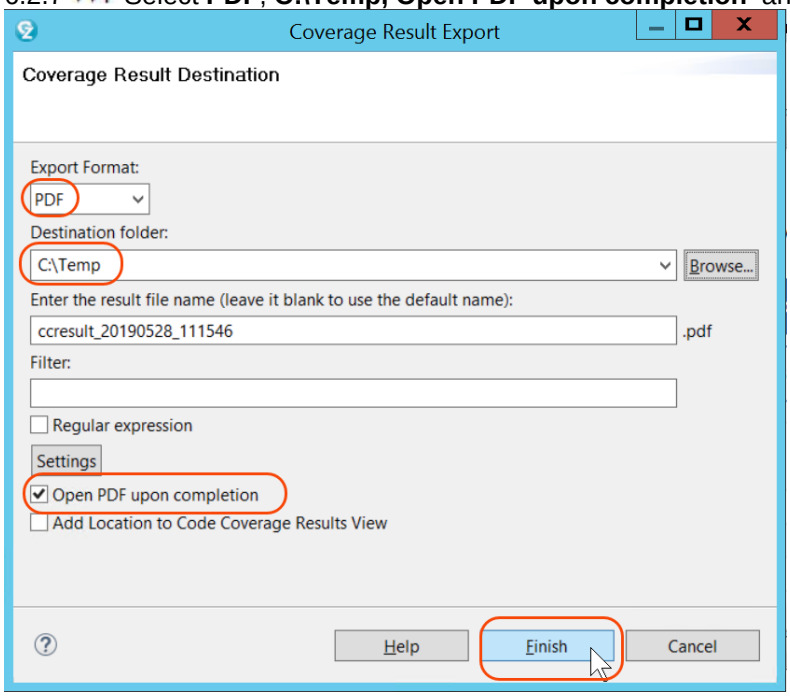


6.2.5 As you can see the error routine on the picture above is not tested..
Imagine how bad if you go to production and never test this condition ....

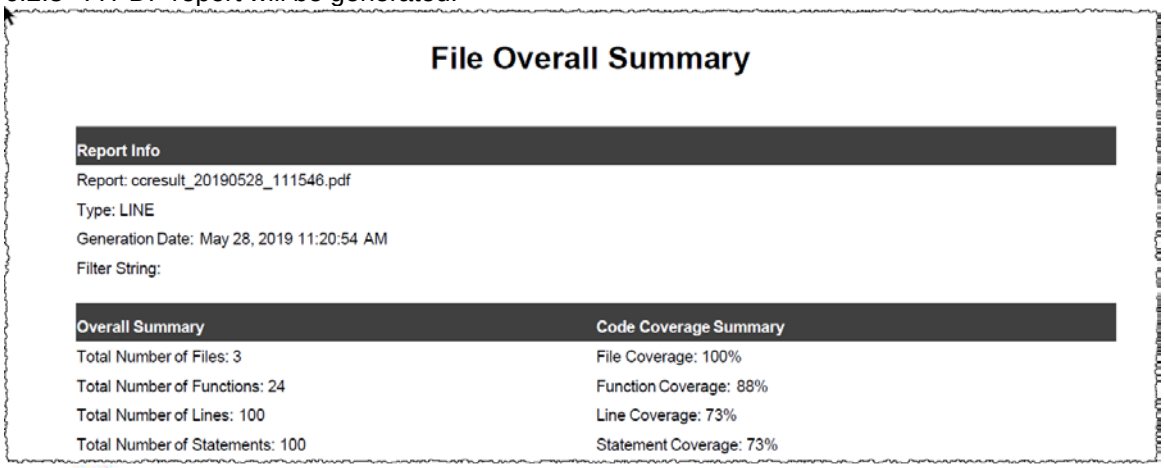
6.2.6 You could also create a PDF report as documentation or export this data to other products (like ADDI or SonarQube)
Right click on the results and select **Export** ..



6.2.7 Select **PDF**, **C:\Temp, Open PDF upon completion** and **Finish**.



6.2.8 A PDF report will be generated.



File Overall Summary

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

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

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

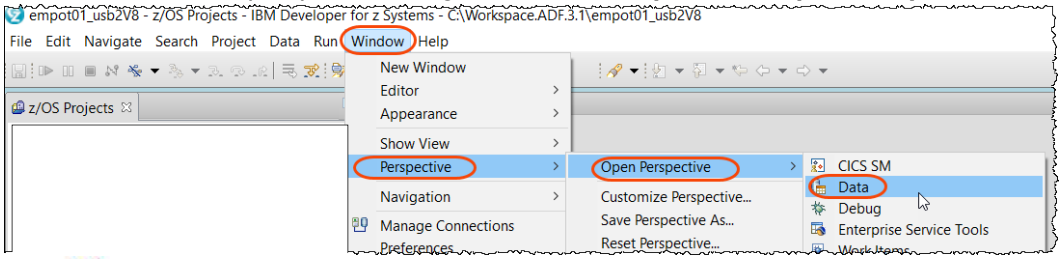
Section 7. (Optional) Executing SQL statements when editing the program.

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

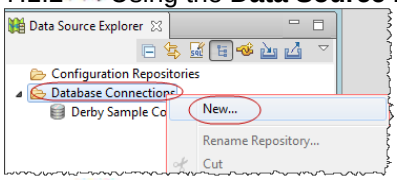
___ 7.1 Creating a connection to the DB2 on z/OS

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

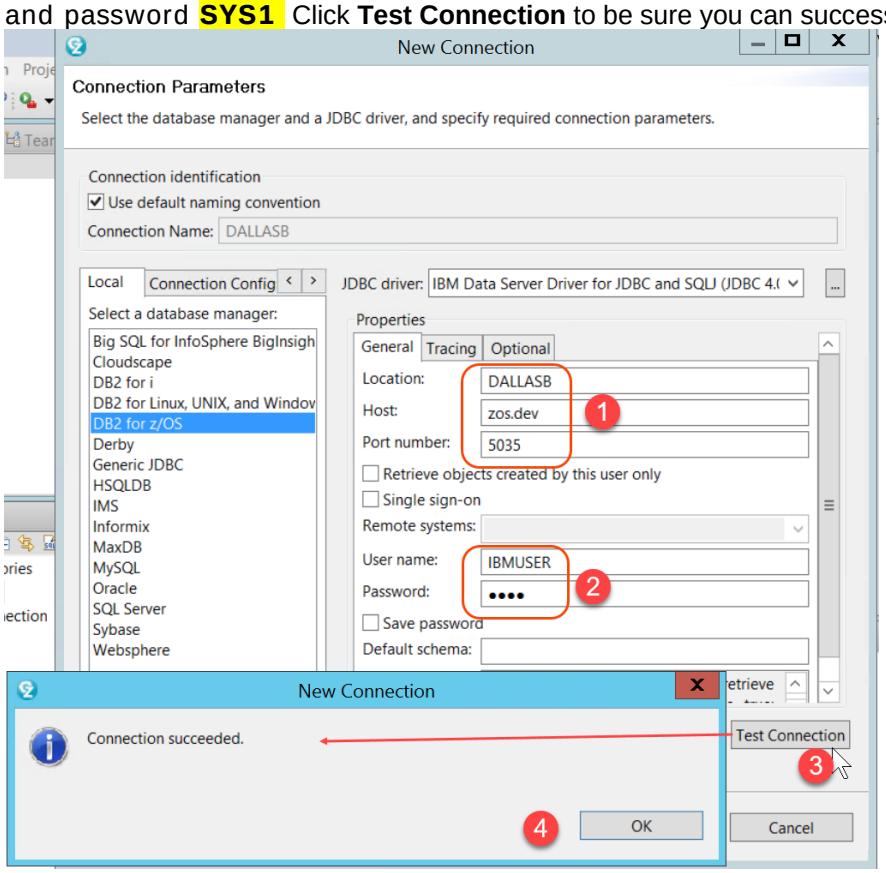
7.1.1 ⏭ Go to **Data** perspective using **Window > Perspective > Open Perspective > Data**



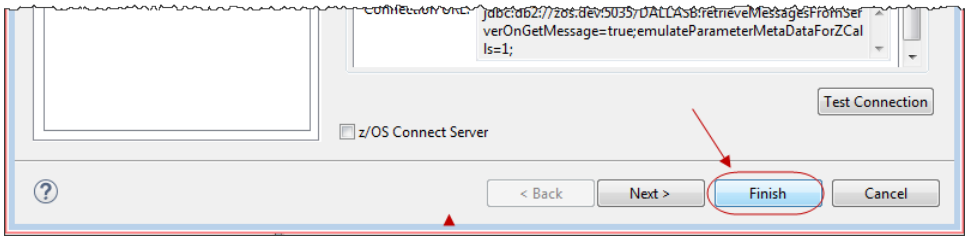
7.1.2 ⏭ Using the **Data Source Explorer** view, create a new z/OS DB2 connection:



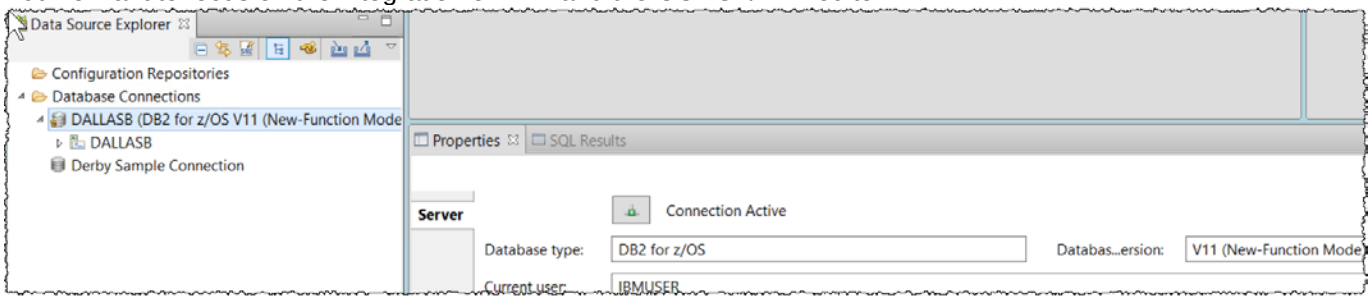
7.1.3 ⏭ Select **DB2 for z/OS,** use Location as **DALLASB**, Host is **zos.dev** and port is **5035**. , **IBMUSER** and password **SYS1** Click **Test Connection** to be sure you can successfully connect to the DB2.



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



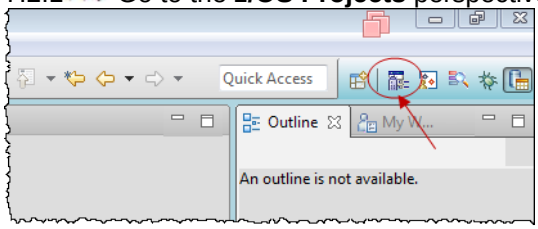
The connection is created. You could see tables, modify contents etc..
But we want to focus on the integration of DB2 and the COBOL/DB2 editor.



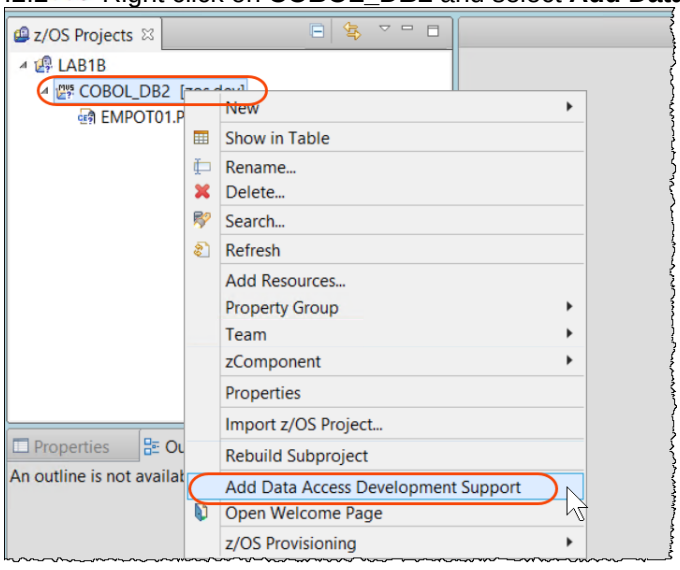
___ 7.2 Running a SQL query from COBOL program

You will need to have authorization to run the queries.

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

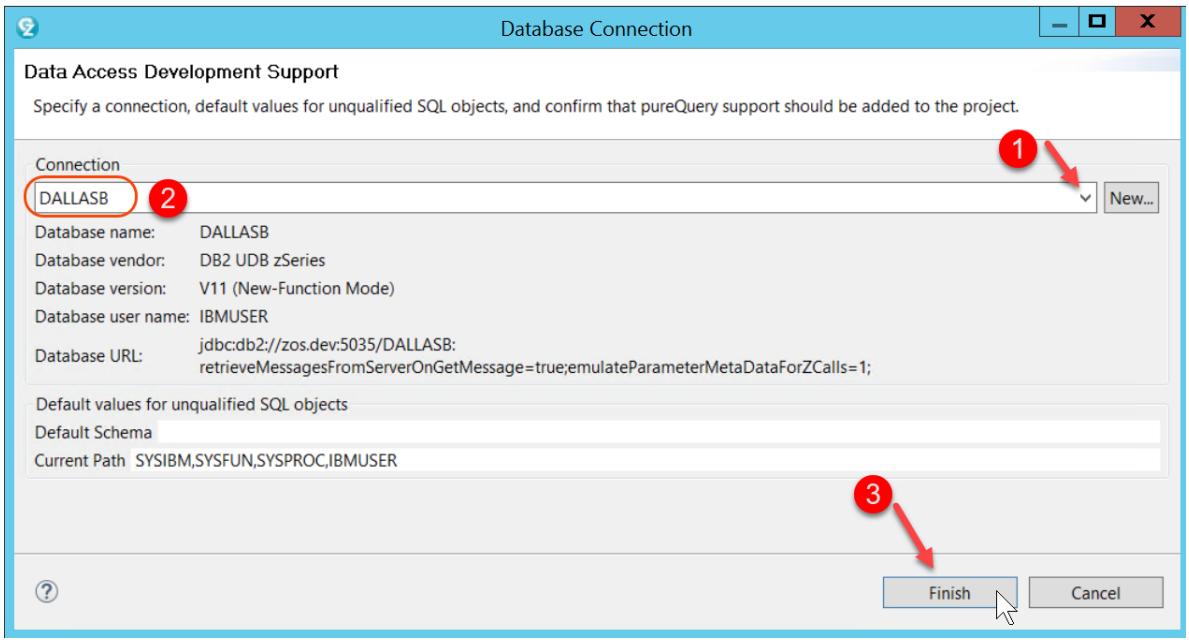


7.2.2 Right click on **COBOL_DB2** and select **Add Data Access Development Support**

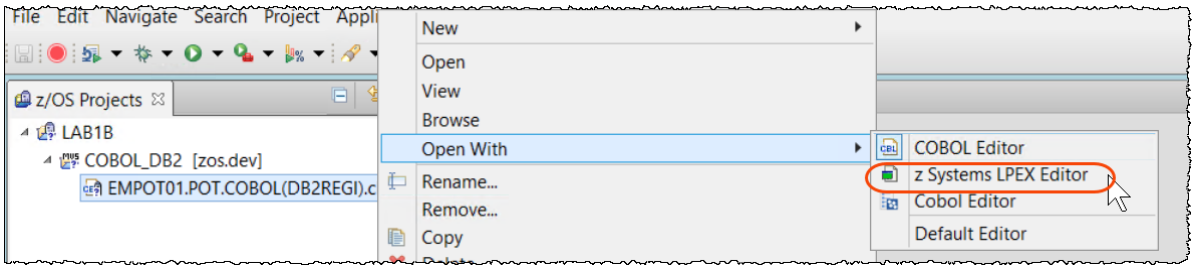


,

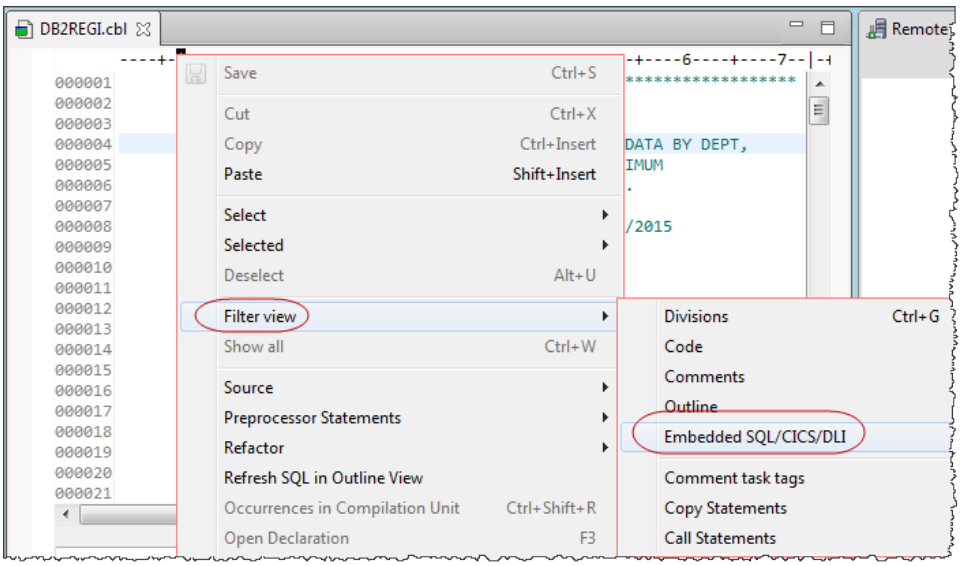
7.2.3 Select the connection created **DALLASB** and click **Finish**



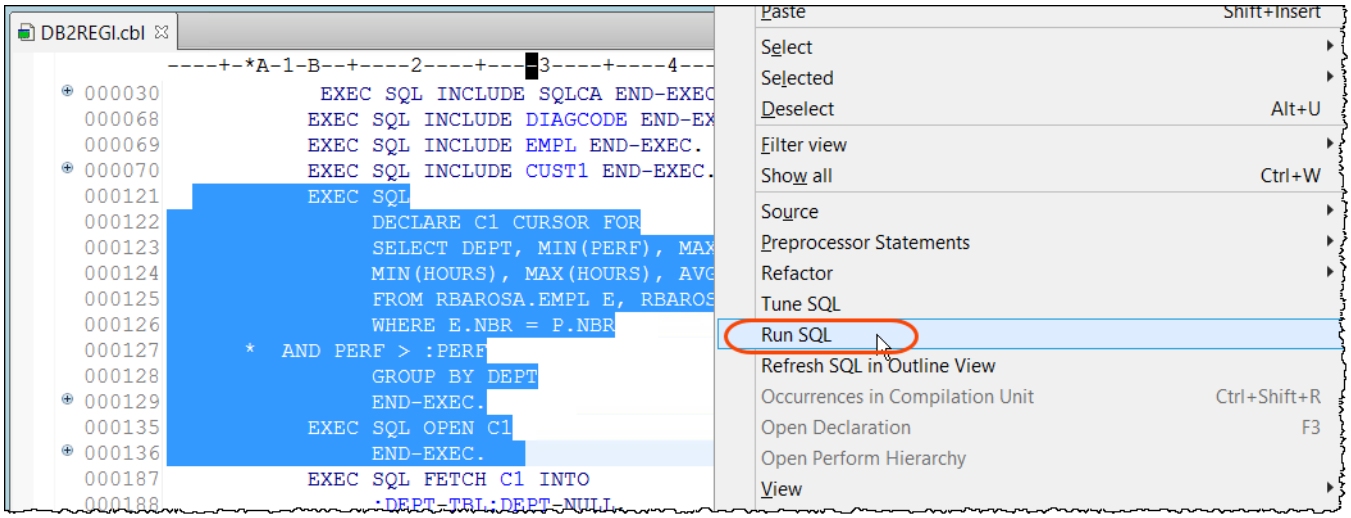
7.2.4 Right click on program **DB2REGI.cbl** and select **Open With** > **z Systems LPEX editor**



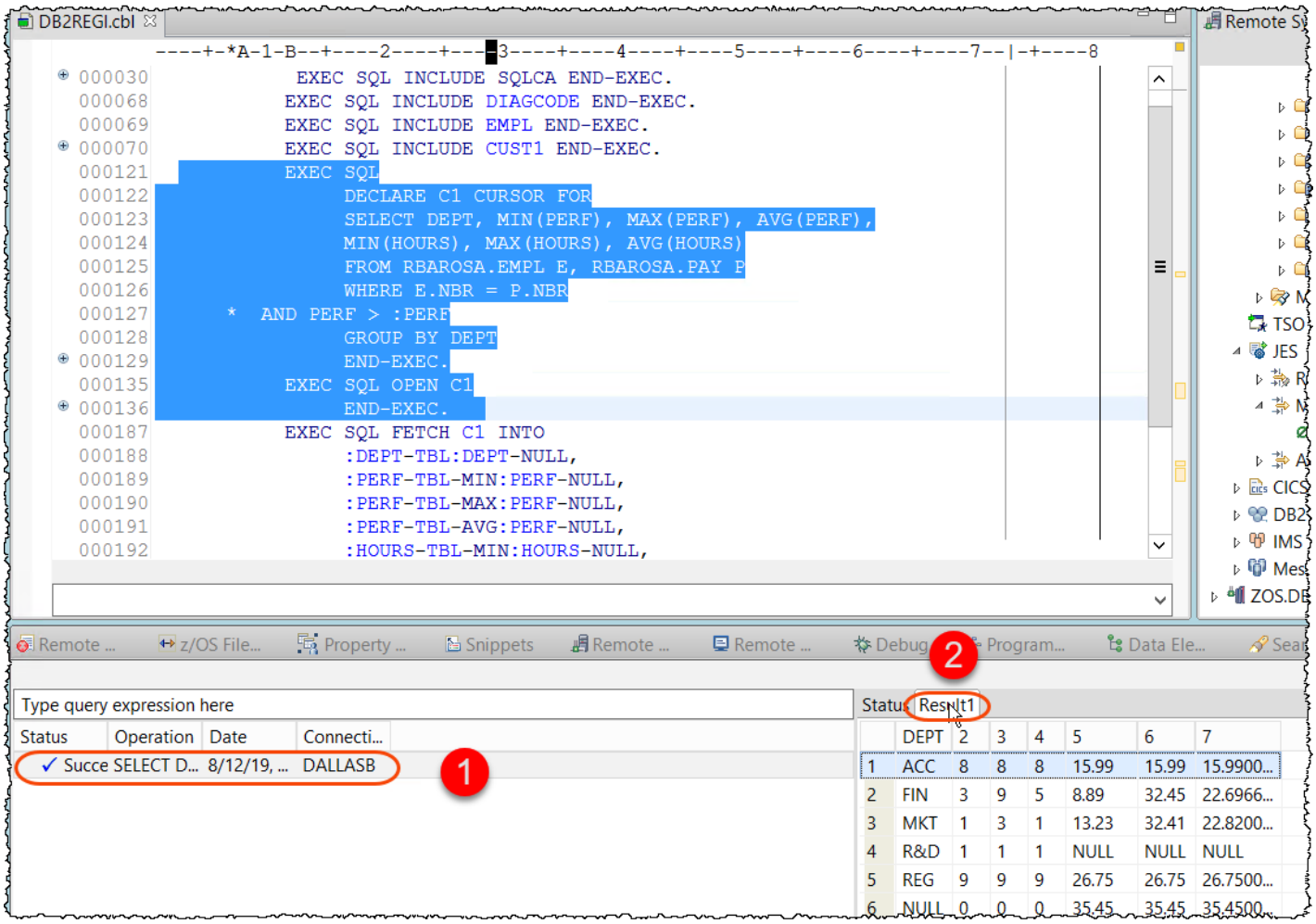
7.2.5 Right click and select **Filter View > Embedded SQL/CICS/DLI**



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

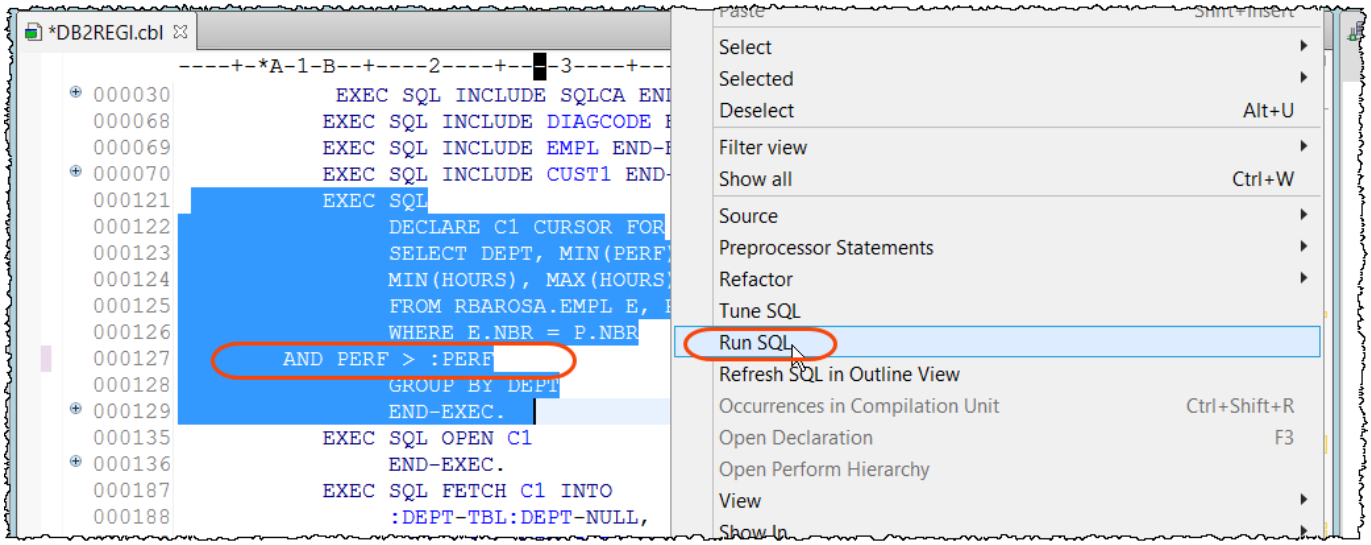


7.2.7 Click as below and you will have the query results

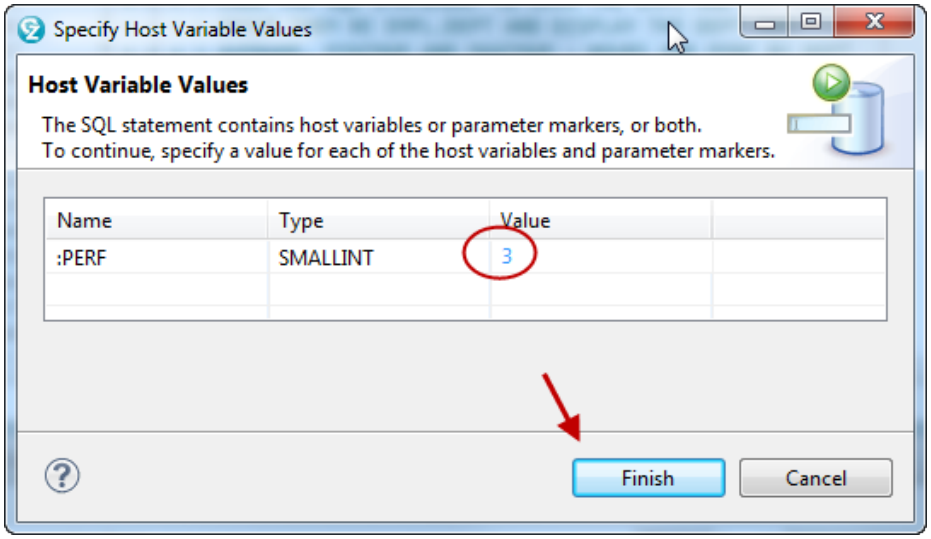


7.2.8 Modifying the SQL query..

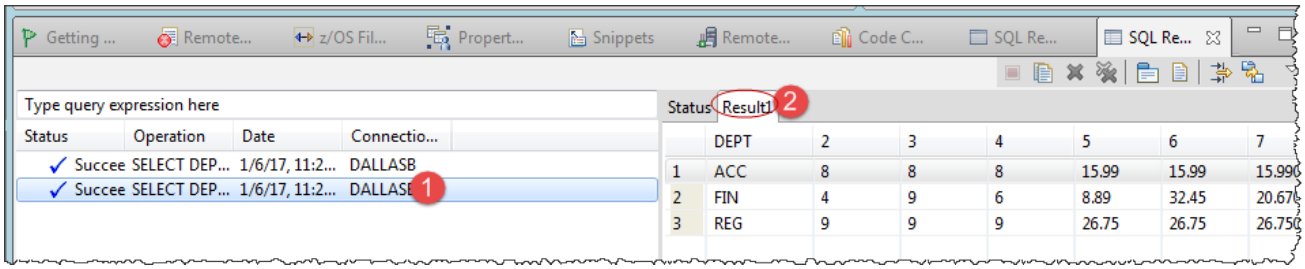
Remove the COBOL comment (***)** from the line **127**,
Select the SQL statements again right click and choose **Run SQL**



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

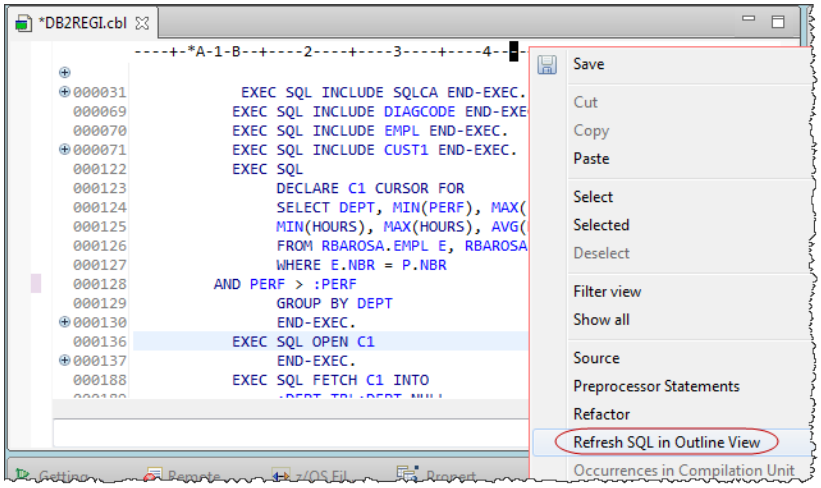


7.2.10 The new results now will be:

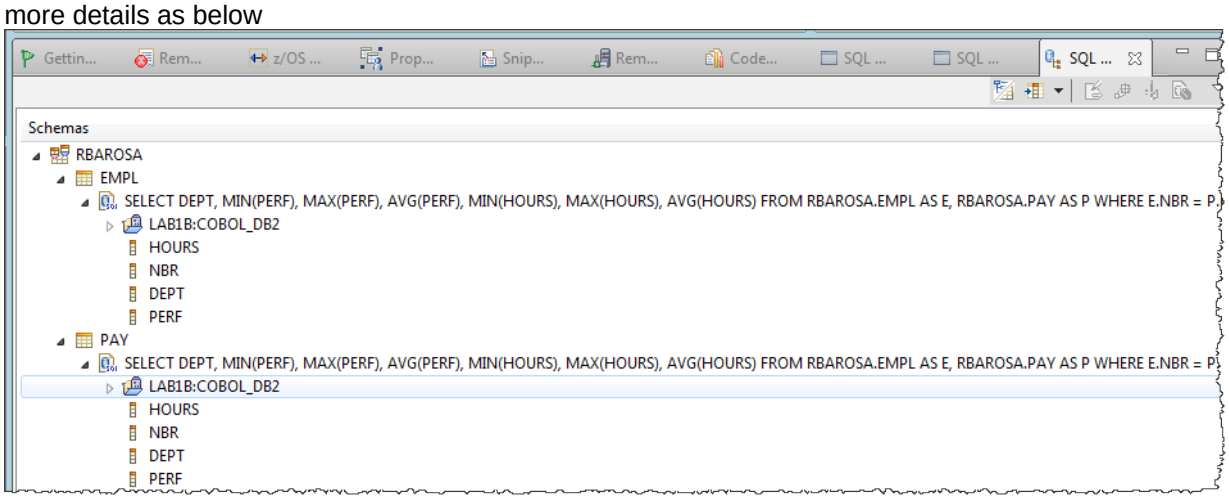


___ 7.3 Using the SQL Outline view

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

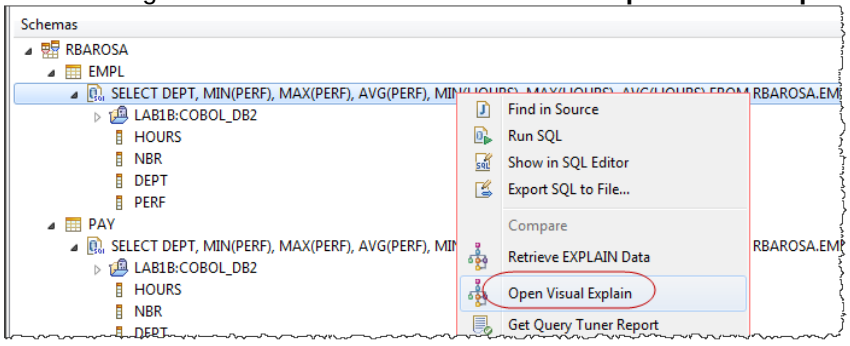


7.3.2 Using *SQL Outline View*, expand **RBAROSA**, **EMPL, PAY** and **the SQL statements** and you will have more details as below

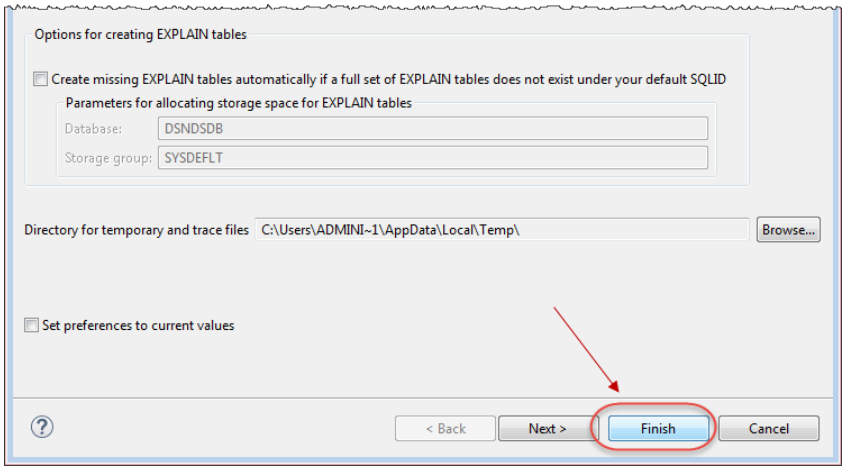


___ 7.4 Using SQL Visual Explain

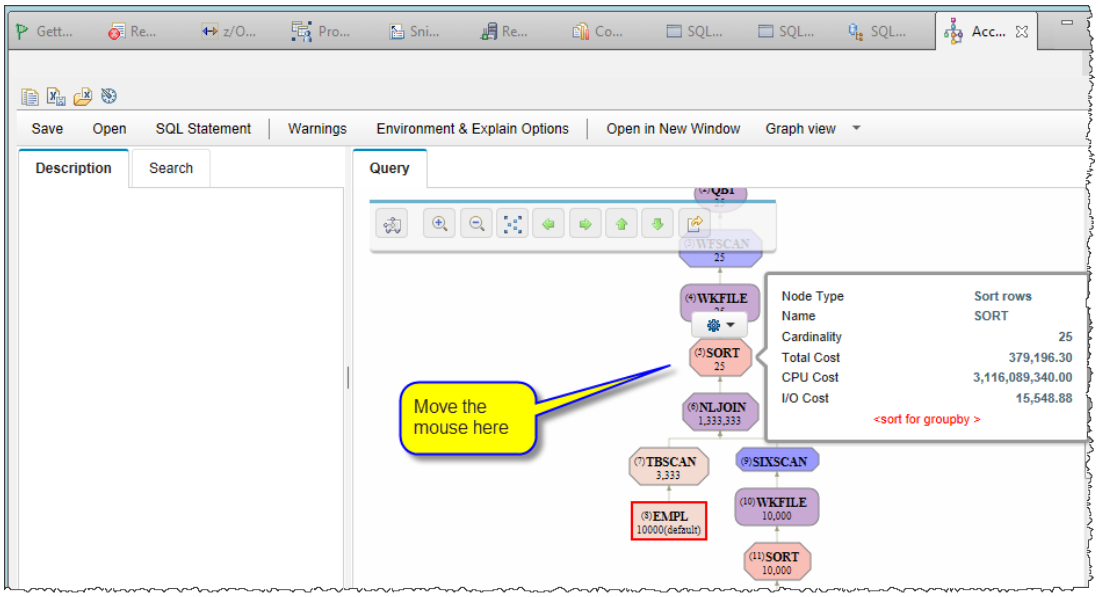
7.4.1 Right click on the first Select and choose **Open Visual Explain**



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

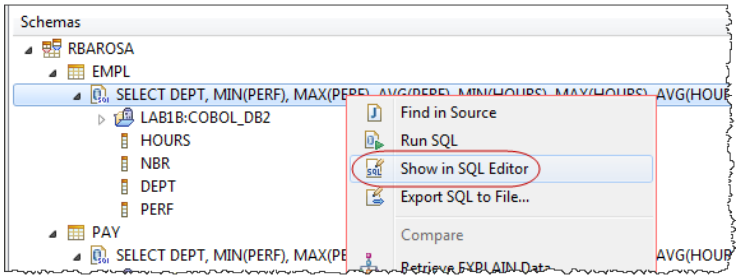


7.4.3 Double click on the tab to have a bigger view.
You will have the report below to work with the SQL explain

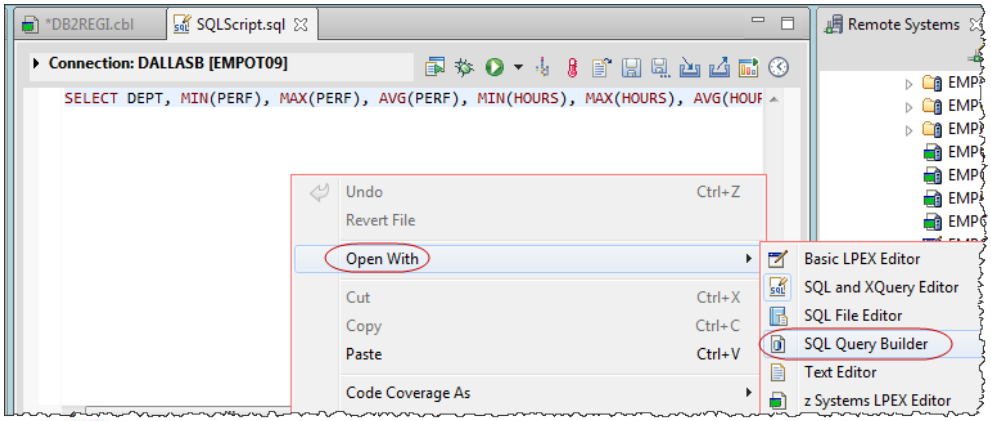


___ 7.5 Reverse engineer the Query

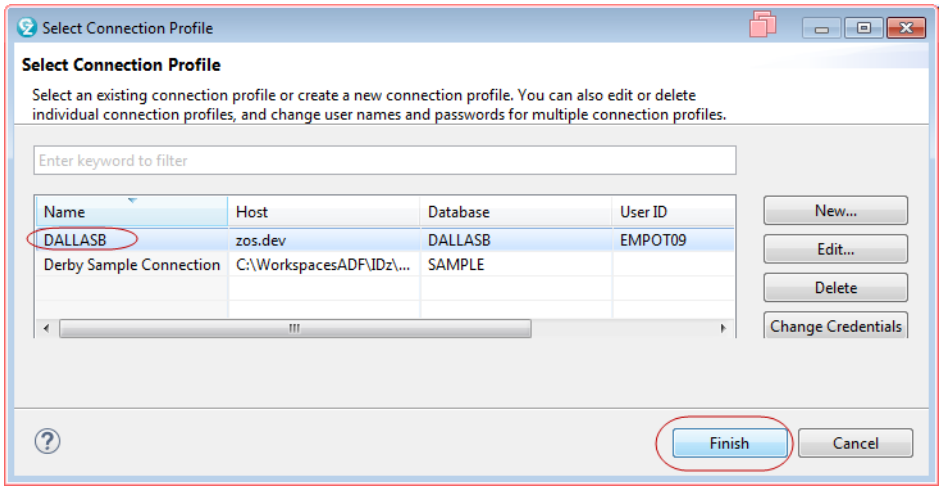
7.5.1 Using the *SQL Outline* view, **right click on the Select** and choose **Show in SQL Editor**



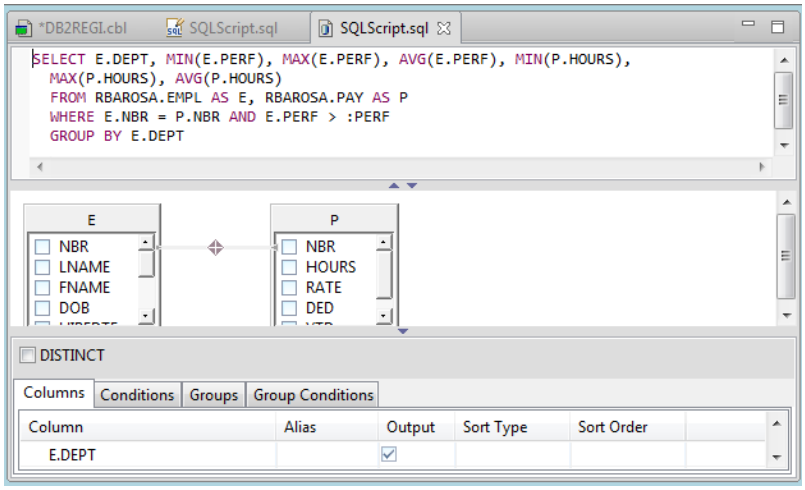
7.5.2 When opened,. select **Open With > SQL Query Builder**



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



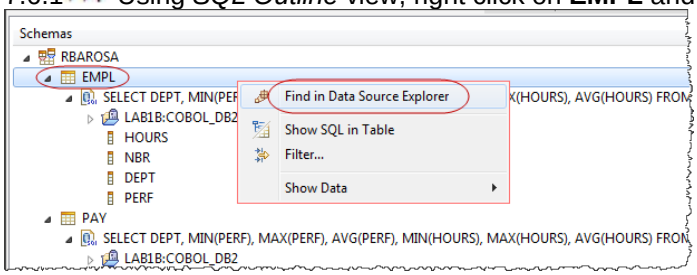
7.5.4 Now we have the graphical editor



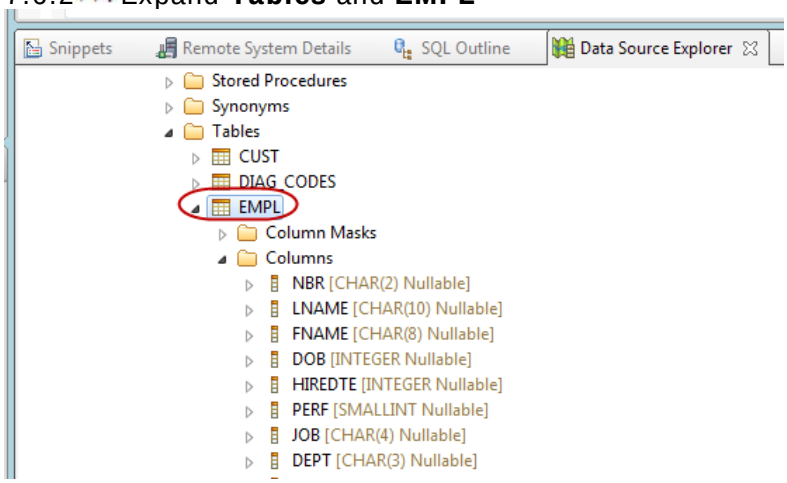
7.5.5 You might use this graphical editor to make changes on SQL statements. Notice that the changes reflects on the COBOL code..

___ 7.6 Displaying DB2 table content

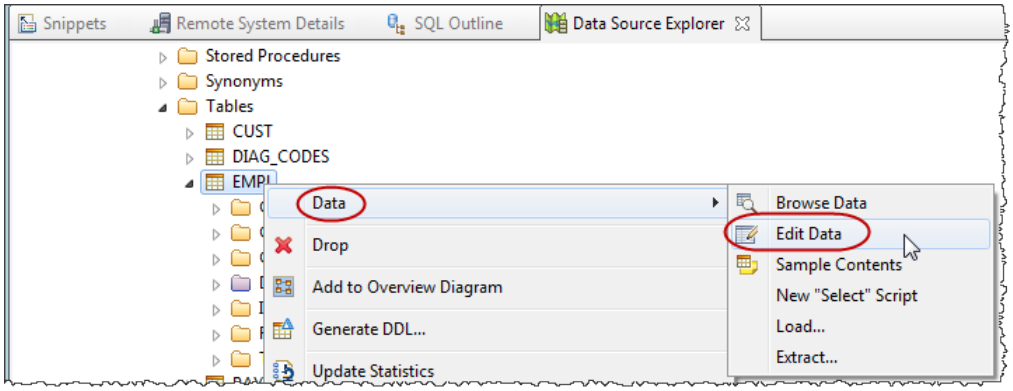
7.6.1 Using *SQL Outline* view, right click on **EMPL** and select **Find Data Source Explorer**



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

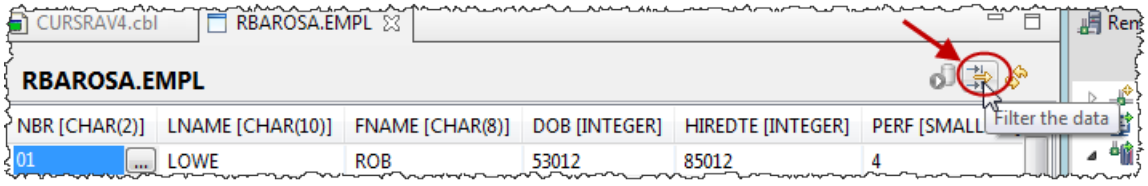


7.6.3 Right click on **EMPL** and select **Data > Edit Data**

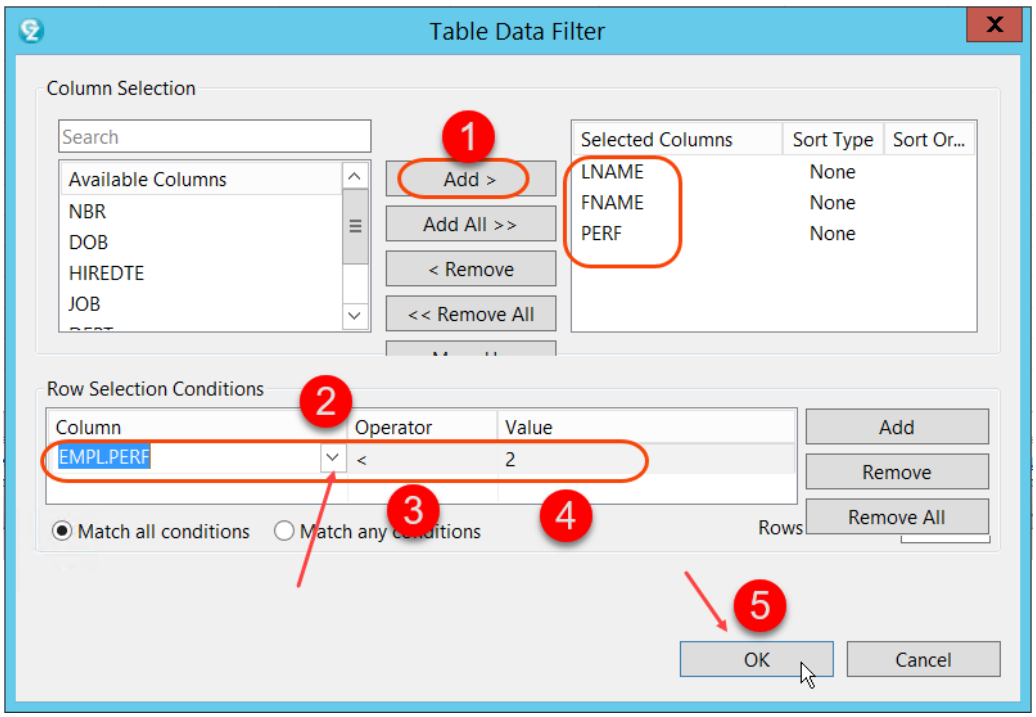


Filtering data results.

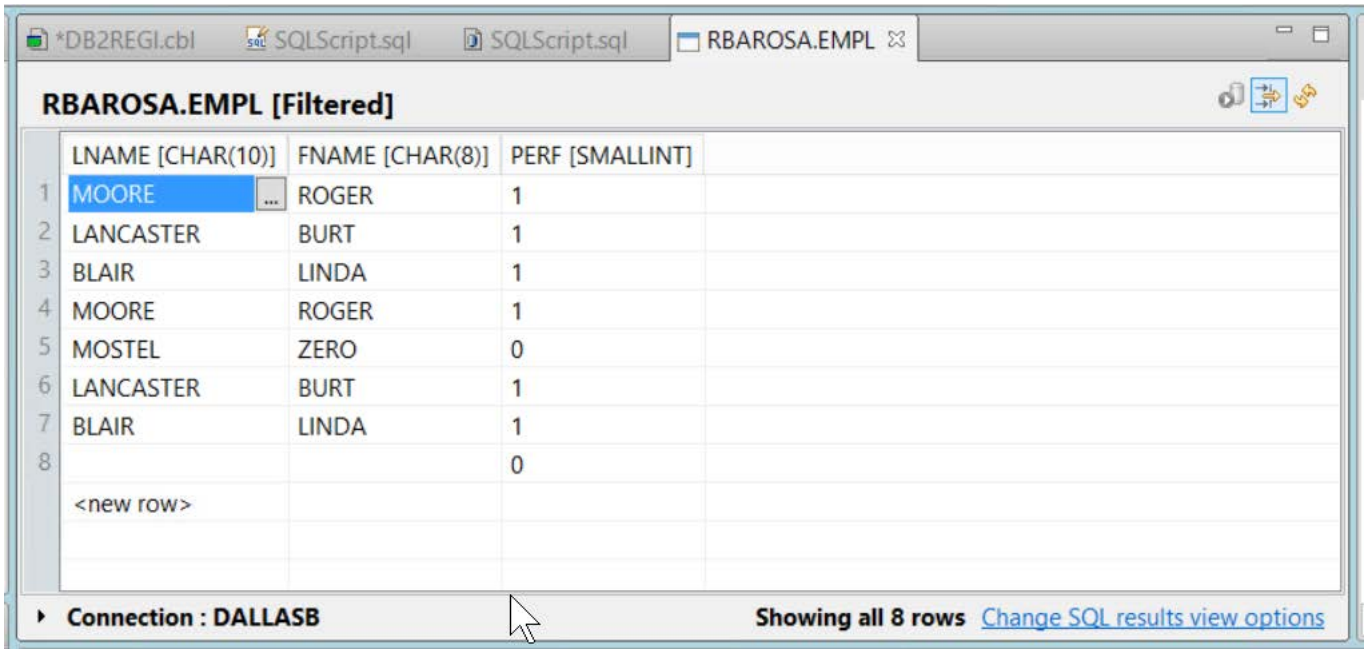
7.6.4 Click **Filter** the data icon



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



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



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

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

Section 8. (Optional) Using File Manager

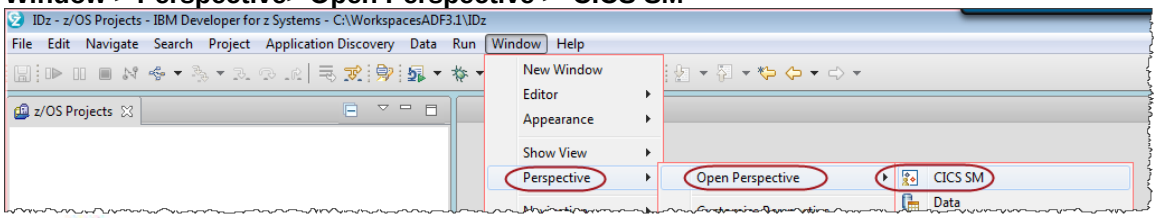
ADFz may optionally have the File Manager (FM) plugin installed. FM works with a broad spectrum of z/OS files and data bases, including VSAM, IAM, and QSAM files, PDS and libraries, DB2 and IMS databases, HFS files, OAM files, CICS queues, MQ queues, and tapes.
File Manager provides powerful formatted editors and viewers, and also provides a full complement of on-line and batch utilities to copy, extract, and load data, to create files and databases, compare and print, and many other utility functions.
File Manager has a conventional 3270 interface that can be accessed from TSO or CICS. here is also an eclipse GUI interface that is available on ADFz.

File Manager has many capabilities we will explore just few examples on this optional lab.

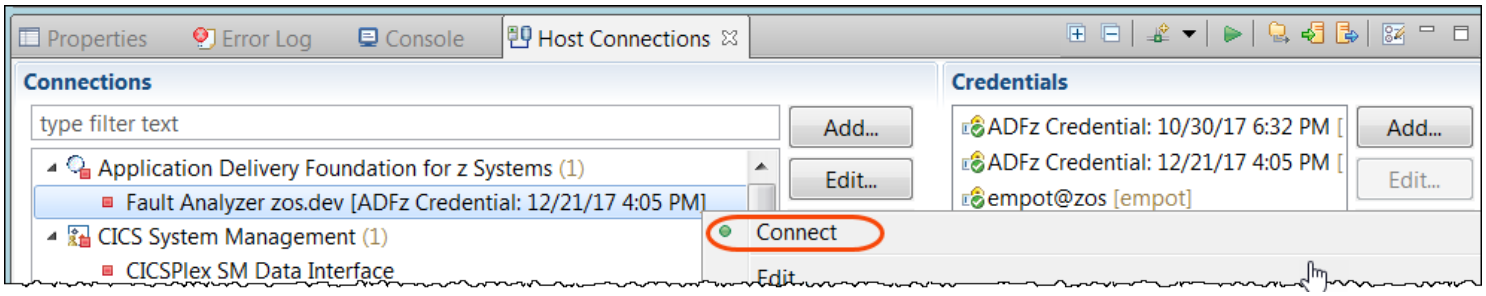
___ 8.1 Verify that you are connected to the PDTOOLS Common components

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

8.1.1 Go to **CICS SM** perspective using
Window > Perspective> Open Perspective > CICS SM



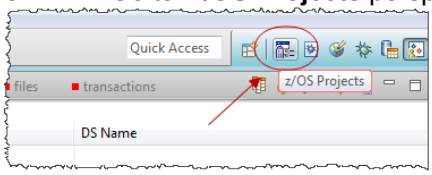
8.1.2 Click on **Host Connections** tab (bottom) and verify that **Application Delivery Foundation for z Systems** has a green icon: .
If it is not green you must connect it. **Right click and select Connect**



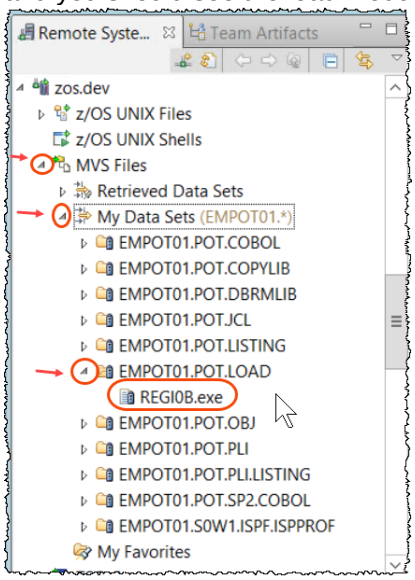
___ 8.2 Using View Load Module utility

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

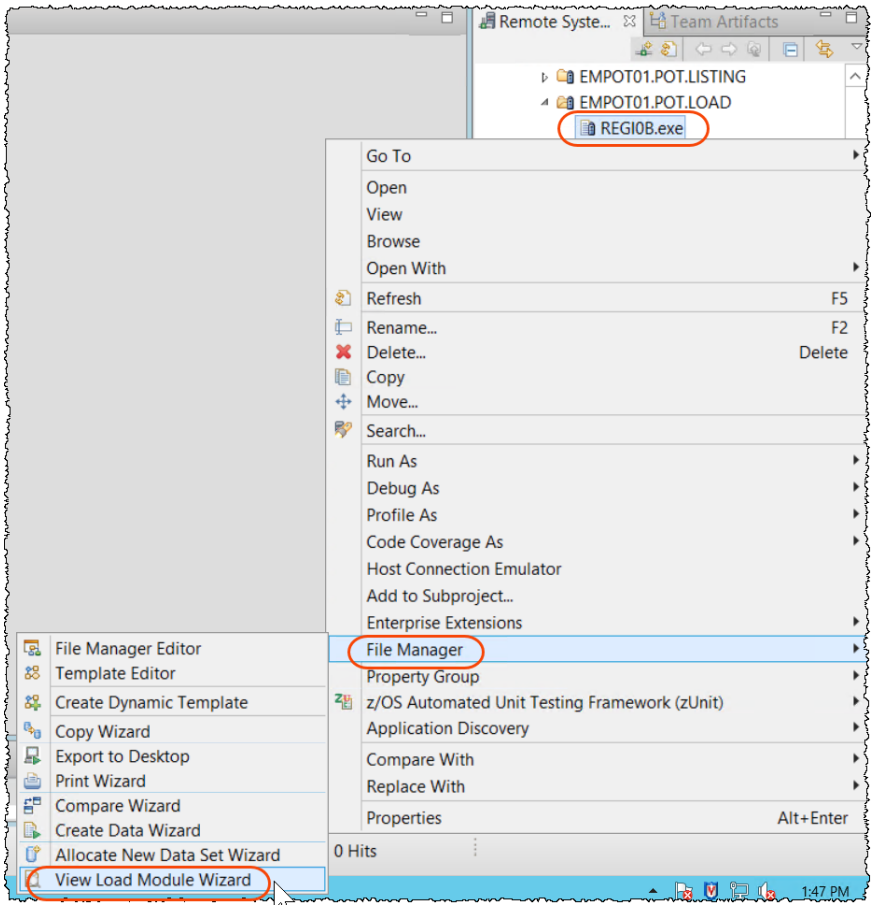
8.2.1 Go to **z/OS Projects** perspective clicking on the upper top right icon as below



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

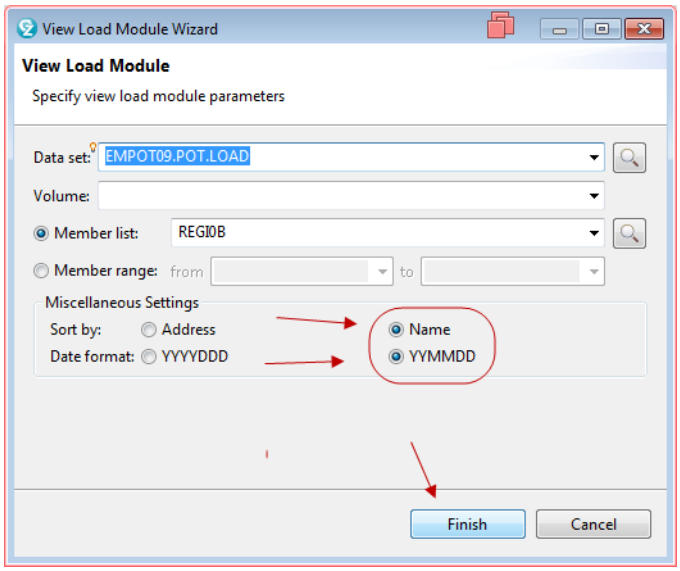


8.2.3 Right click on **REGI0B.exe** and select **File Manager** and **View Load Module Wizard**.



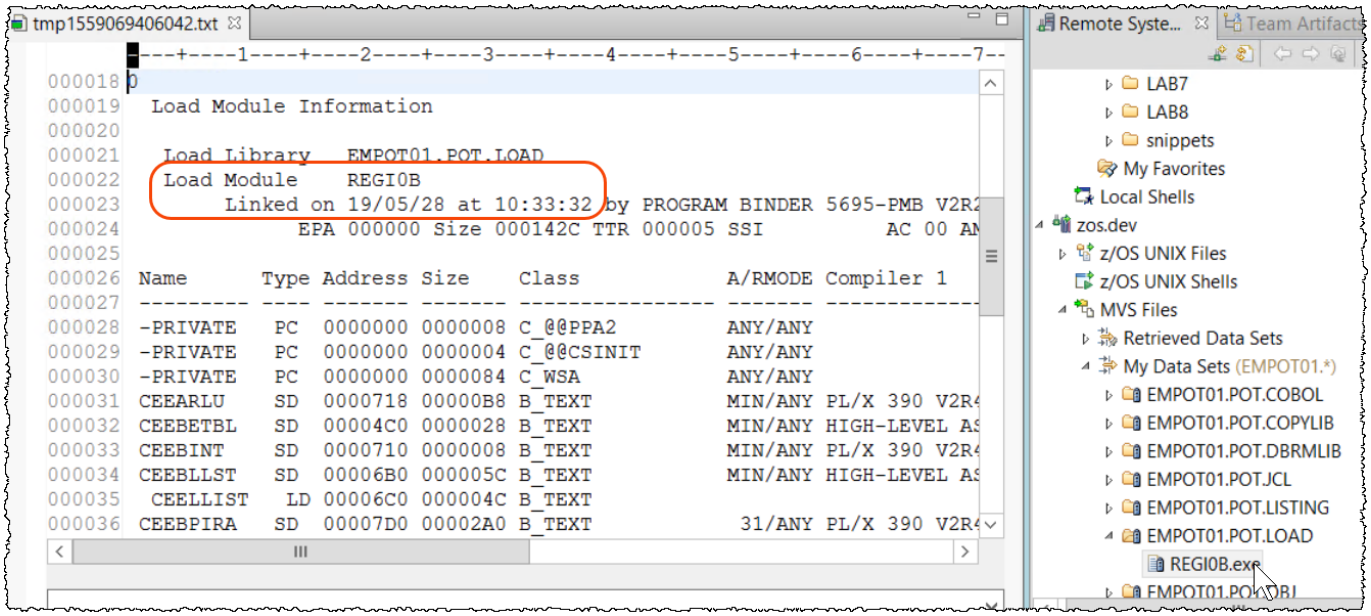
If you get a *Sign on* dialog for *Fault Analyzer* use **empot01** credentials and click **OK**

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



8.2.5 The result is below..

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



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

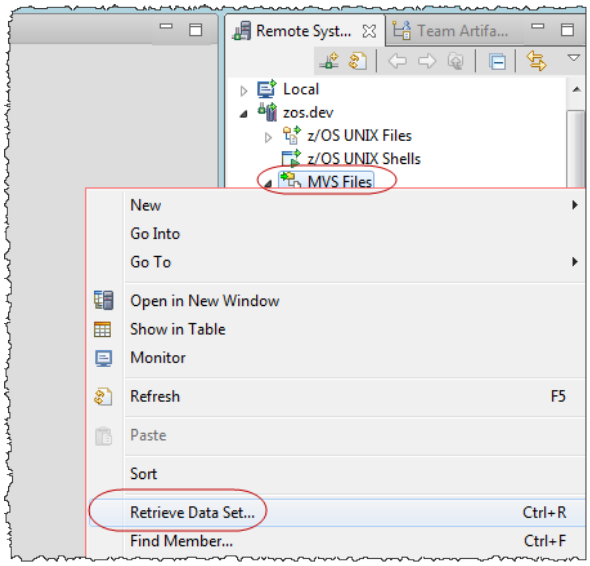
8.2.6 ⏭ Close the edited file. Can use **CTRL + Shift + F4** to close all opened editors. Or just click on the ⊠. of each opened editor

___ 8.3 Working with z/OS data sets

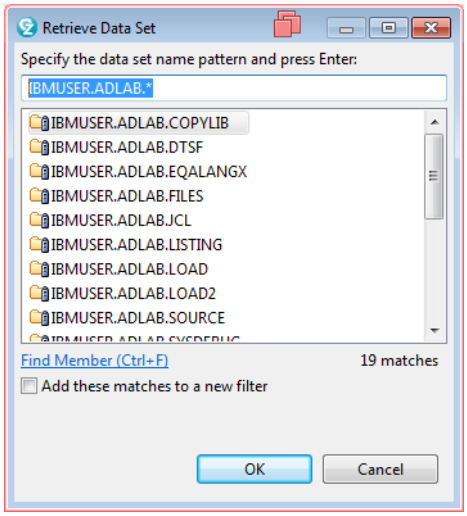
In this section, you will open a variable blocked sequential dataset in the editor or viewer, using a copybook as a layout.
You will use the very basic features of the editor, including navigating a file, we will not cover finding data in a file, changing data, inserting, deleting and sorting records.

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

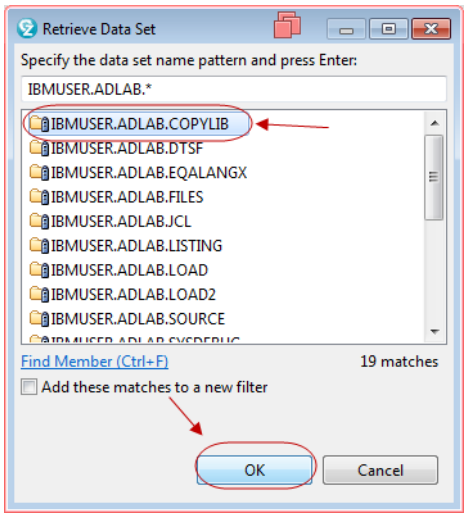
⏭ Using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set** ….
This is equivalent at the TSO 3.4 function.



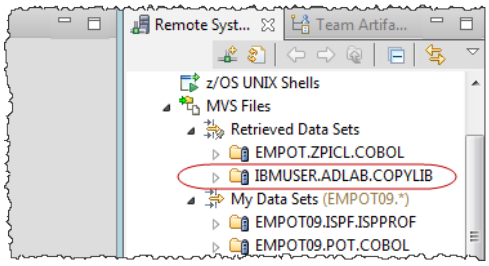
8.3.2 ⏭ Type **IBMUSER.ADLAB.*** and press **Enter.**



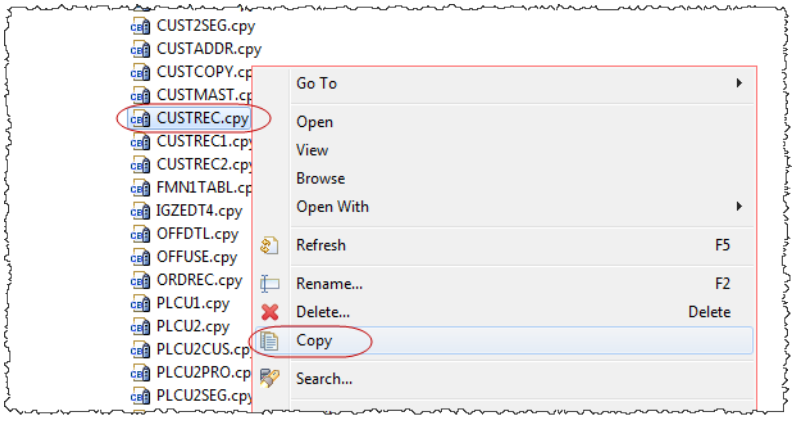
8.3.3 Select **IBMUSER.ADLAB.COPYLIB** and click **OK.**



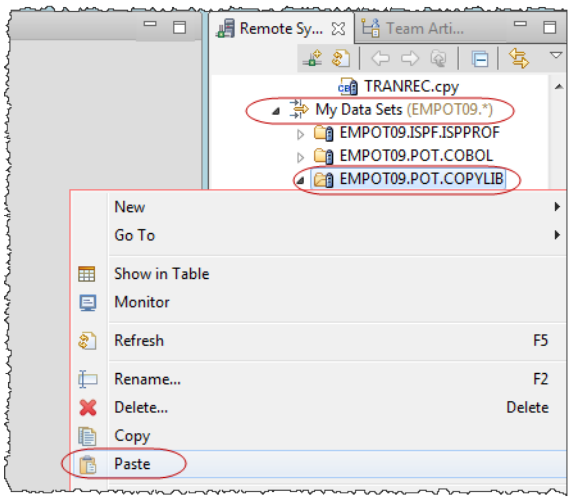
8.3.4 The data set will be under *Retrieved Data Sets*.



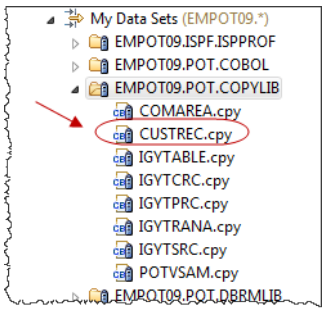
8.3.5 Expand **IBMUSER.ADLAB.COPYLIB**, right click on **CUSTREC.cpy** and select **Copy**. You want to copy this member to your PDS.



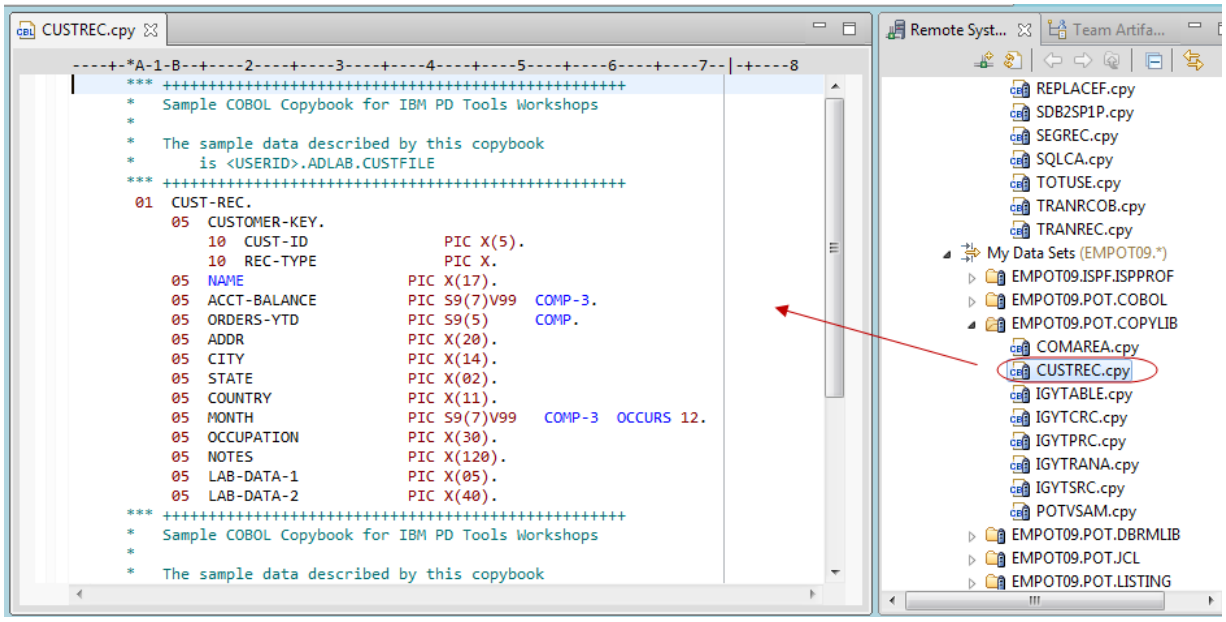
8.3.6 Navigate to **My Data Sets**, right click on **EMPOT01.POT.COPYLIB** and select **Paste**



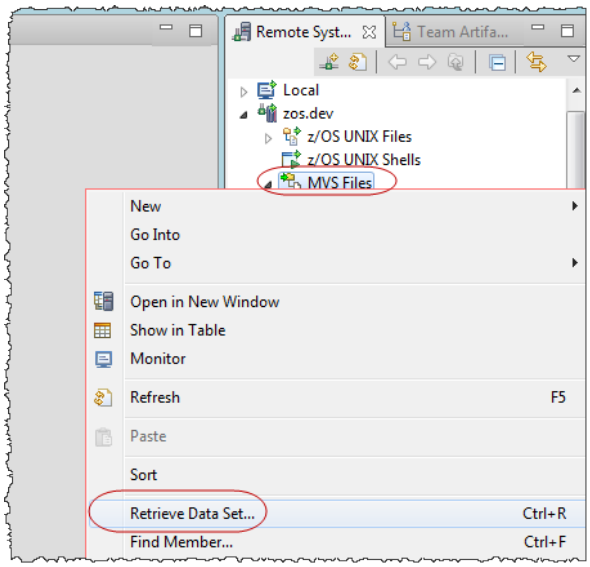
8.3.7 You should have:



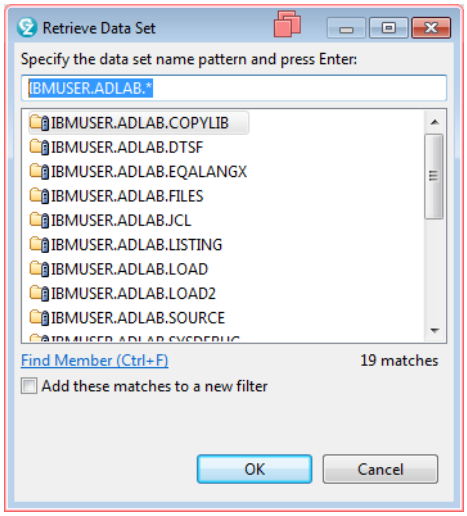
8.3.8 Double click on **CUSTREC.cpy** to edit it.
This copybook is the file layout of the file that you will see later. Note than some fields are COMP-3.



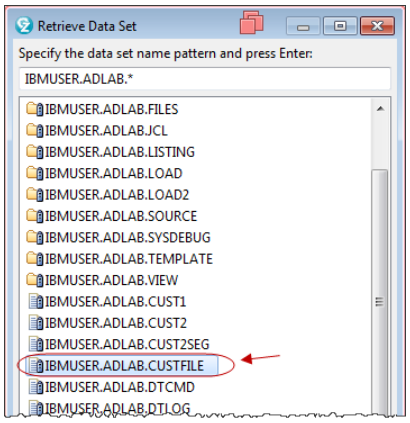
8.3.9 To see the data set, using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set** ….
This is equivalent at the TSO 3.4 function.



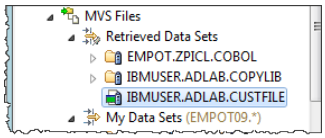
8.3.10 Type **IBMUSER.ADLAB.*** and press **Enter.**



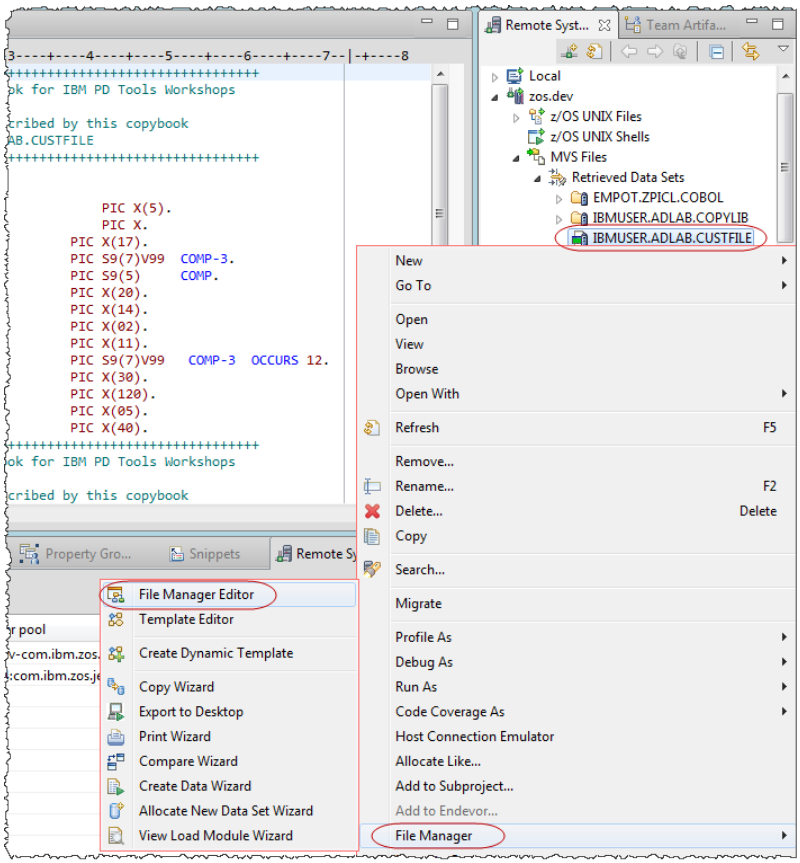
8.3.11 Scroll down, select **IBMUSER.ADLAB.CUSTFILE** and click **OK.**



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

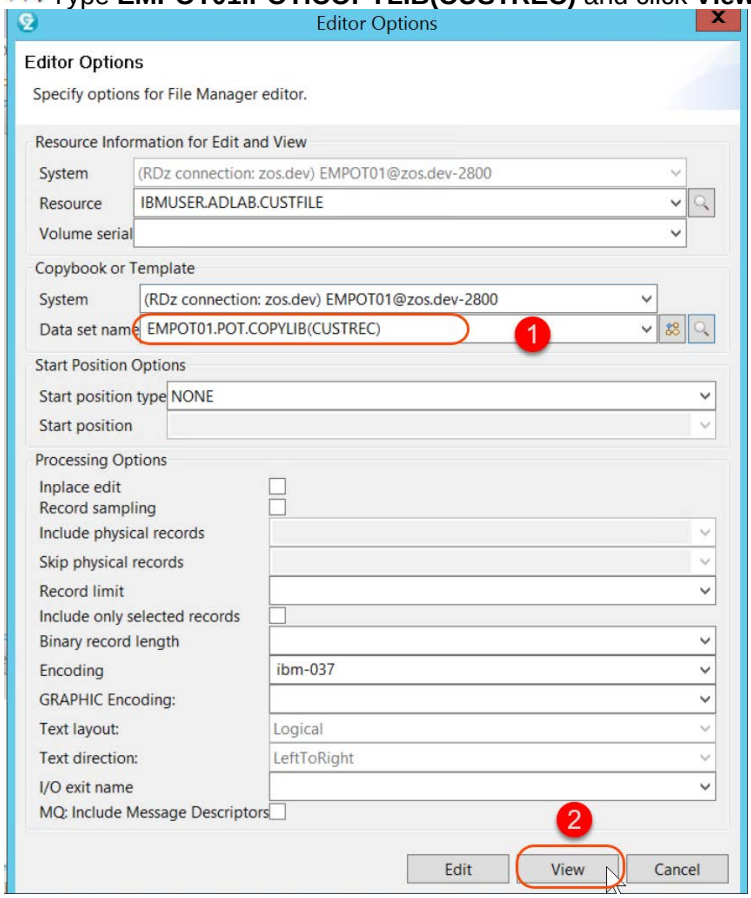


8.3.13 Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**. You want to map this file against the COBOL COPYBOOK that you copied.

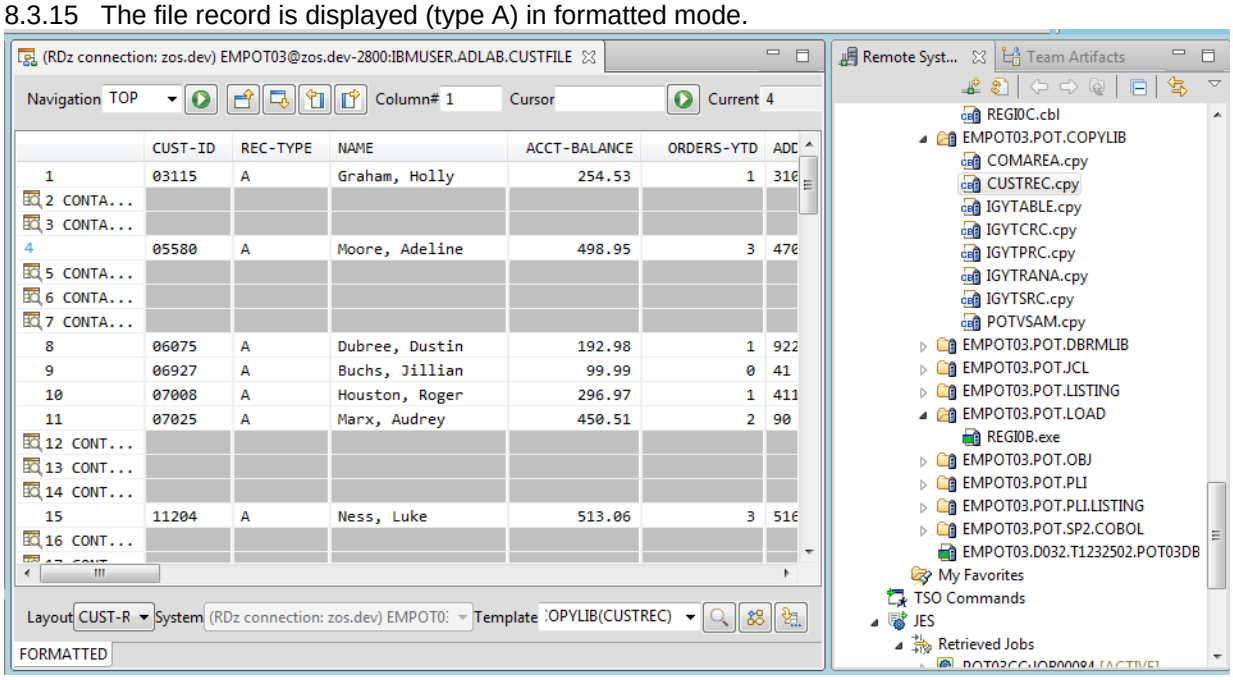


8.3.14 On *Data set name* you will use the PDS and member that you have copied the COPYBOOK.

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



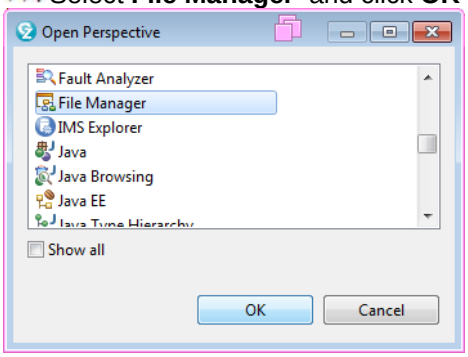
8.3.15 The file record is displayed (type A) in formatted mode.



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

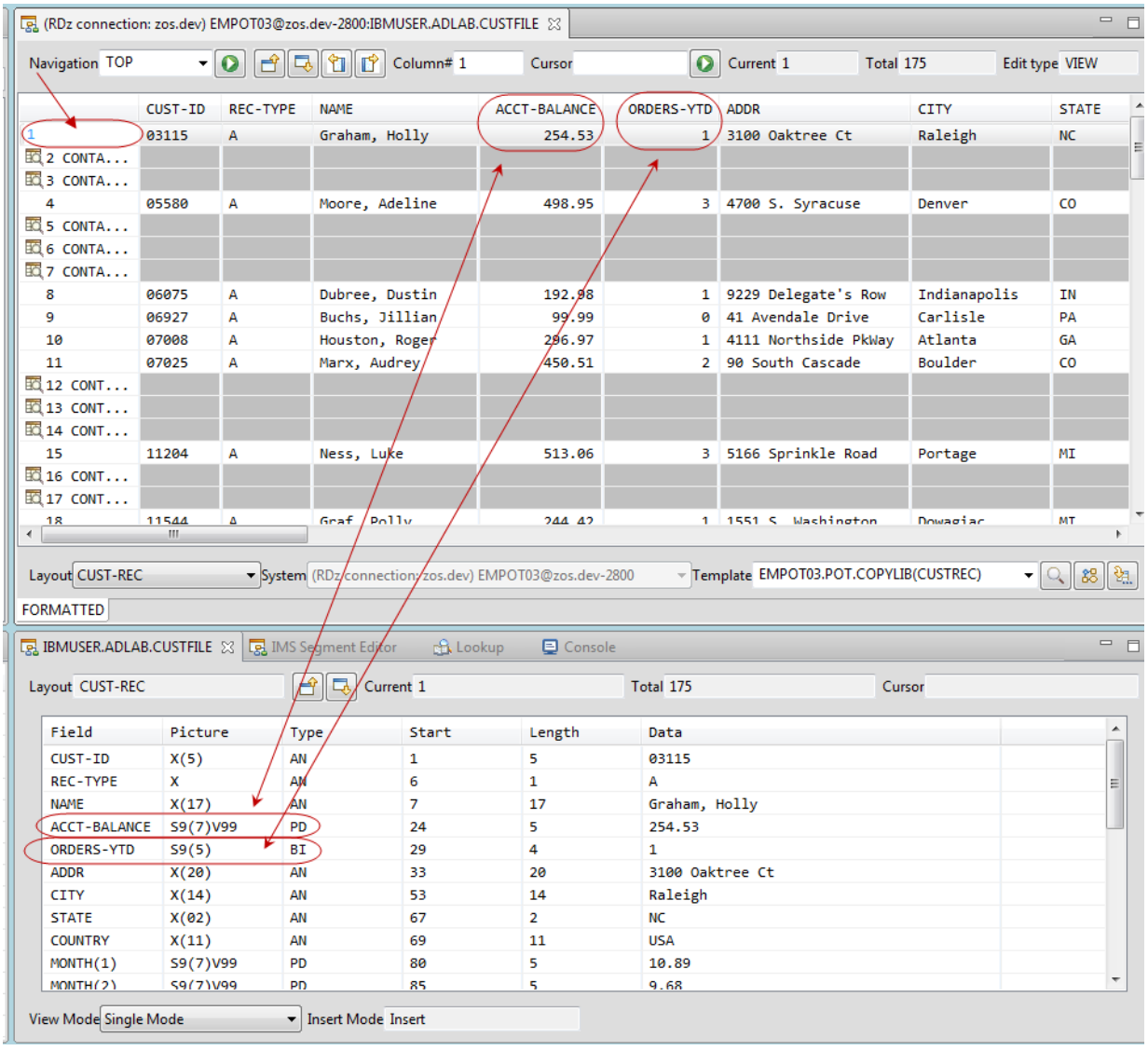
⏭ Select **Window > Perspective > Open Perspective > Other…**

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



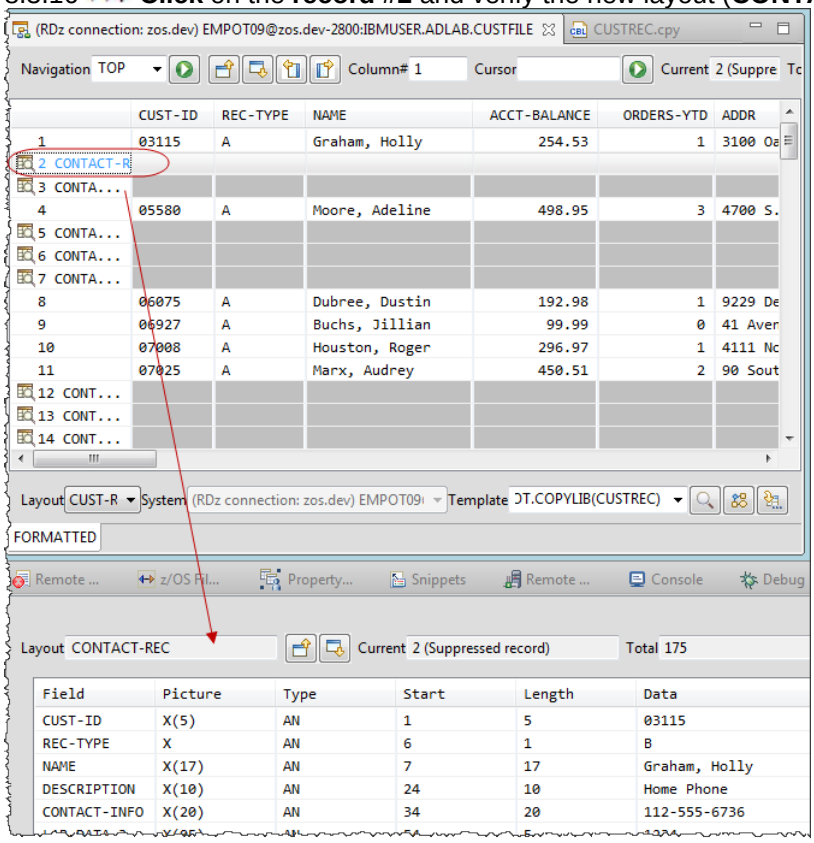
⏭ **Click** on the **record #1**.

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

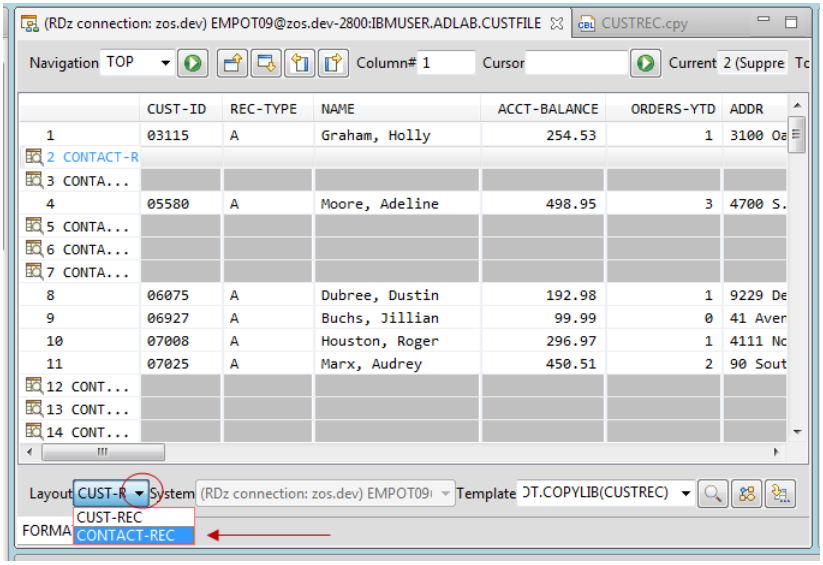


You may have to scroll down this view to see the formatted field names.
Also notice that there are two views of the data. The multiple record view appears at the top by default, and the single record view appears at the bottom and displays only one record at a time.
If you select (click) a record in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

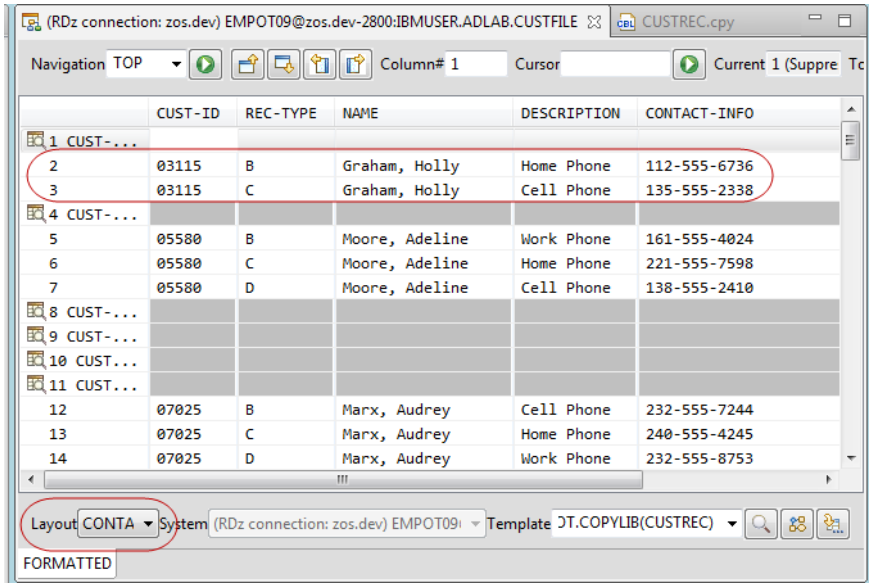
8.3.16 **Click** on the **record #2** and verify the new layout (**CONTACT-REC**)..



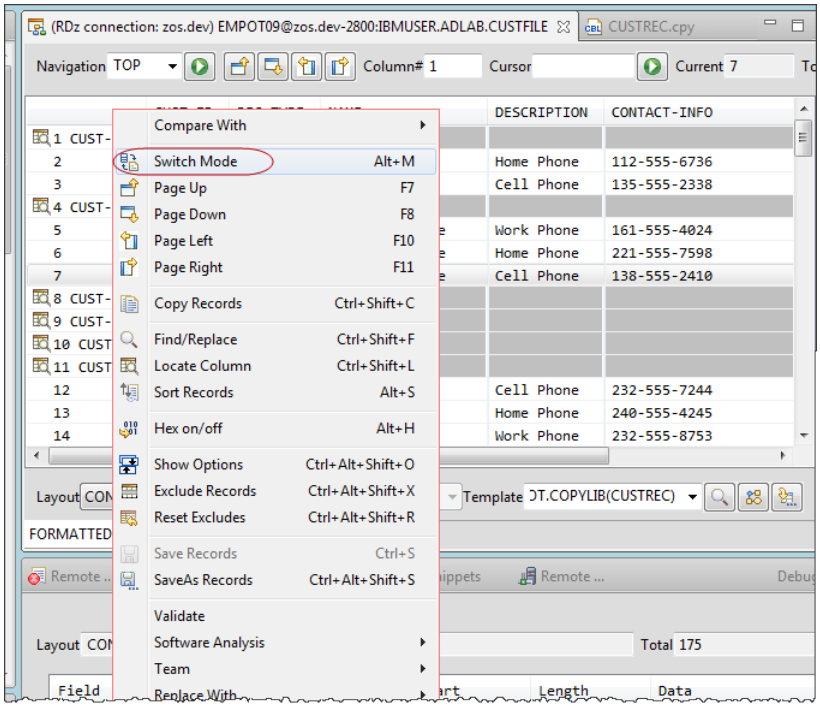
8.3.17 Click on the *Layout* and select **CONTACT-REC**. As you see, you can work with multiple layouts.



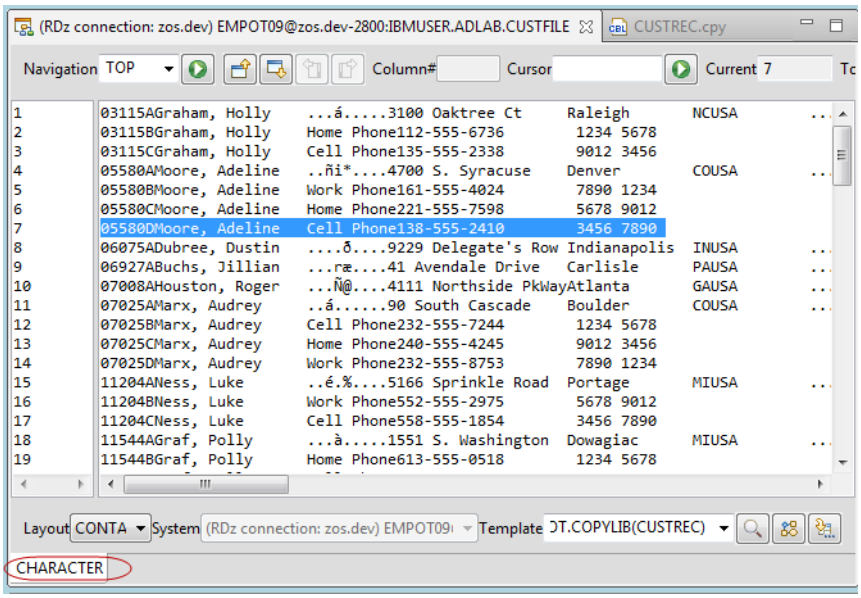
8.3.18 You now can see that records 2 and 3 are displayed (type B and C) in formatted mode.



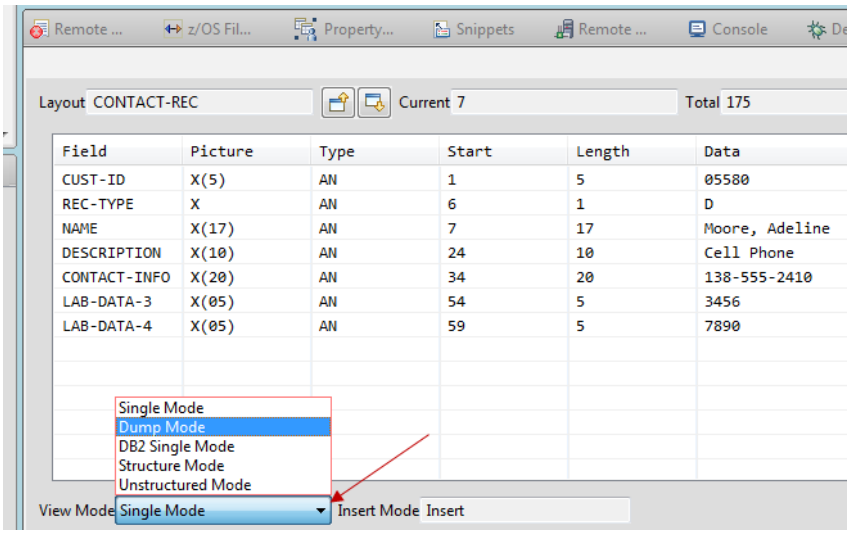
8.3.19 ⏭ Right click on the editor and verify that there are many options.
⏭ Click on **Switch Mode**

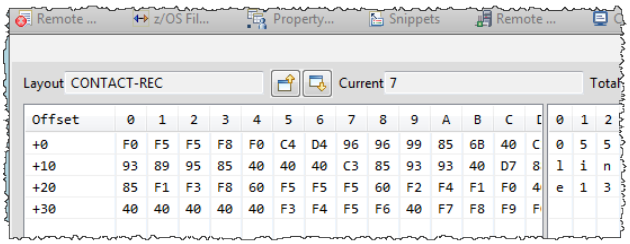


8.3.20 The multiple-record view display is changed to character mode. In character mode, the data is not formatted according to the fields in layout.



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





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

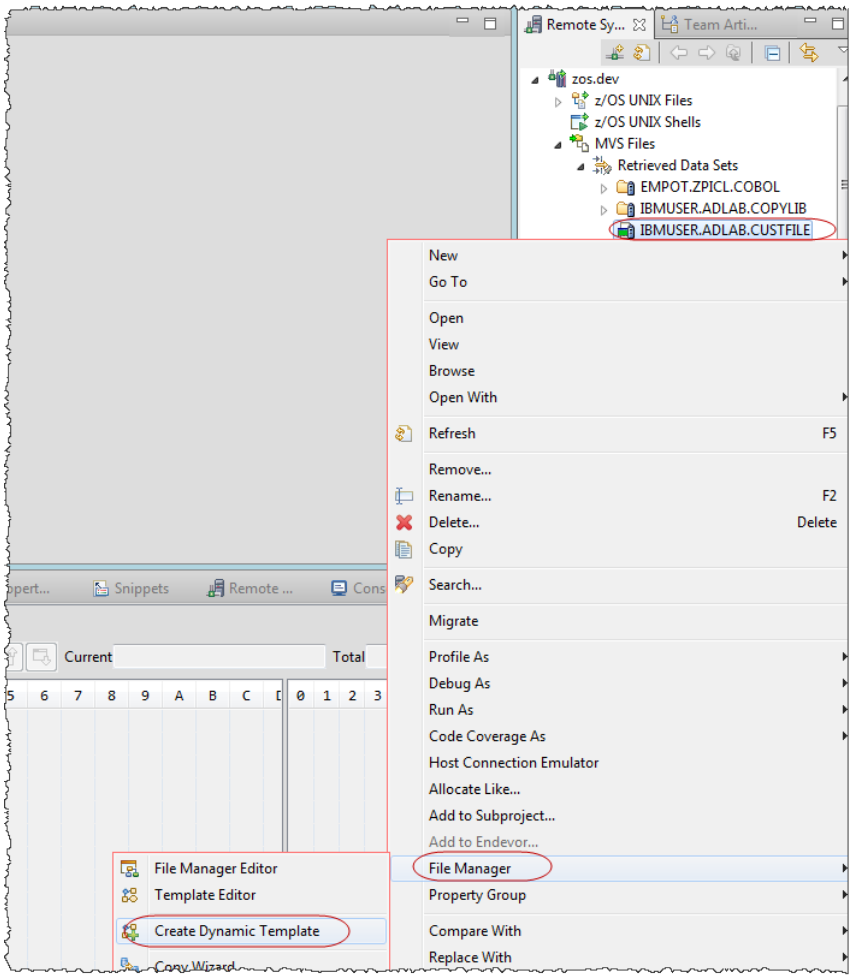
8.3.22 ⏭ Use **Ctrl + Shift + F4** to close all opened editors.

___ 8.4 Working with templates

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

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

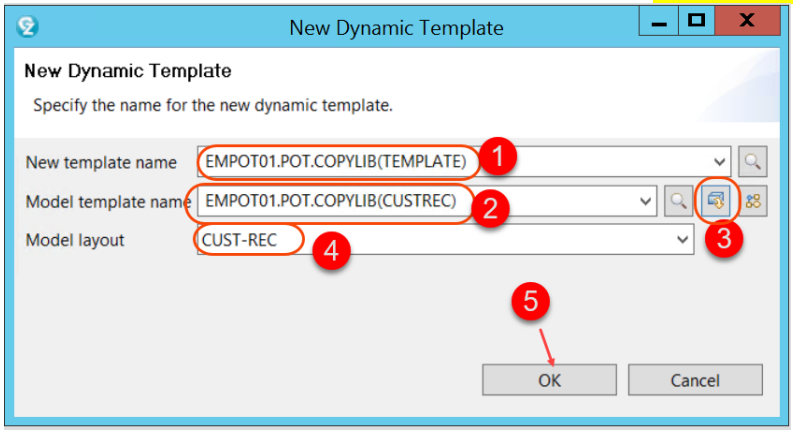
8.4.1 Go back to **z/OS Projects** Perspective and using *Remote Systems Explorer* view, right click **IBMUSER.ADLAB.CUSTFILE** file and select **File Manager -> Create Dynamic Template**



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**).

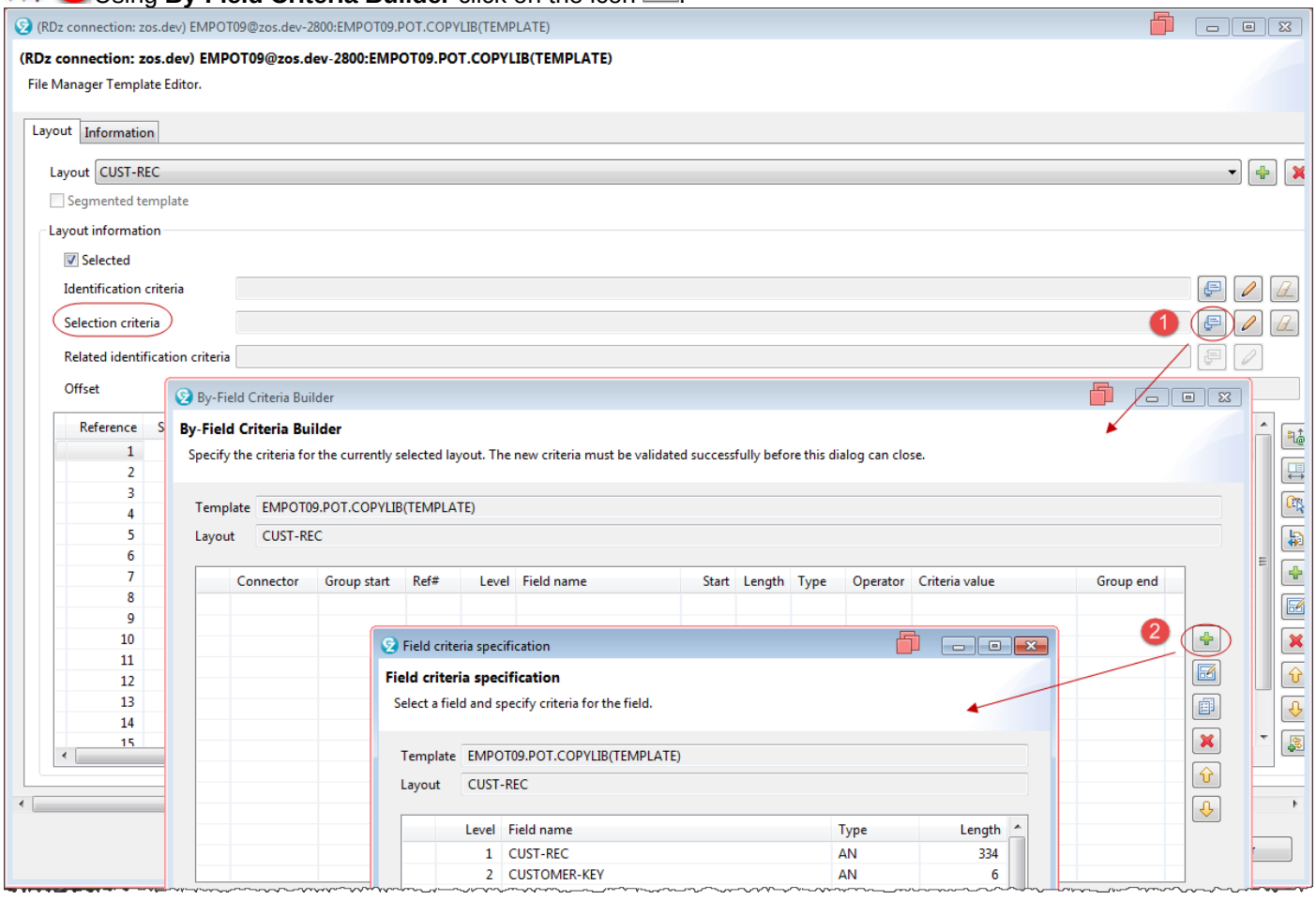
⏭ Be sure that **CUST-REC** is selected and click **OK.** If an overwrite message appears, click **YES.**



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**.

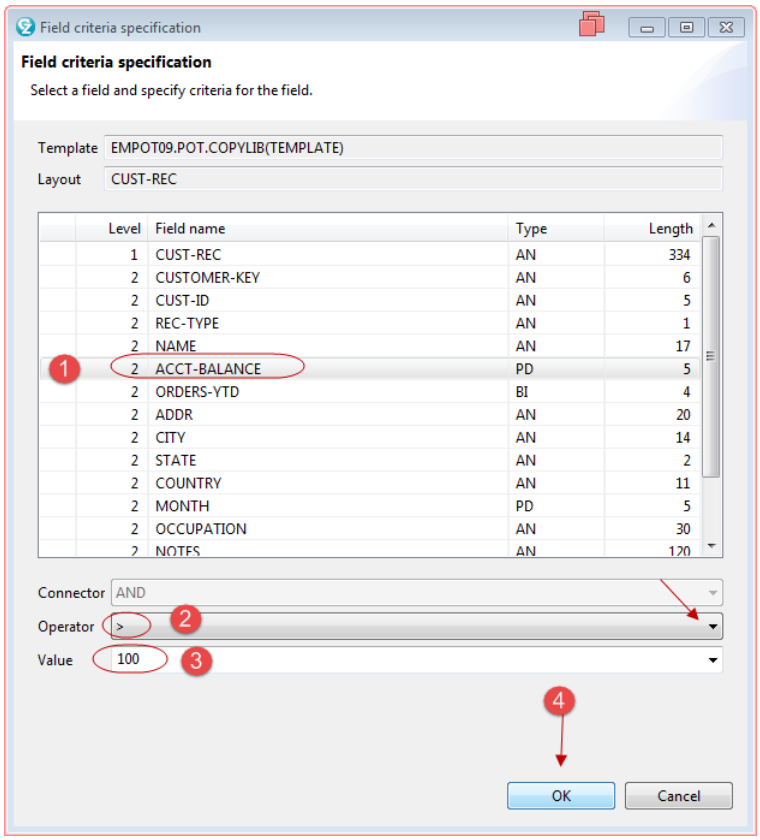
⏭ 1 On ***Selection criteria*** click on the icon .

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



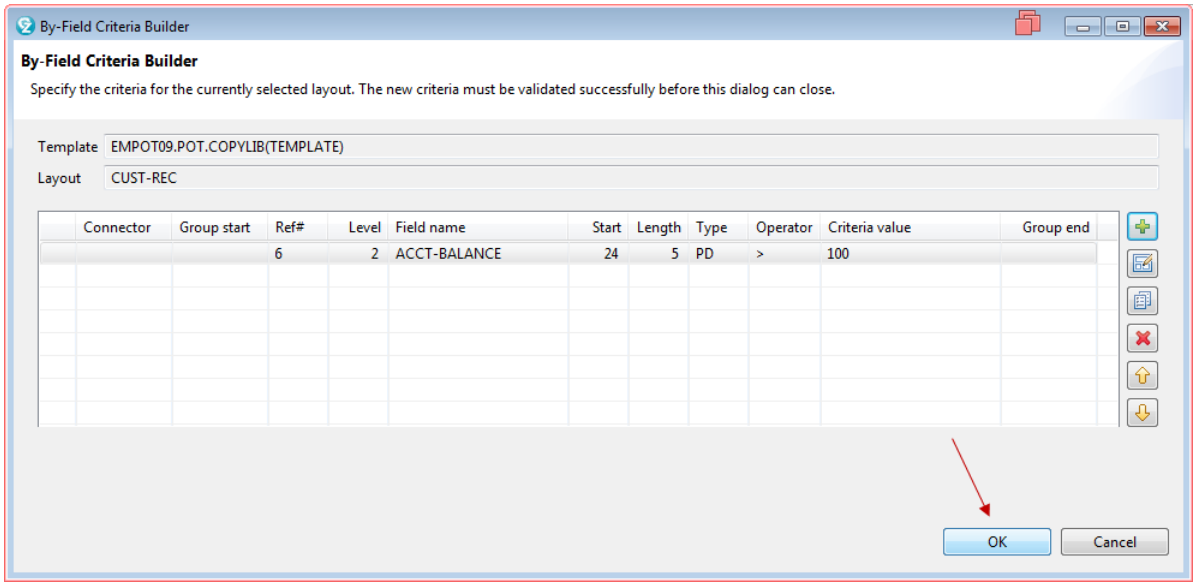
8.4.4 ⏭ 1 On the *Field criteria specification* select **ACCT-BALANCE**

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

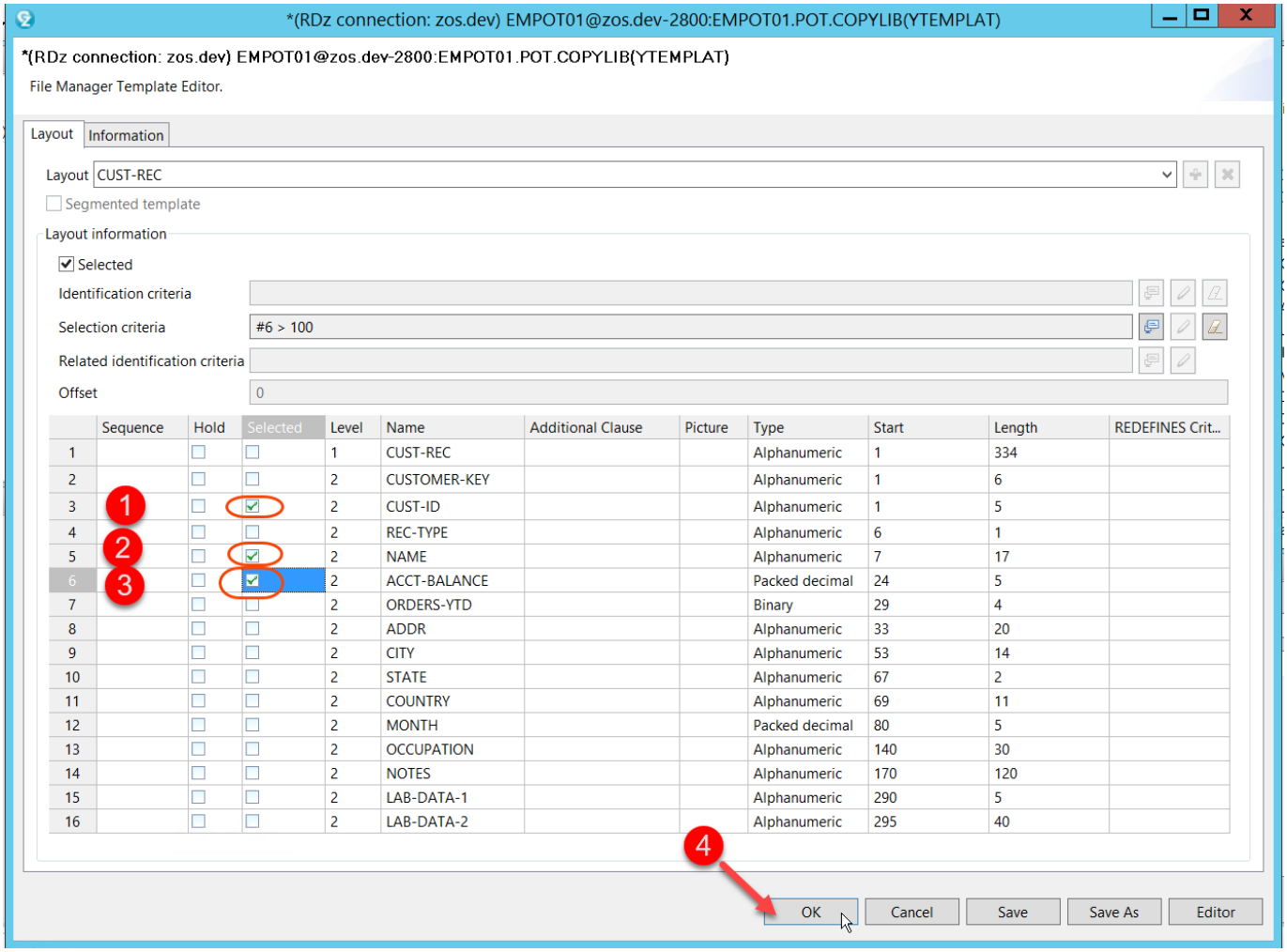


8.4.5 The result is shown below..

⏭ Click **OK**

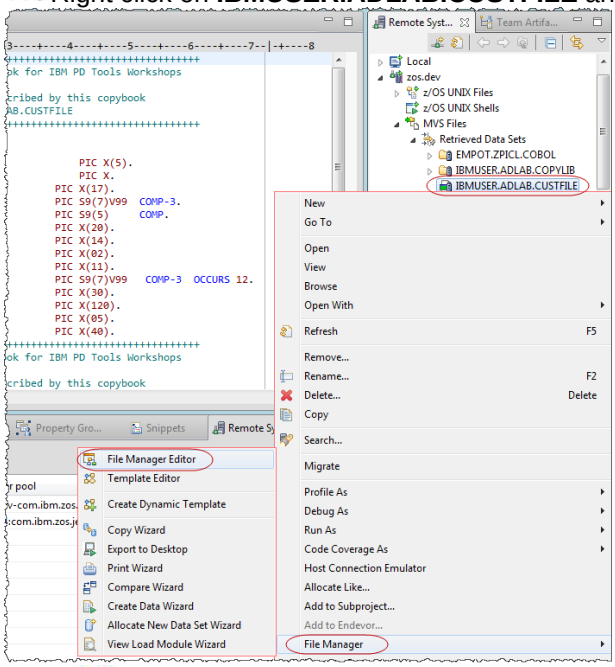


8.4.6 Using the column **Selected** double-click on the boxes
1 **CUST-ID,** 2 **NAME** and 3 **ACCT-BALANCE**.
Click **OK.**
The template is created. At any time, you can edit and modify it.

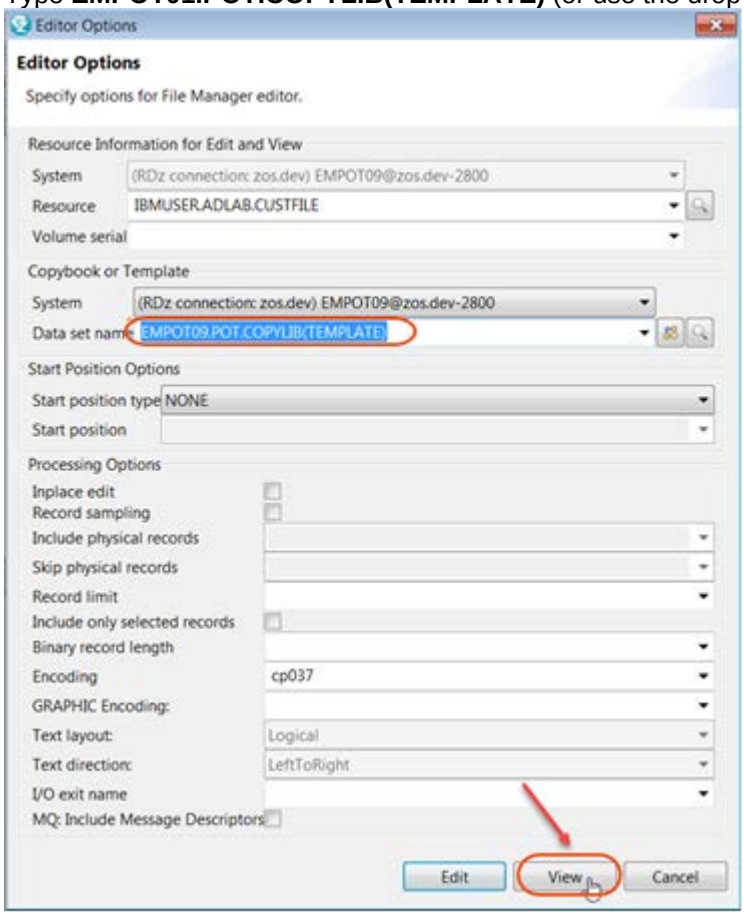


8.4.7 Now that the template is created you can use against the data set.

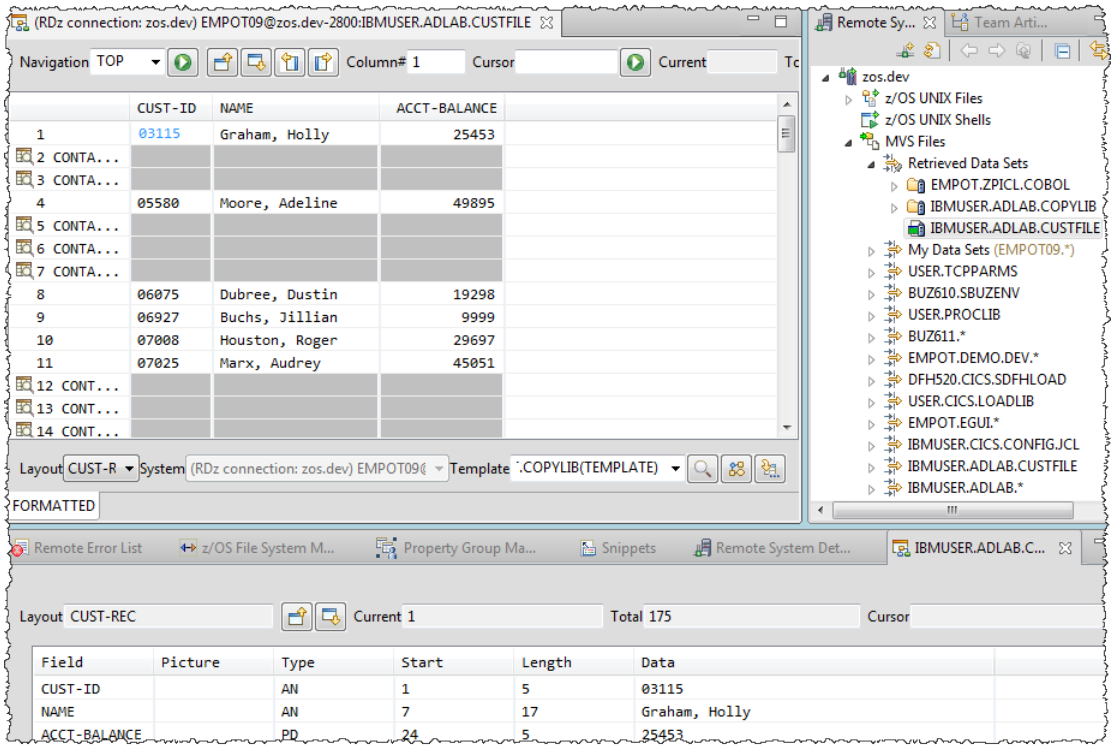
⏭ Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**.



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.**



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. .

LAB 2C -(OPTIONAL) Application Discovery : Find a candidate API from Catalog Manager Application (EGUI)

Updated August 12 2019 by Regi, Reviewed by ???

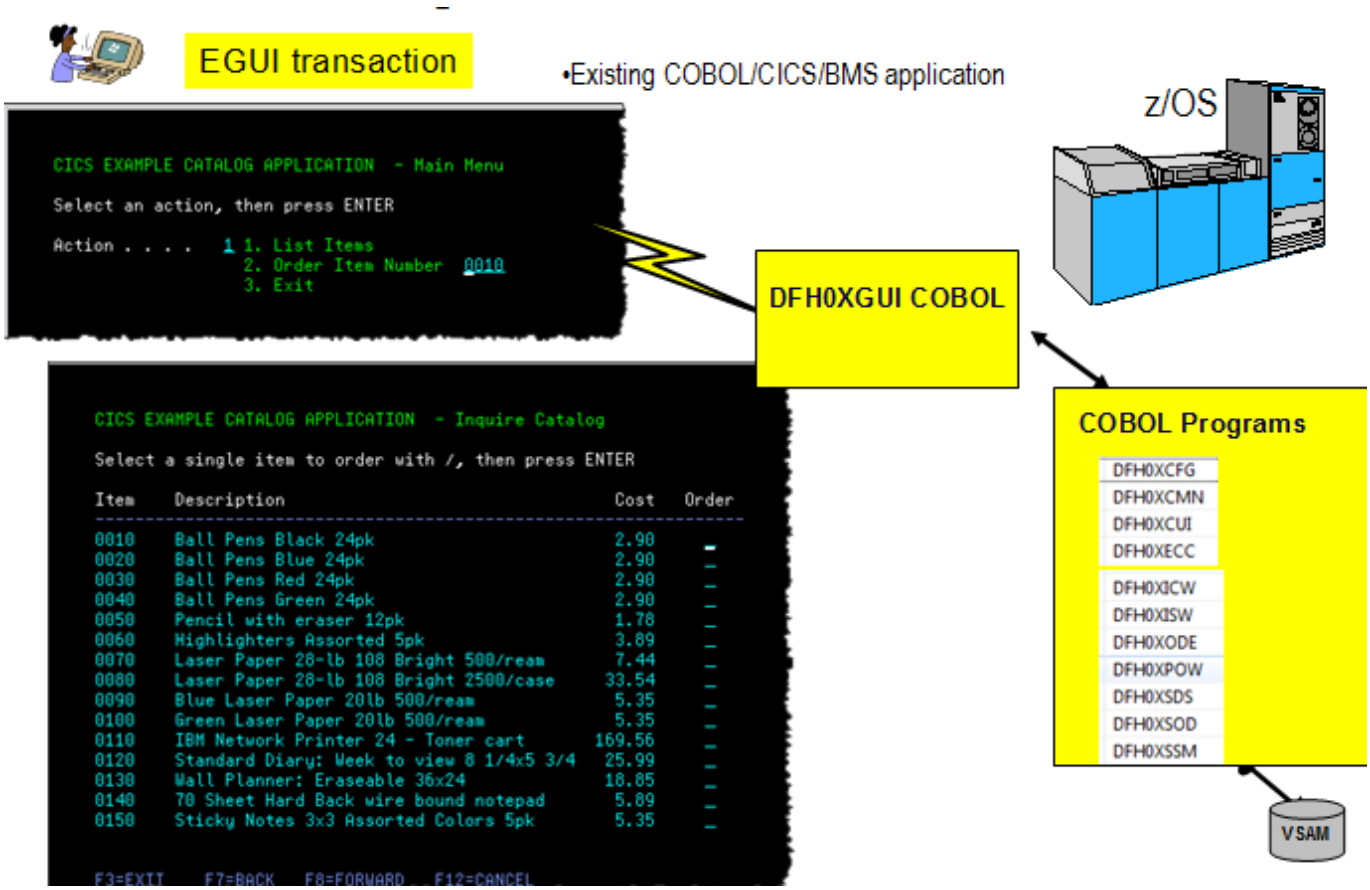
Acknowledgments:
We would like to thank the original authors of this exercises:
Aymeric Affouard, Eric Phan, Paul Pilotto & Nigel Williams.

Overview of development tasks

This lab you will go through the process of discovering a possible API from existing the *Catalog Manager Application* (EGUI transaction).
Catalog Manager application can be found here:
https://www.ibm.com/support/knowledgecenter/SSGMCP_5.3.0/com.ibm.cics.ts.exampleapplication.doc/topics/dfhxa_t100.html

Later another LAB (LAB 2D) will demonstrate how to create an API that allows REST clients to access this application.

Below the existing EGUI transaction:



The main tasks that you will perform are:

1. **Get familiar** with the programs and run them using the 3270 terminal.
2. Discover the *Catalog Manager application* using **IBM Application Discovery**
.

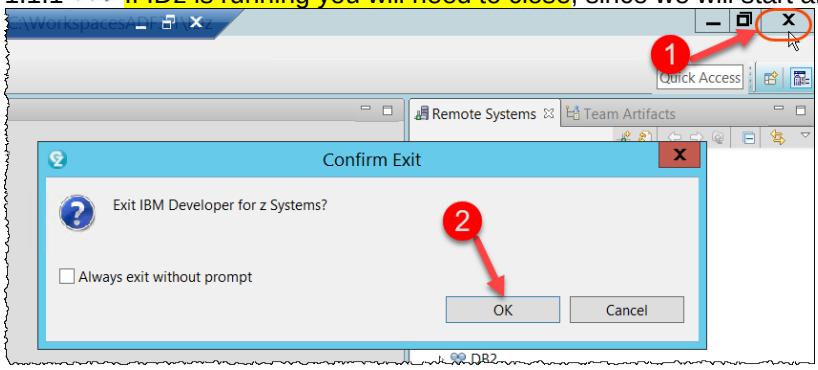
Each time you see a symbol ⏭ it means that you have to "do" something on your computer – not merely read the document.

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

You will start a 3270 emulation and execute a transaction named EGUI to become familiar with the Application that you intend to apply a transformation.

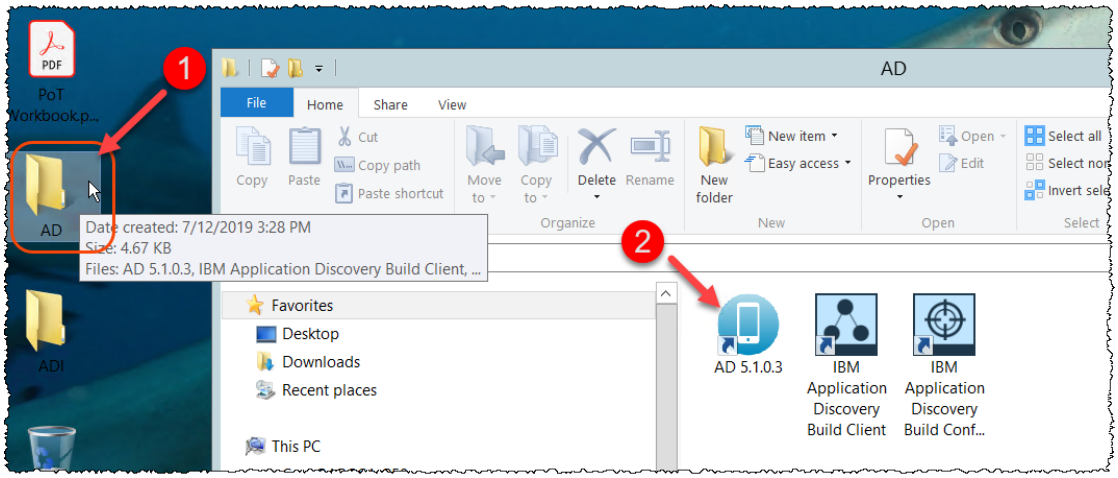
___ 1.1 Emulating a CICS 3270 terminal and running the EGUI transaction

1.1.1 ⏭ If IDz is running you will need to close, since we will start another IDz instance for this lab.

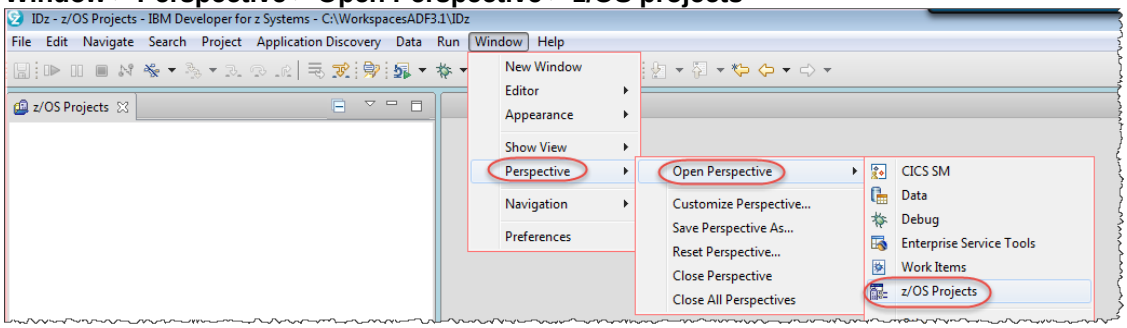


1.1.2 Start *IBM Developer for z Systems* .

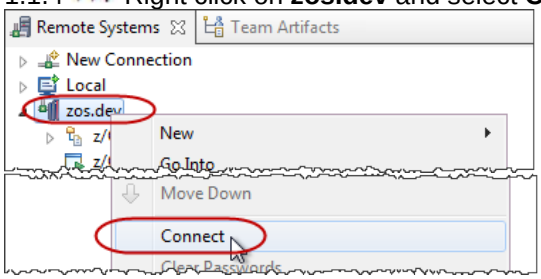
⏭ Look for a folder on the windows desktop with the word "**AD**"  , double click on it and double click on the icon **AD 5.1.0.3**  :



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



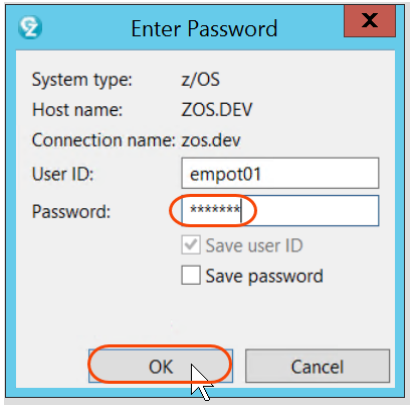
1.1.4 ⏭ Right click on **zos.dev** and select **Connect**



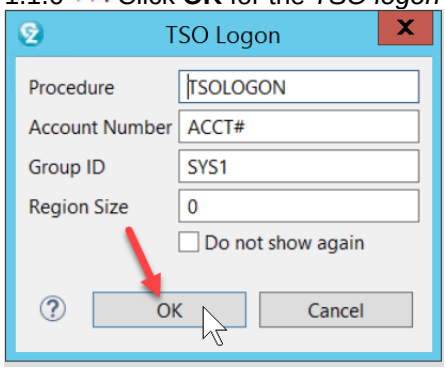
1.1.5 You will be are prompted for your z/OS userid and password.

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

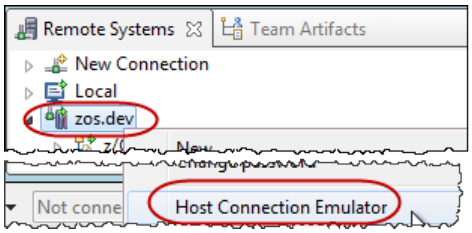
⏭ Click **OK** to connect to z/OS.



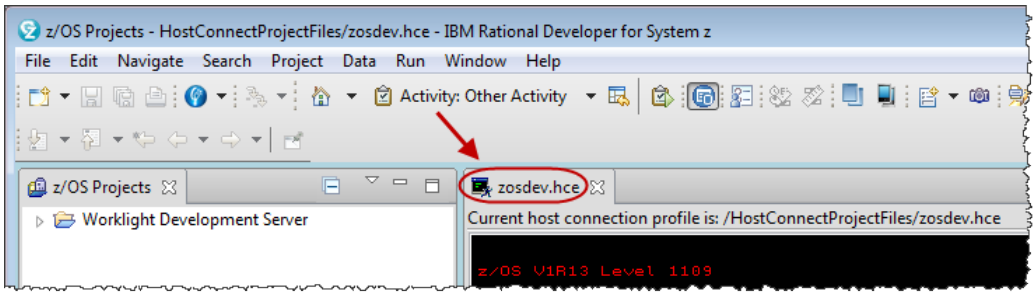
1.1.6 ⏭ Click **OK** for the *TSO logon* dialog



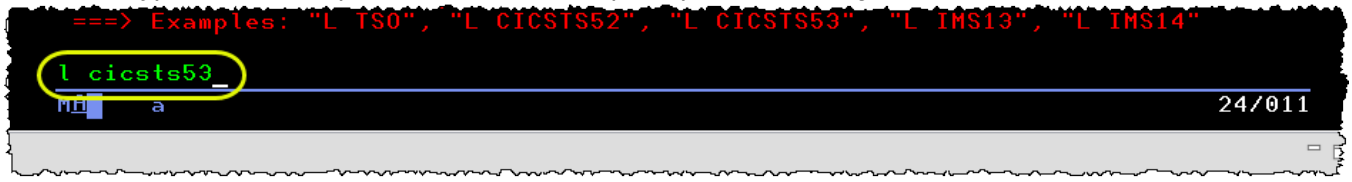
1.1.7 Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



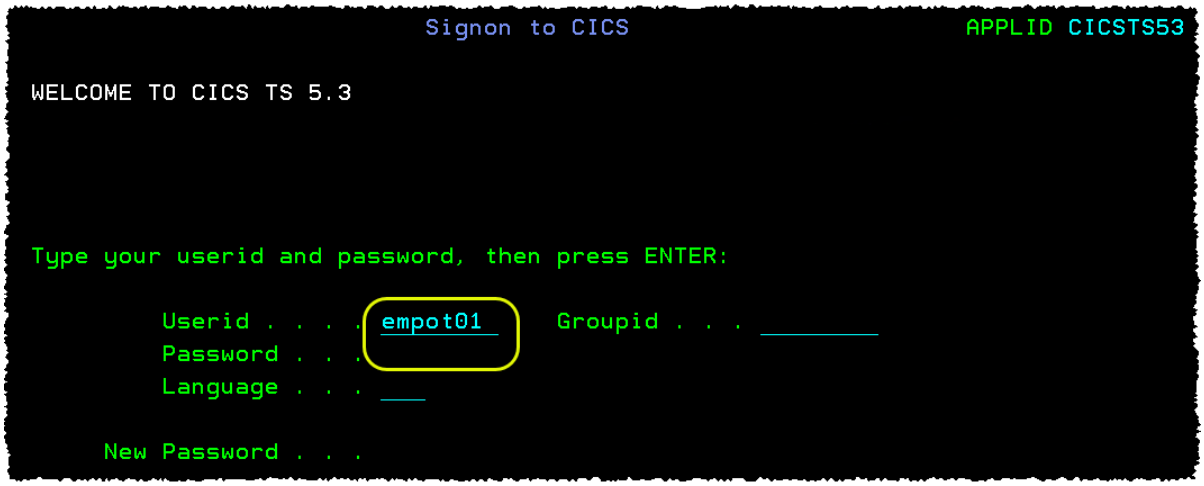
1.1.8 Since you will need more space, **double-click** on the **zos.dev.hce** title



1.1.9 Type **l cicsts53**. (where l is L lower case) and press **Enter key**.



1.1.10 Logon using your z/OS user id empot01 and password and press **Enter**.



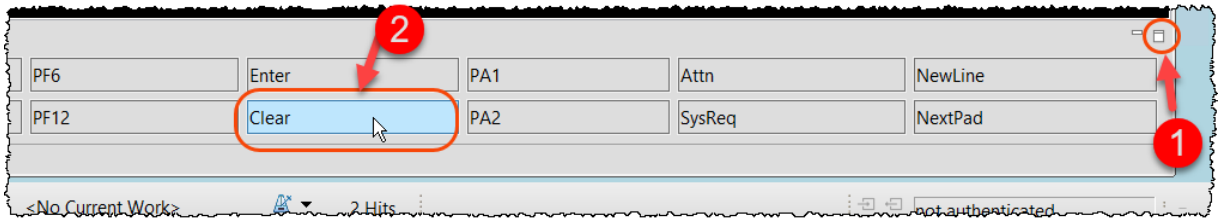
1.1.11 The sign-on message is displayed

___ 1.2 Run CICS transaction EGUI

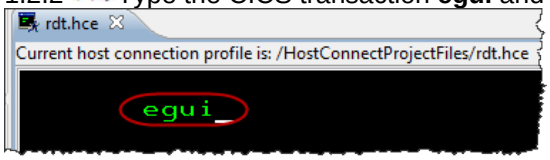
You should be now on the z/OS CICS named CICSTS53. This is the CICS instance where you will deploy the service.

1.2.1 You will need to use the **clear** key. Using the right lower corner, select this icon : It will display possible keys, including the clear button.

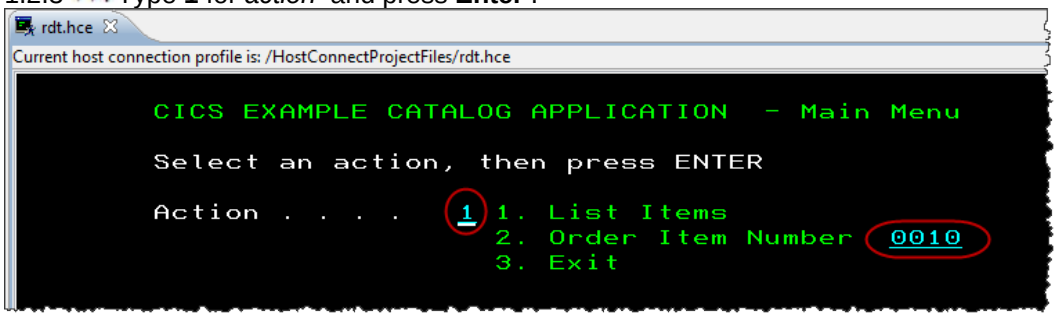
Click on **Clear** (If necessary may also use the **Reset** key after clicking **NextPad**)



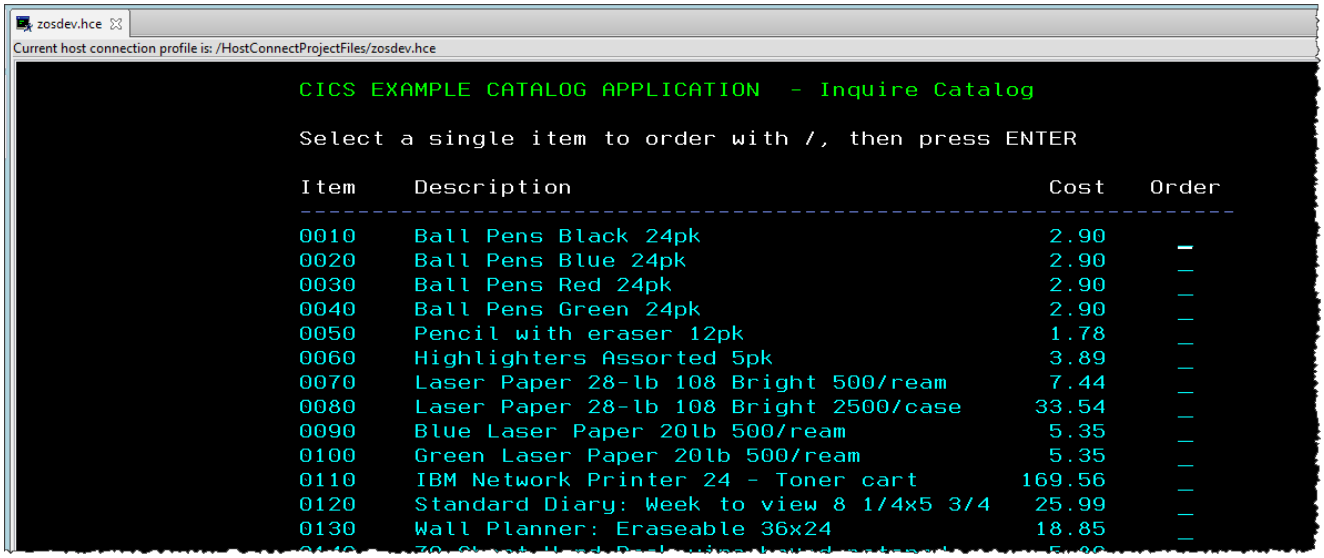
1.2.2 Type the CICS transaction **egui** and press the **Enter** key.



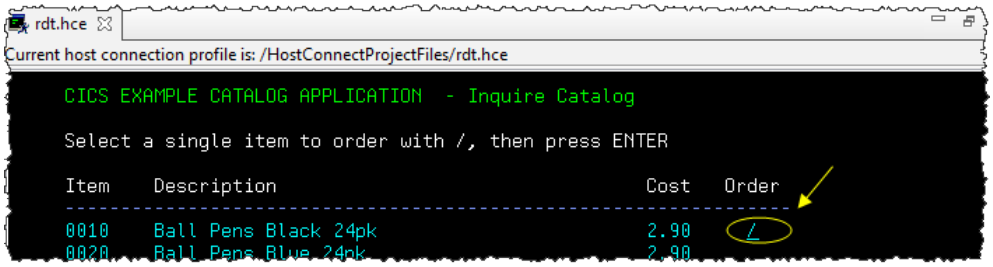
1.2.3 Type **1** for *action* and press **Enter** .



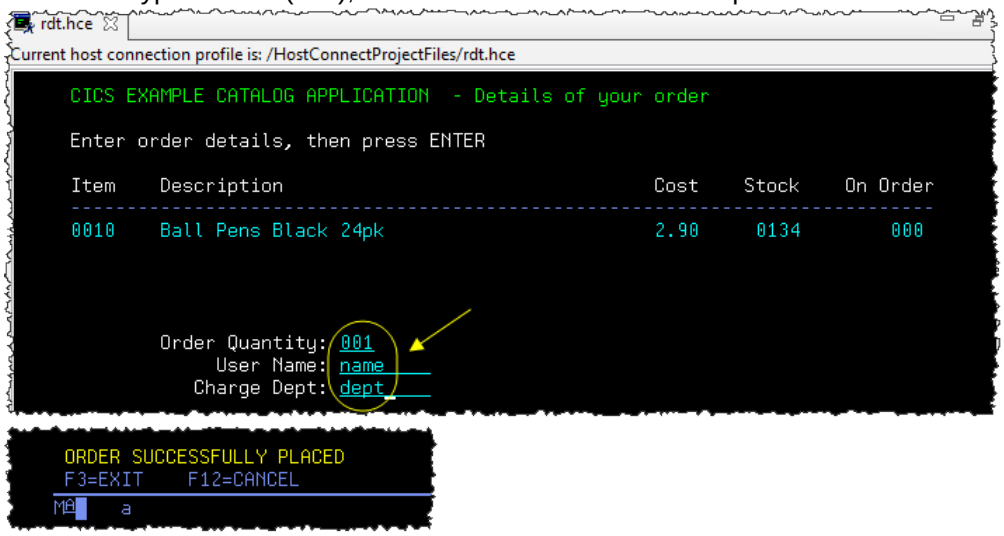
1.2.4 The results will be as shown…



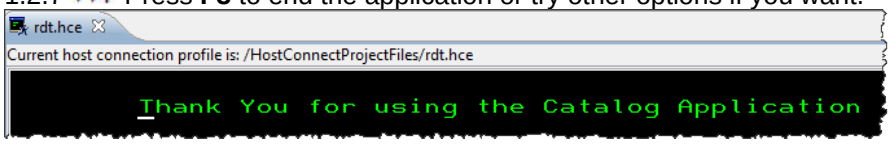
1.2.5 Select the item using the "**/**" as seen below and press **Enter**..



1.2.6 Type a value (**001**), **name** and **name** as below and press **Enter**



1.2.7 Press **F3** to end the application or try other options if you want.



Remember. In case you need to use *Reset* or *Clear* on this screen.

The easiest graphical way: Click on the icons as below:


and


and


1.2.8 **Close** the terminal emulation.

What have you done so far?

You emulated a 3270 terminal using IBM Developer for System z.
You also executed the CICS transaction **EGUI** that executed the program **DFH0XGUI**

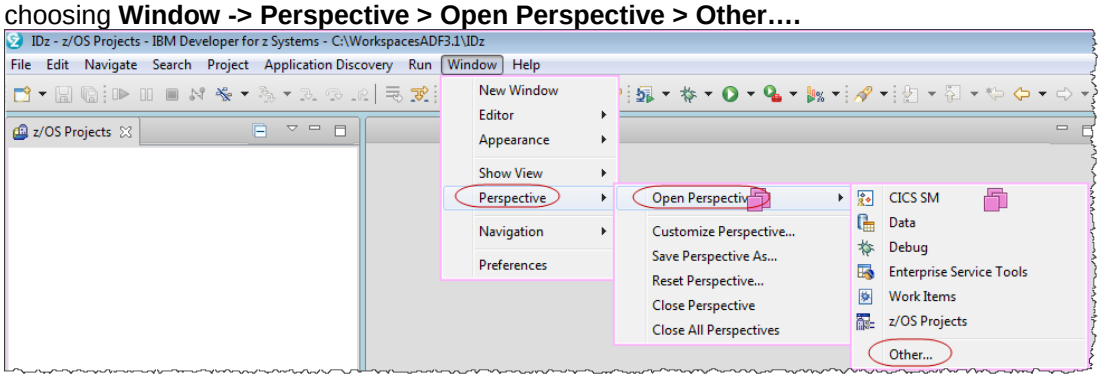
The objective here was to be sure that the COBOL programs to be used soon are working fine and returning a correct data from a VSAM file.
Now you are ready to import one program that invoke the other COBOL programs and create the Web Services.

Section 2 – Discover a candidate API and find its interface

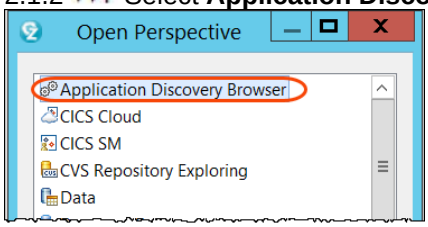
In this section you will use IBM Application Discovery to discover the candidate API and its interface, the copybook you need to make the API.

___ 2.1 Explore the Catalog Manager application

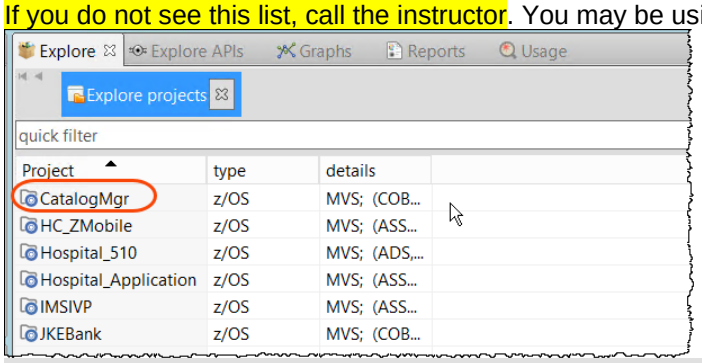
2.1.1 Using IDz, switch to the *Application Discovery Browser* perspective choosing **Window -> Perspective > Open Perspective > Other….**



2.1.2 Select **Application Discovery Browser** and click **OK**.

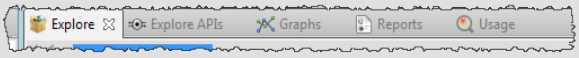


2.1.3 The AD Explore view should show the projects below.
If you do not see this list, call the instructor. You may be using a wrong workspace.



IBM Application Discovery Eclipse Interface Overview?

After opening the Application Discovery perspective there will be several tabs (views) already opened that highlight the main features of the IBM Application Discovery solution.



__1. The initial screen on the Application Discovery Analysis browser contains 5 main tabs which represent the 5 main features of the solution. The first is the **Explore** tab. It contains a list of the projects that have been created in Application Discovery. A project is a logical Application Discovery construct that can represent an application, a subset of an application, specific business functions or whatever the user decides is suitable for them.

__2. The tab **Explore APIs** shows the API and services used in the selected project.
To display the API Usage tab, go to Explore APIs > select API or service > double click API Usage.

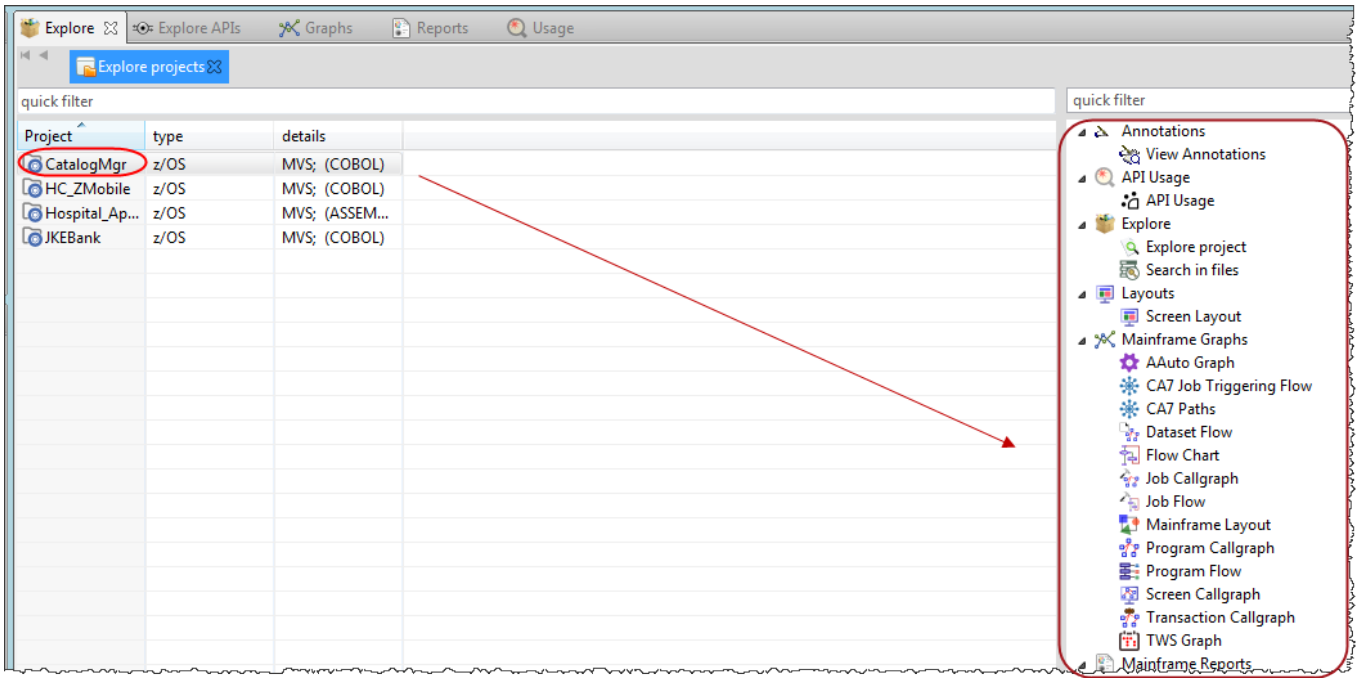
__3. Tab **Graphs:** As the name implies, it will contain graphs of the application being analyzed. An example of such graph could be a **Transaction Call Graph** that we will show in the next section.

__4. Tab **Reports:** It will contain reports generated for a application analysis. An example of such report is an Impact Analysis report which will also be explored.

__5. Tab **Usage:** It will contain usage information for a particular component of a source being analyzed such as a usage analysis for a COBOL variable. This feature will also be explored.

___ 2.2 Explore the Catalog Manager application

2.2.1 Click on the **CatalogMgr** project to see the list of analysis options available for the project

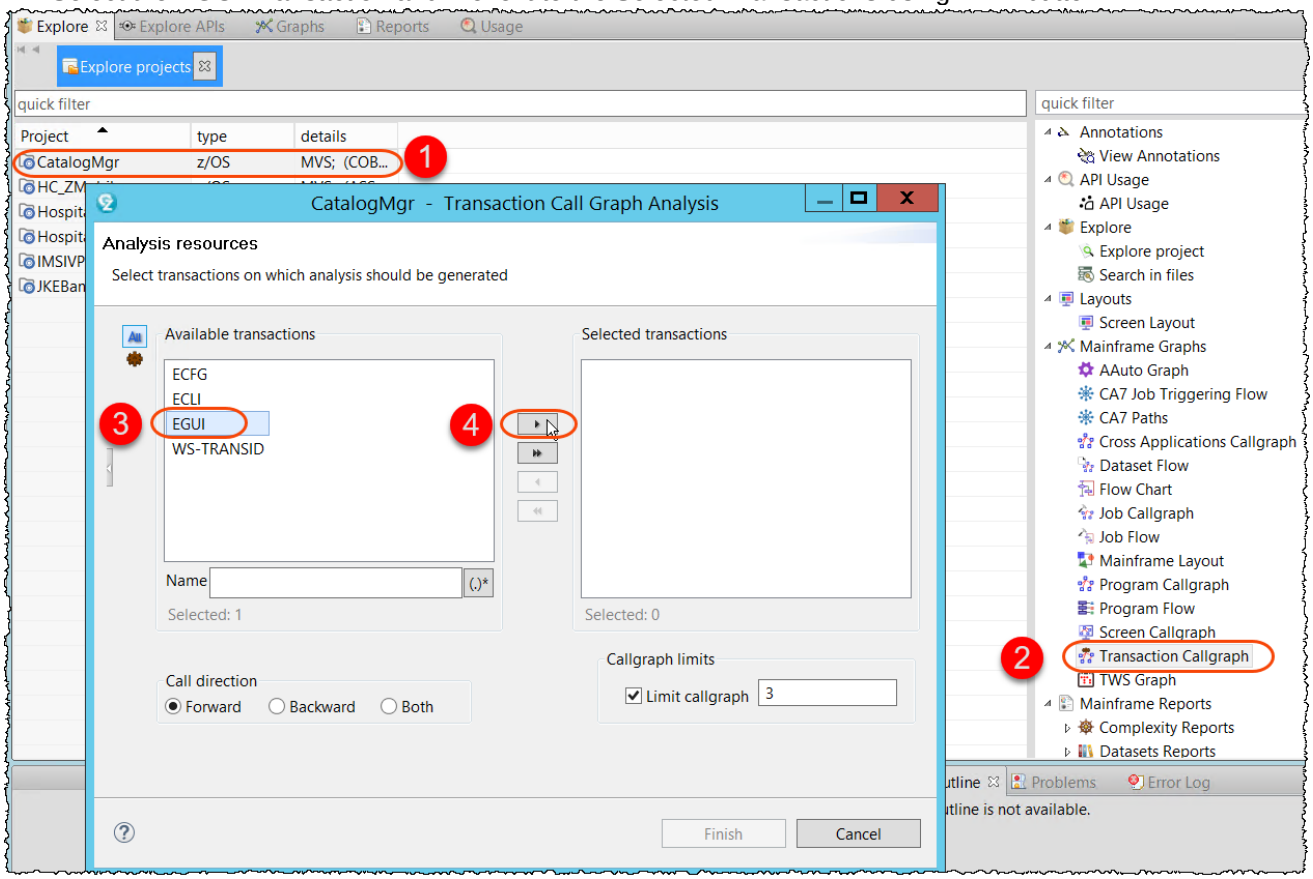


The *Transaction Callgraph* diagram is not displayed?

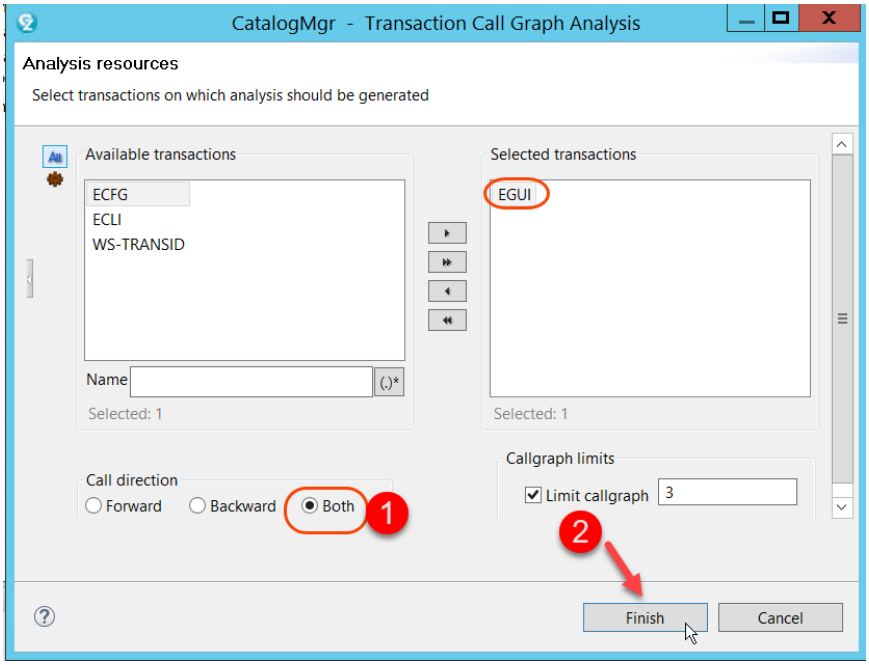
Close IDz then start it again.

2.2.2 To start with the *CatalogMgr* application analysis, double click **Transaction Callgraph** to get the list of transactions used on this application.

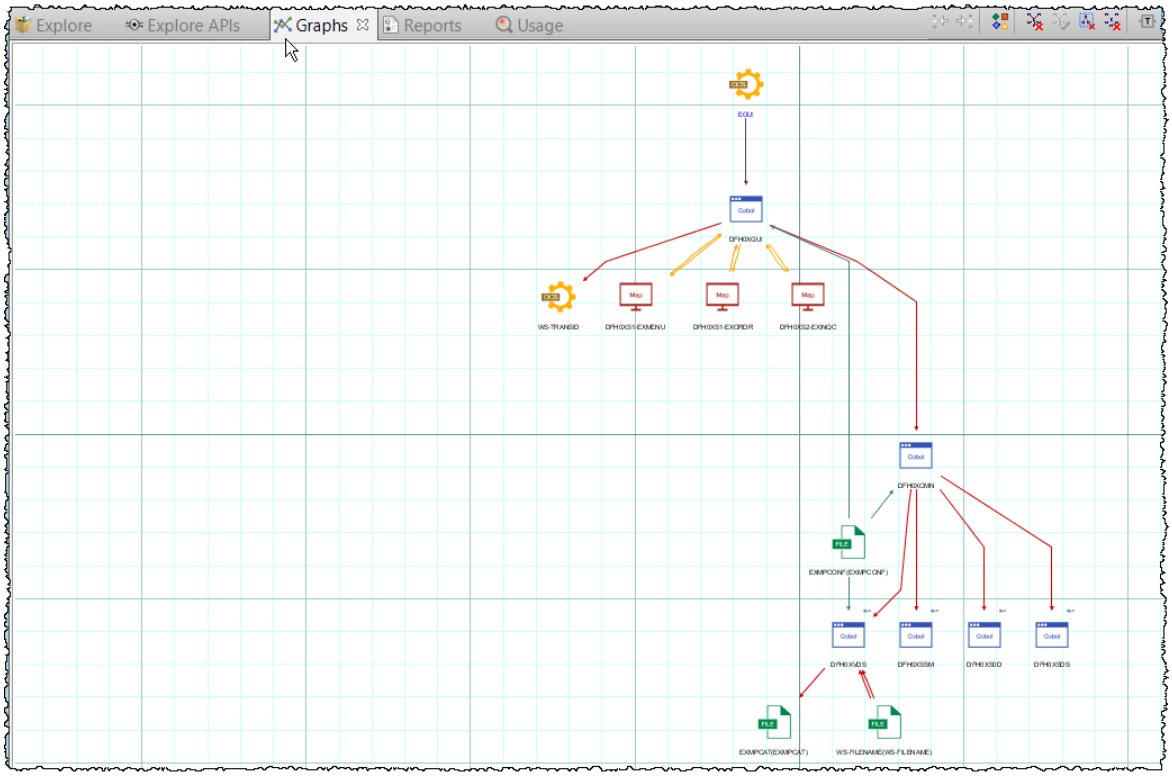
Select the **EGUI** Transaction and move it to the *Selected Transactions* using button.



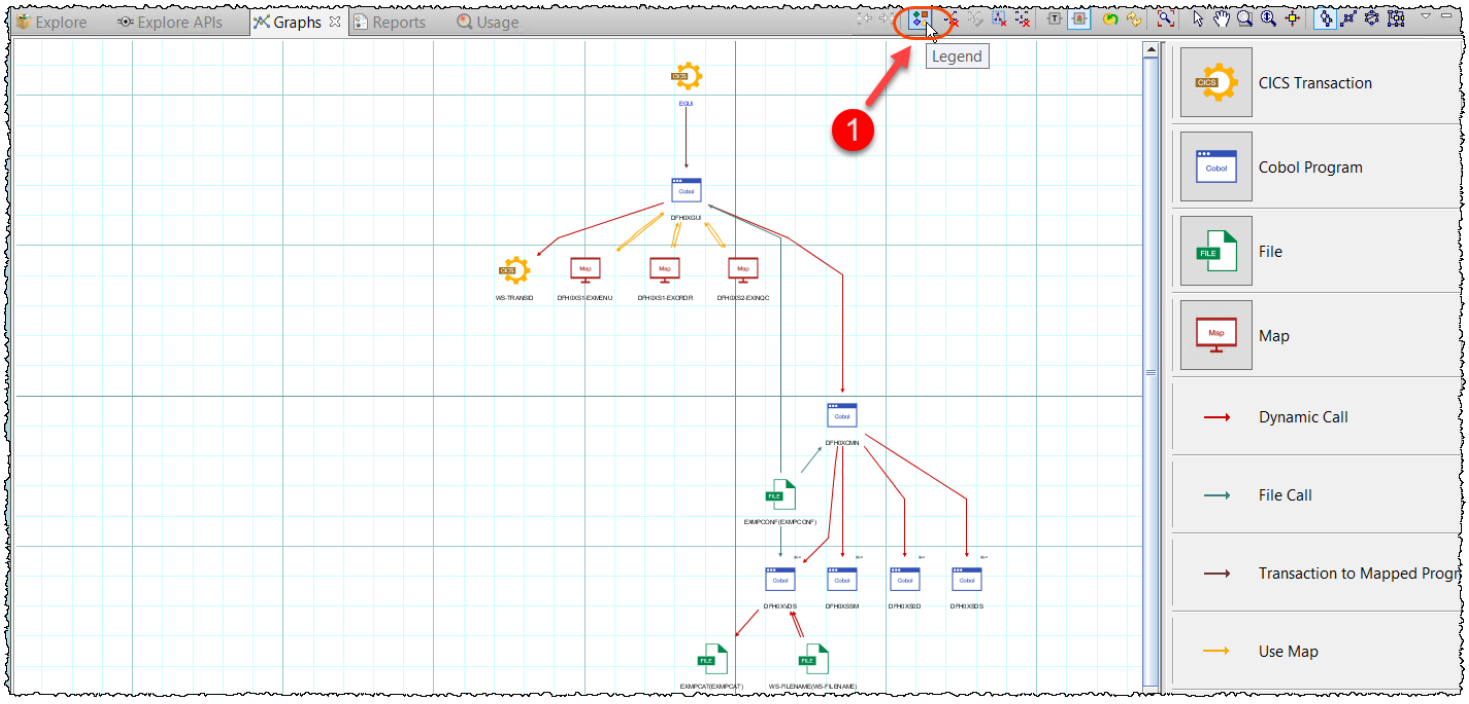
2.2.3 Select the radio button for **Both** as the *Call Direction* and click **Finish**



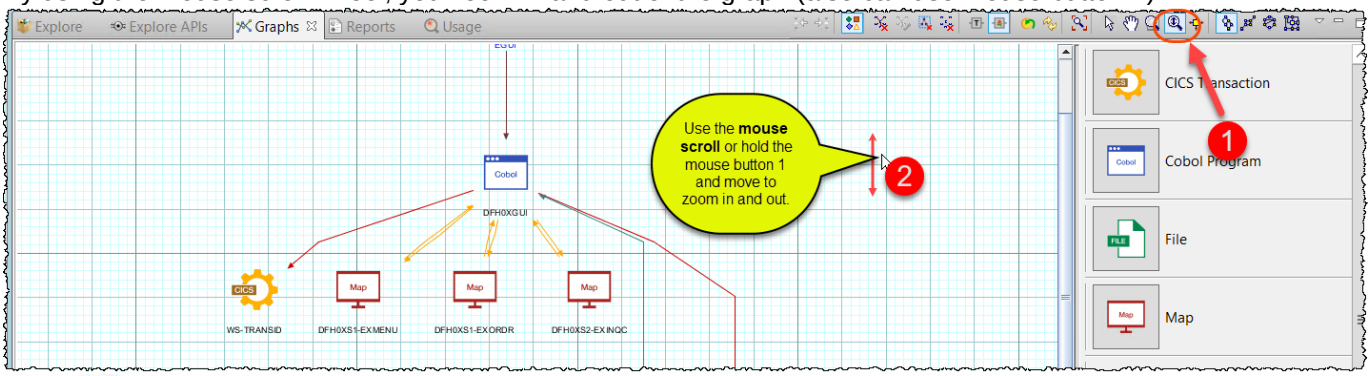
2.2.4 This provides an overview of the flow of the transaction with its maps, databases, or files in this case, and programs



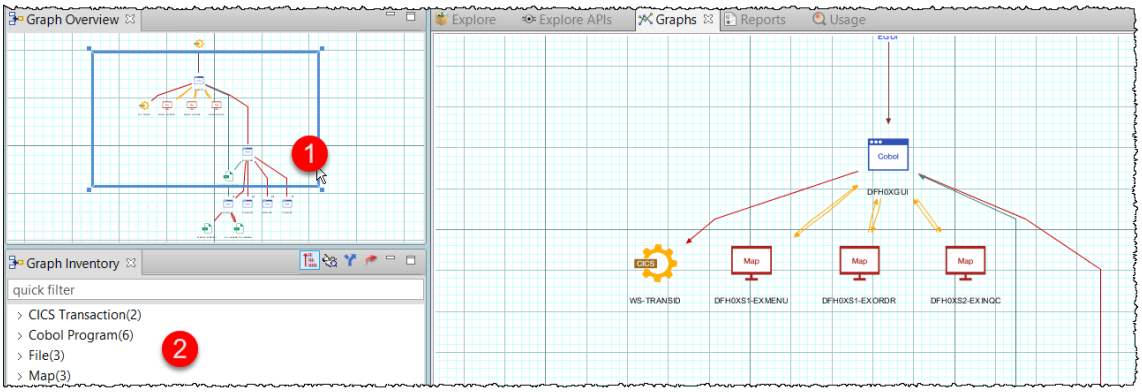
2.2.5 Click on the Legend button so the lines and colored code components can be easily deciphered



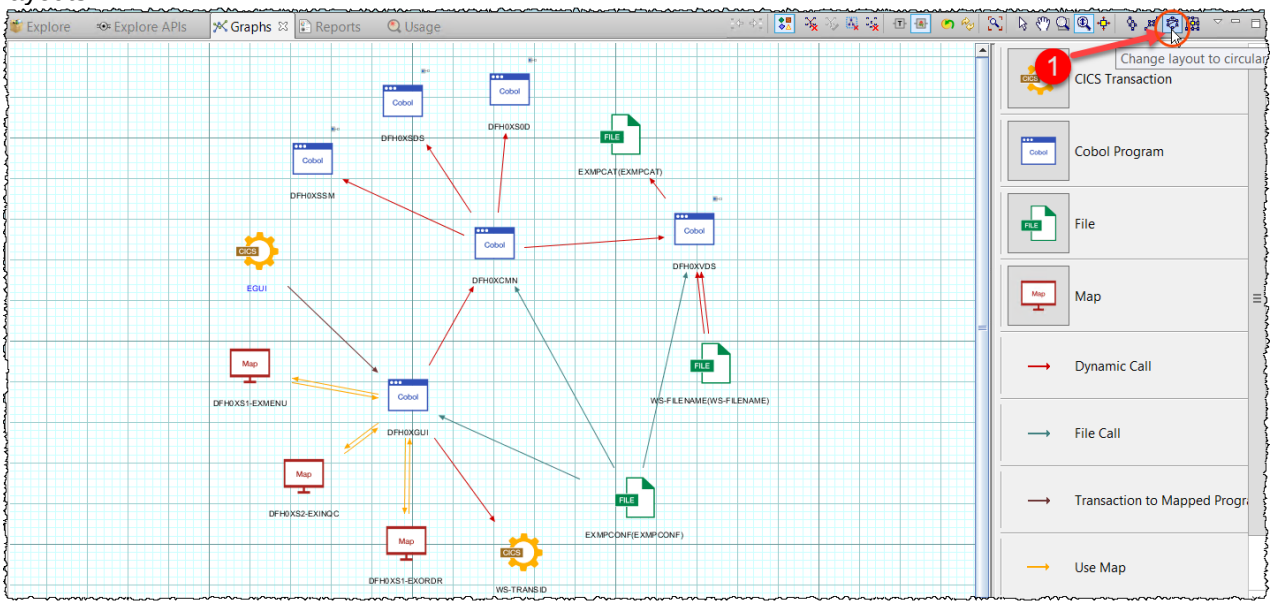
2.2.6 Click on the **Interactive zoom button**  to zoom in and out the graph.
By using the mouse scroll wheel, you zoom in and out of the graph (also can use mouse button 1).



2.2.7 Also using the **Graph Overview** on the left helps to navigate around the graph.
Drag around the blue rectangular box to view different parts of the graph as seen on 1 .below
The **Graph Inventory** on the left shows a consolidated inventory of the components displayed on the graph as seen in 2 below:

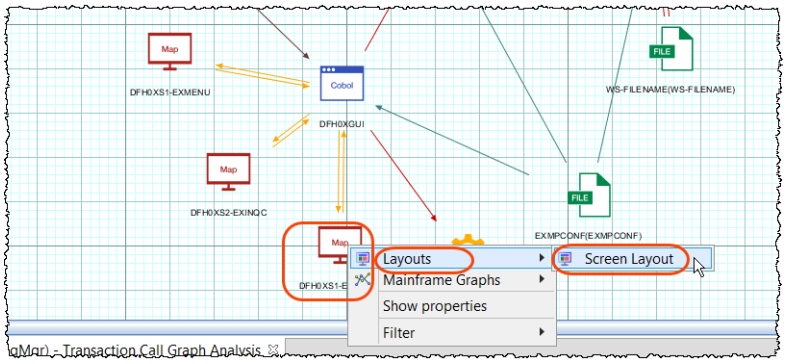


2.2.8 You can show the graphs in multiple layouts. Click from one of the four buttons to view the graph is other layouts

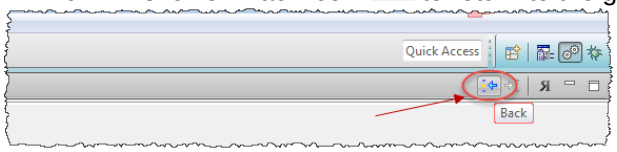


2.2.9 Explore the screen layout in the resultant graph. Note that EGUI invokes the program **DFH0XGUI** and 3 maps.

⏭ Right click on the first map **DFH0XS1 EXMENU** and Select **Layouts > Screen Layout**. Notice how the BMS map is translated into an actual screen layout.

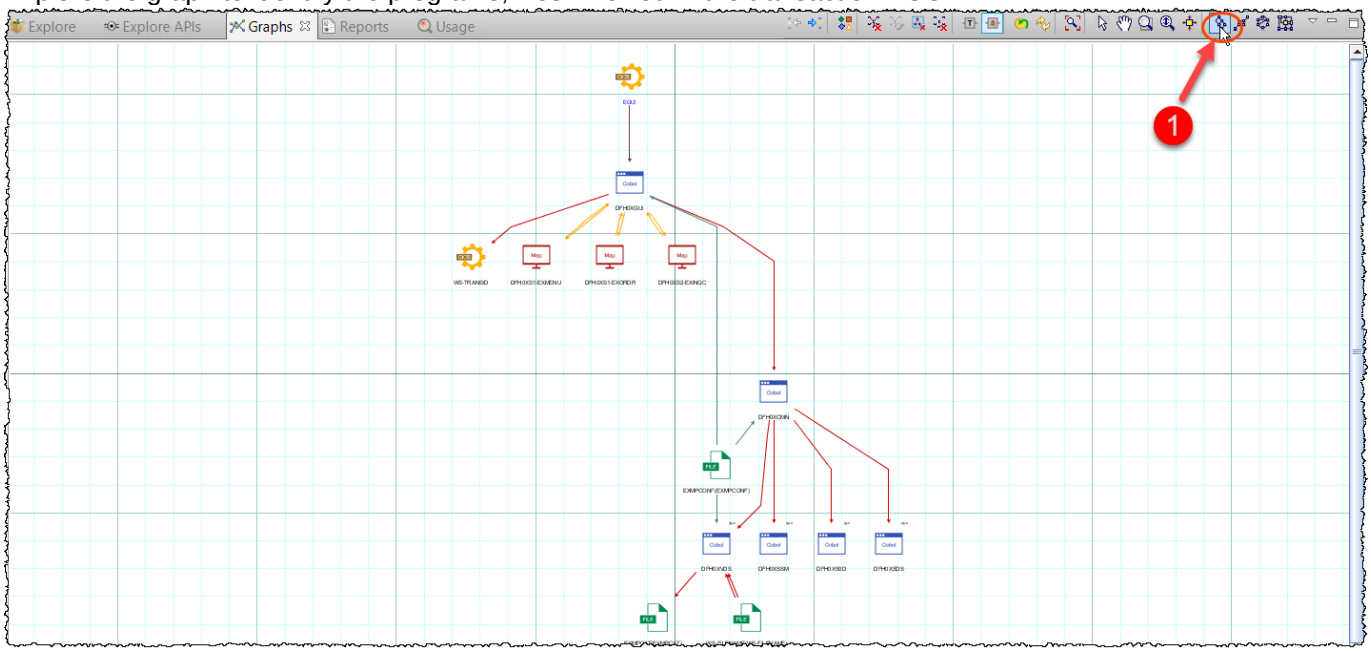


2.2.10 ⏭ Click on Back icon to return to the graphic



2.2.11 ⏭ Click on To change the hierarchical graphic.

Explore the graph to identify the programs, files involved in the transaction EGUI.



Exploring transactions

Notice how easily the set of programs and the flow related to an entire transaction was brought up with the help of transaction and program call graphs. These graphs also indicate the databases, files and screens used by the transaction.

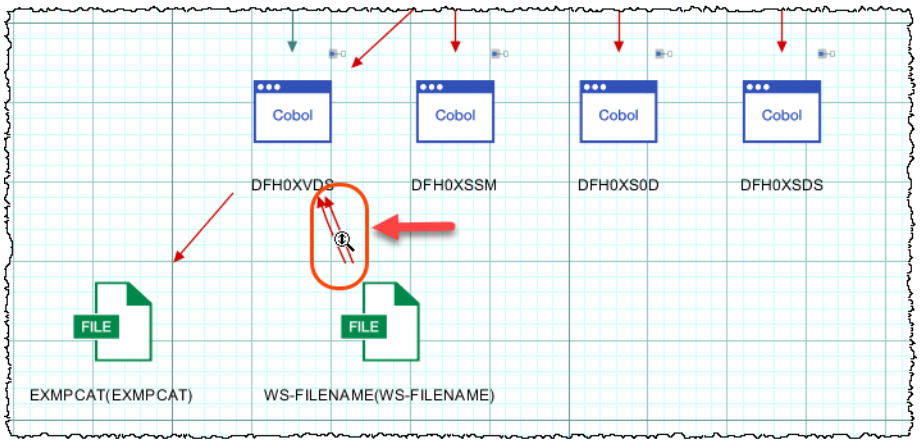
With this exercise, we have seen how we can Identify the transaction flow and related programs

___ 2.3 Identify the transaction flow and related programs

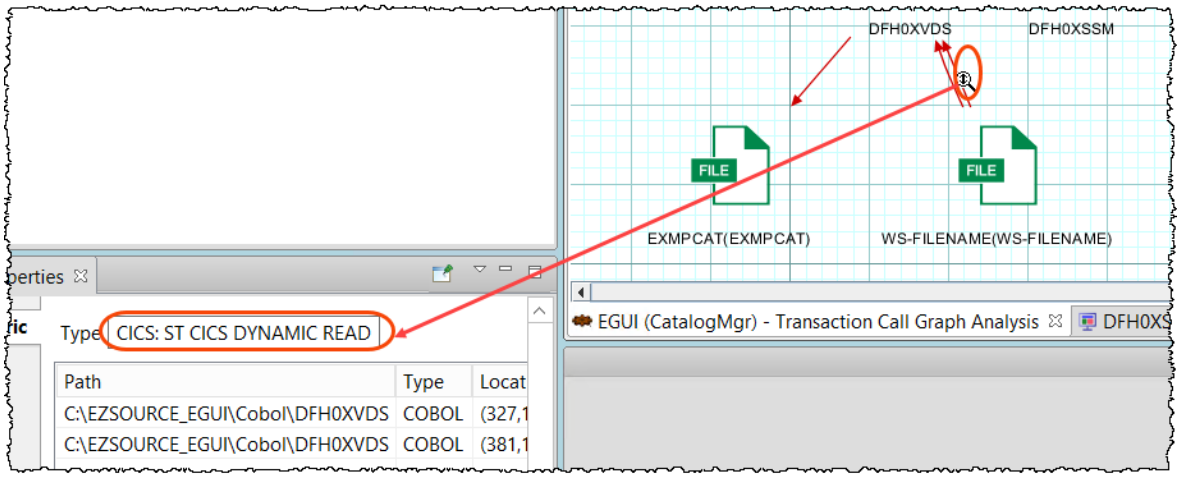
Since the objective is to create an API from the *Catalog Manager Inquiry flow*, one of the different methods of analysis is to work backward from the file. The **EGUI** transaction flow interacts with the **VSAM Catalog Manager file** as viewed from the graphs. This is a good starting point for a developer, to identify the transaction flow.

2.3.1 ⏭ Using the zoon icon 

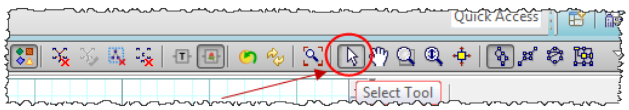
2.3.2 ⏭ Zoom in to the **WS-FILENAME** file.
The arrows that point into the file indicate a modify and those that point out of the file indicate a READ.
⏭ Click on the arrow that points away from the file (the second on right).



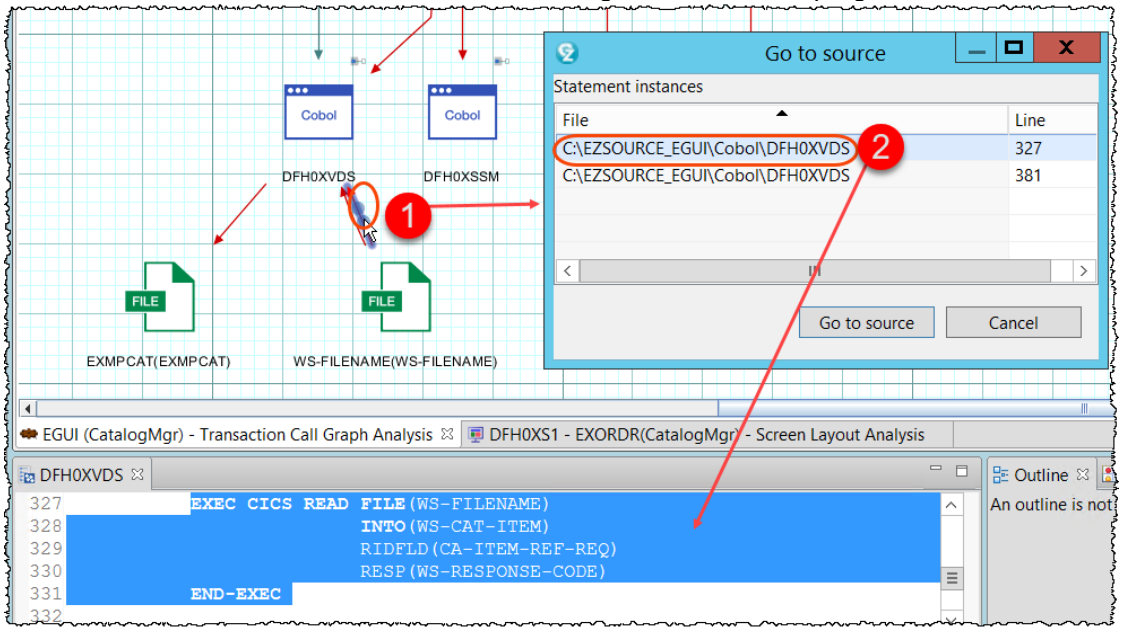
2.3.3 Notice the **Properties** tab at the left bottom of the screen.
It indicates that the arrow represents a link of **`CICS: ST CICS DYNAMIC READ`**. It has been identified that the program **DFH0XVDS** reads from the file and is the starting of the reading of the Catalog Manager Transaction flow working backwards from the file access



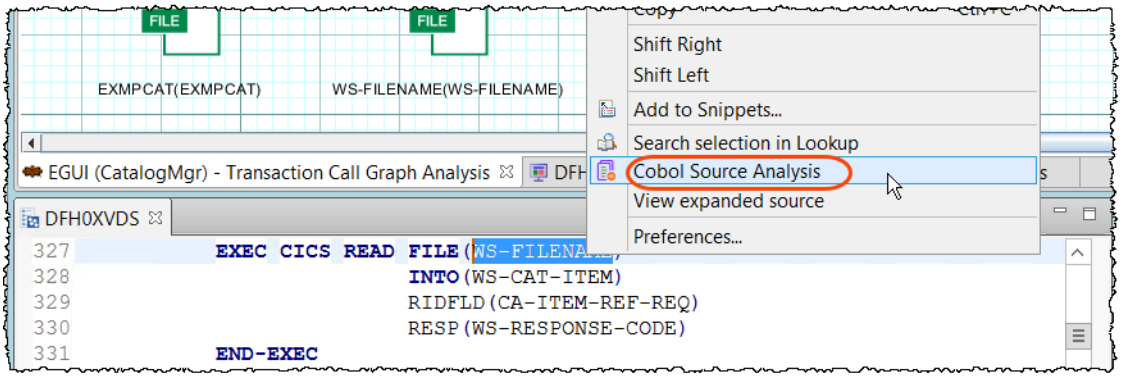
2.3.4 From the Properties tab you can see that the filename is in fact a dynamic filename. We can explore deeper to understand where the Filename gets moved to the **WS-FILENAME** variable field.
⏭ Click on icon  (Select Tool) since you need to explorer the graphic



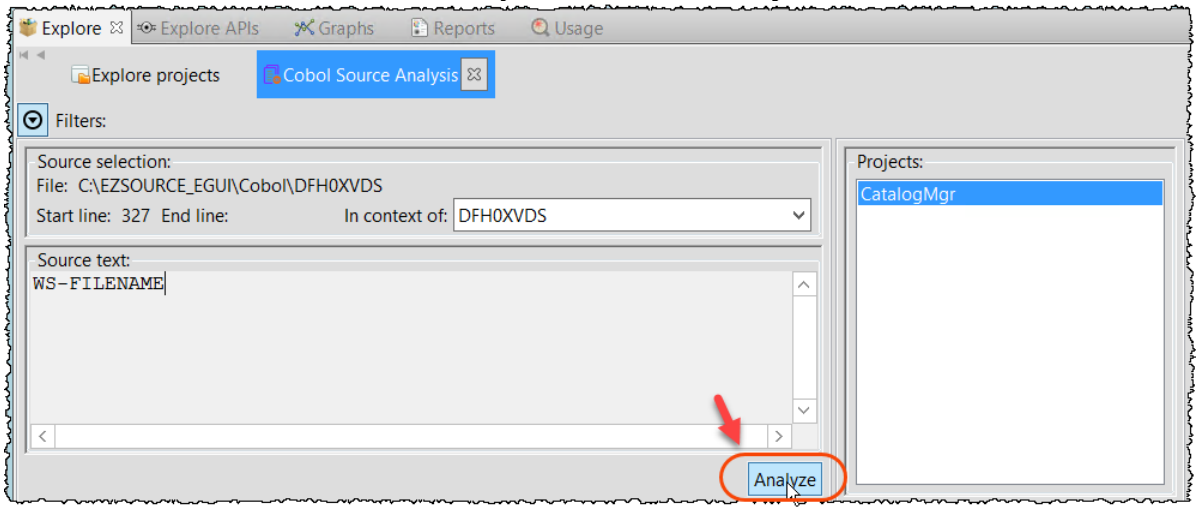
2.3.5 _By double clicking the arrows from the file and to the line 327 the program the source will open in the Source view, at the bottom of the editor, mentioning the name of the program. Click **Go to source**



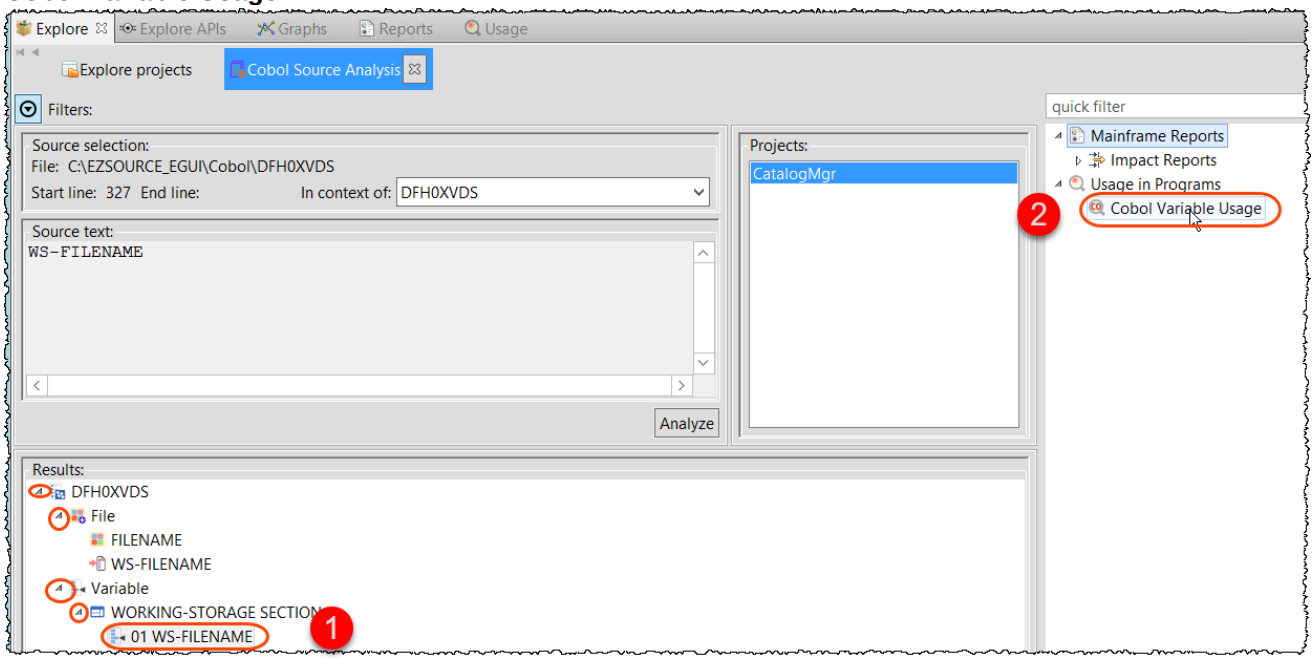
2.3.6 In the Source view select the variable **WS-FILENAME,** right click and select **Cobol Source Analysis**



2.3.7 From the new *Cobol Source Analysis* Tab click the **Analyze button**

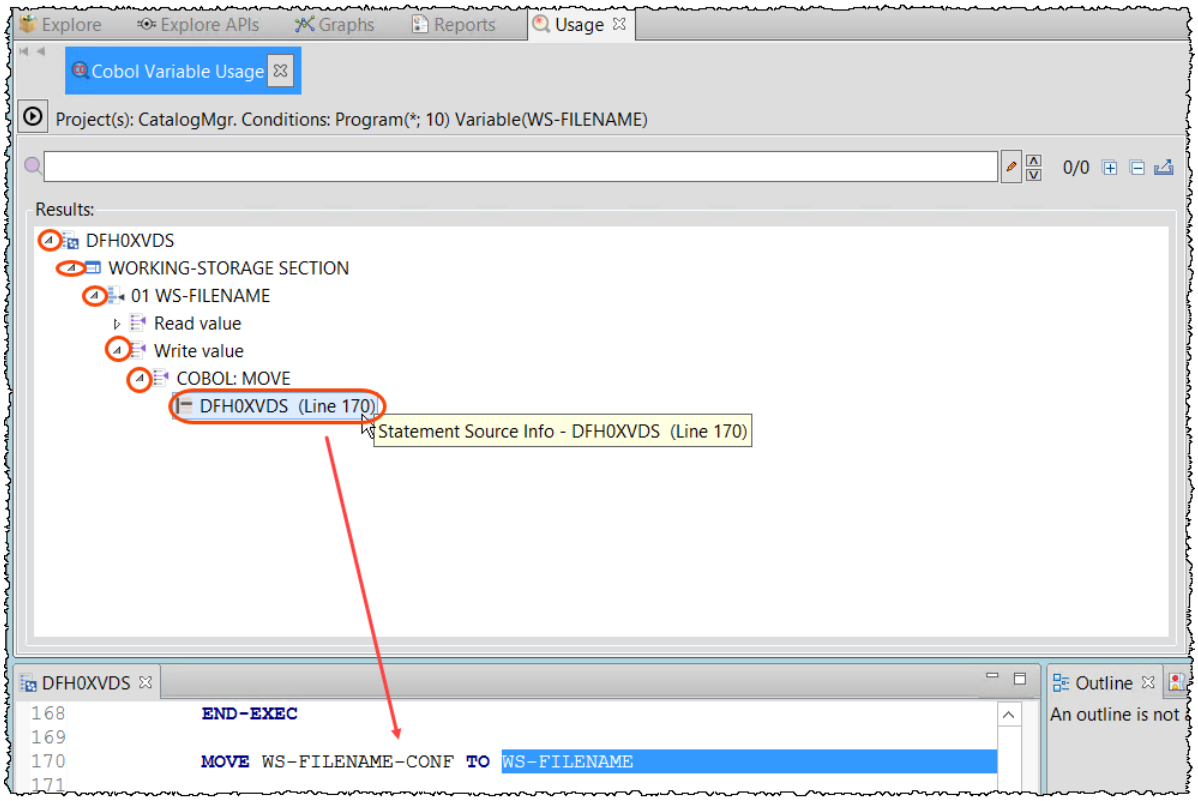


2.3.8 ⏭ Expand **DFH0XVDS, File, Variable** and **WORKING-STORAGE SECTION** and select the **WS-FILENAME** variable. On the right-hand side, a context sensitive menu opens when selecting **WS-FILENAME,** double click the **Cobol Variable Usage**



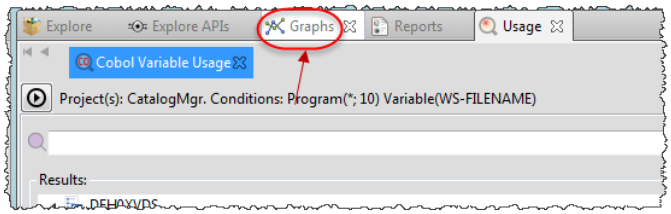
2.3.9 Since we are interested in the value that gets written to the variable **WS-FILENAME**:

⏭ Expand **DFH0XVDS**, the **WORKING-STORAGE SECTION** for the Write-value and dig into the COBOL:MOVE statement by double clicking the line of the program. It will bring you to the instruction where **WS-FILENAME** gets filed

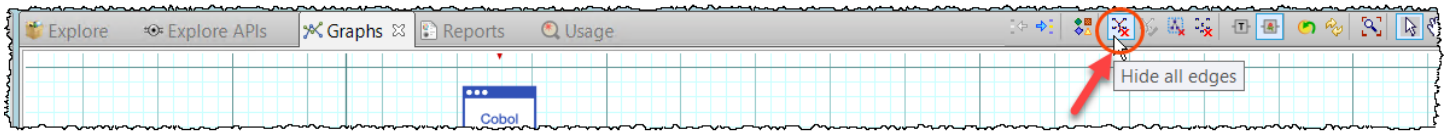


2.3.10 The value of the filename gets read from another file which value gets read into the variable **WS-FILENAME-CONF**. We are not going to analyze deeper to prove the value, but you can try yourself same kind as analysis for **WS-FILENAME-CONF**

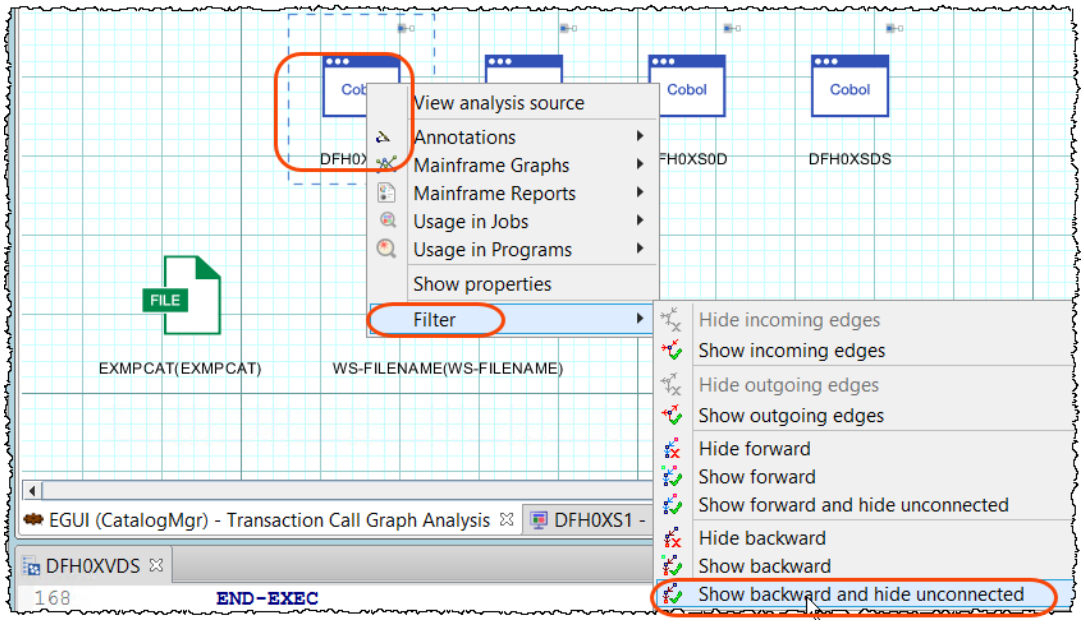
2.3.11 ⏭ Go back to the diagram clicking on **Graphs** tab:



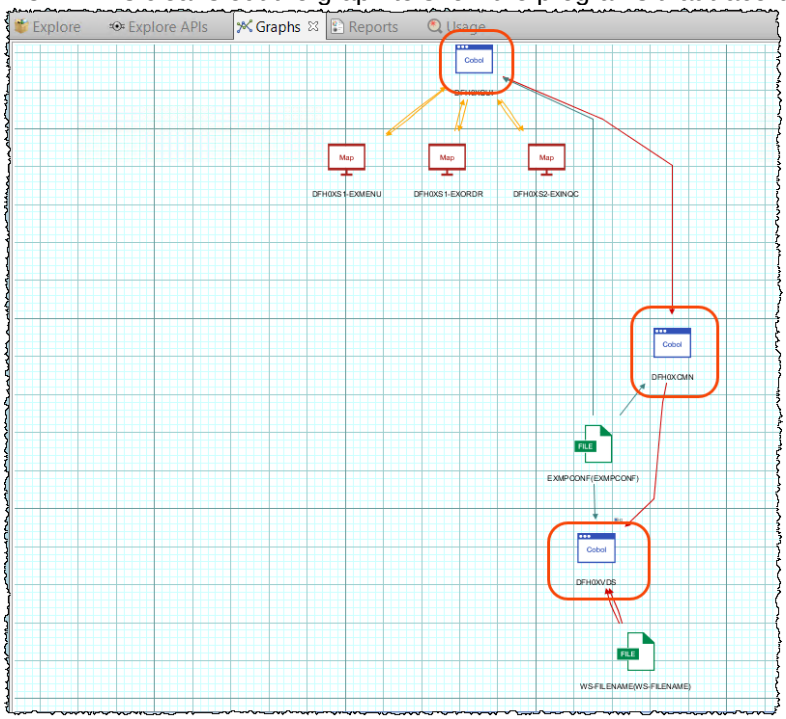
2.3.12 ⏭ To isolate and identify the other programs that make up the Catalog manager Inquiry, **click on the icon**  to Hide all edges



2.3.13 ⏭ Right click on program **DFH0XVDS** and select **Filter > Show backward and hide unconnected**.



2.3.14 This cleans out the graph to show the programs that trace backward from the **READ** program **DFH0XVDS**



2.3.15 The graph clearly indicates the backward trace of the Inquire transaction from C**atalog Manager File > DFH0XVDS > DFH0XCMN > DFH0XGUI**

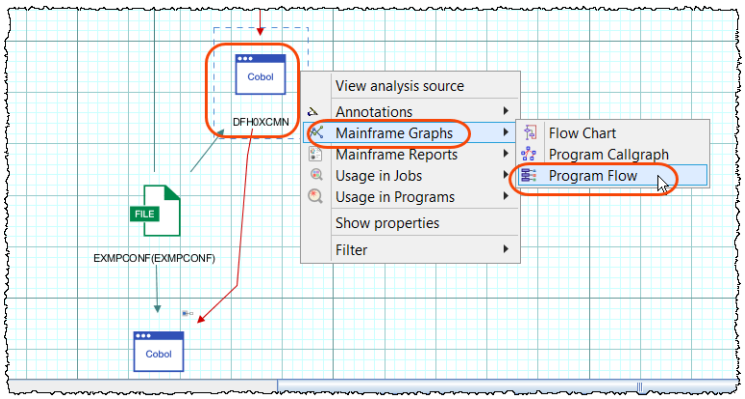
Isolating transaction flows

Notice how easily the programs querying the database and/or files can be identified.
With this exercise, we have seen how we can Identify the read transaction flow and related programs

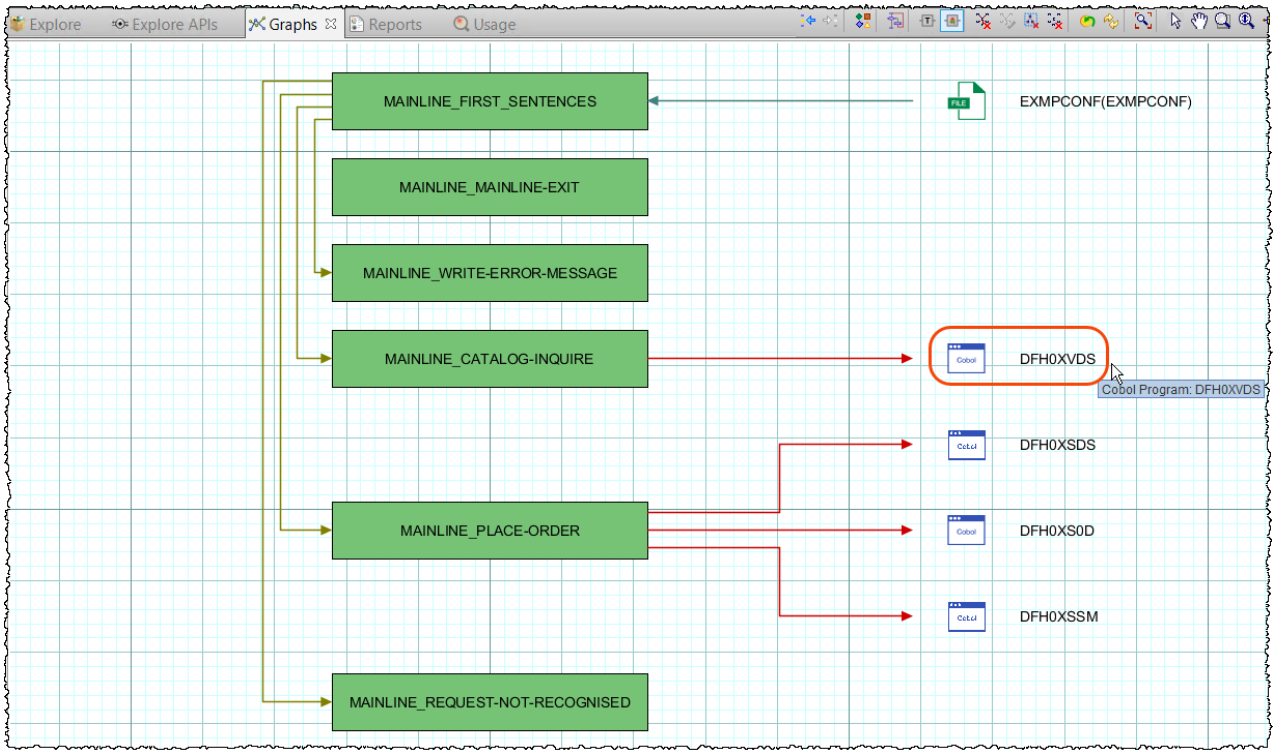
___ 2.4 Identify the program that can be exposed as an API

From the above exercise, the Read transaction flow has been identified **Catalog Manager File > DFH0XVDS > DFH0XCMN > DFH0XGUI**. Which among these programs can we expose as an API. DFH0XGUI is a presentation logic program using BMS maps and hence cannot be exposed as an API. One of the ways to identify which amongst DFH0XCMN and DFH0XVDS can be exposed as an API is to check if DFH0XCMN has any additional functionality.

2.4.1 Right click on **DFH0XCMN** to invoke the Program flow by selecting **Mainframe Graphs > Program Flow**.



2.4.2 The program flow chart indicates that **DFH0XCMN** calls other programs like **DFH0XVDS**



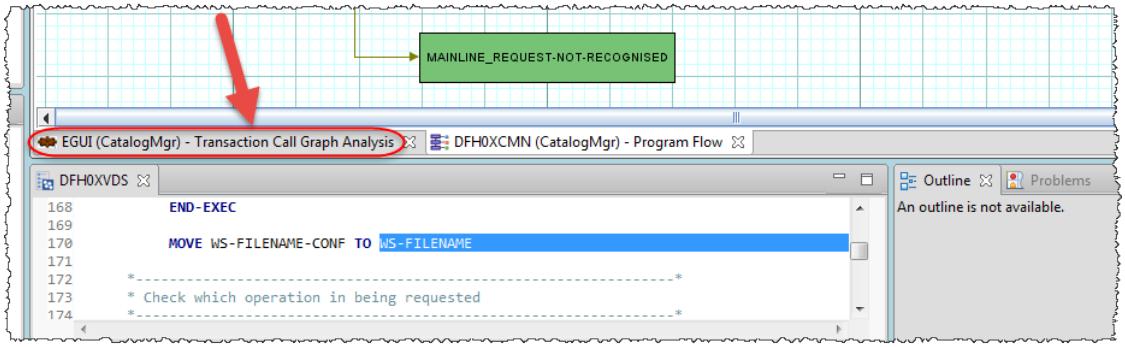
This could include other business logic that is invoked before returning to the presentation logic program DFH0XGUI. Since the aim is to replicate the Inquire functionality in EGUI, **DFH0XCMN** is the better API candidate

Exploring program logic

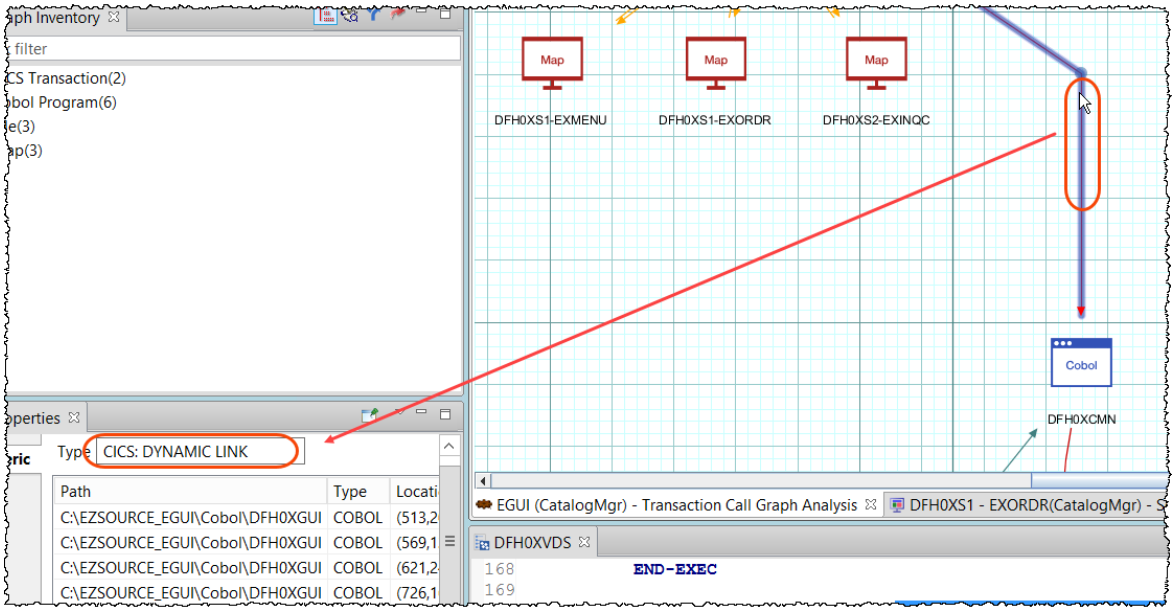
Program flow can be easily analyzed from Application Discovery. The flow gives a view of the called programs, files etc.
With this exercise, we have seen how we can Identify the program that can be exposed as an API

___ 2.5 Identify the interface structure that can be used to create an API

2.5.1 Click on **Transaction Call Graph Analysis** tab to return to the transaction flow graph.



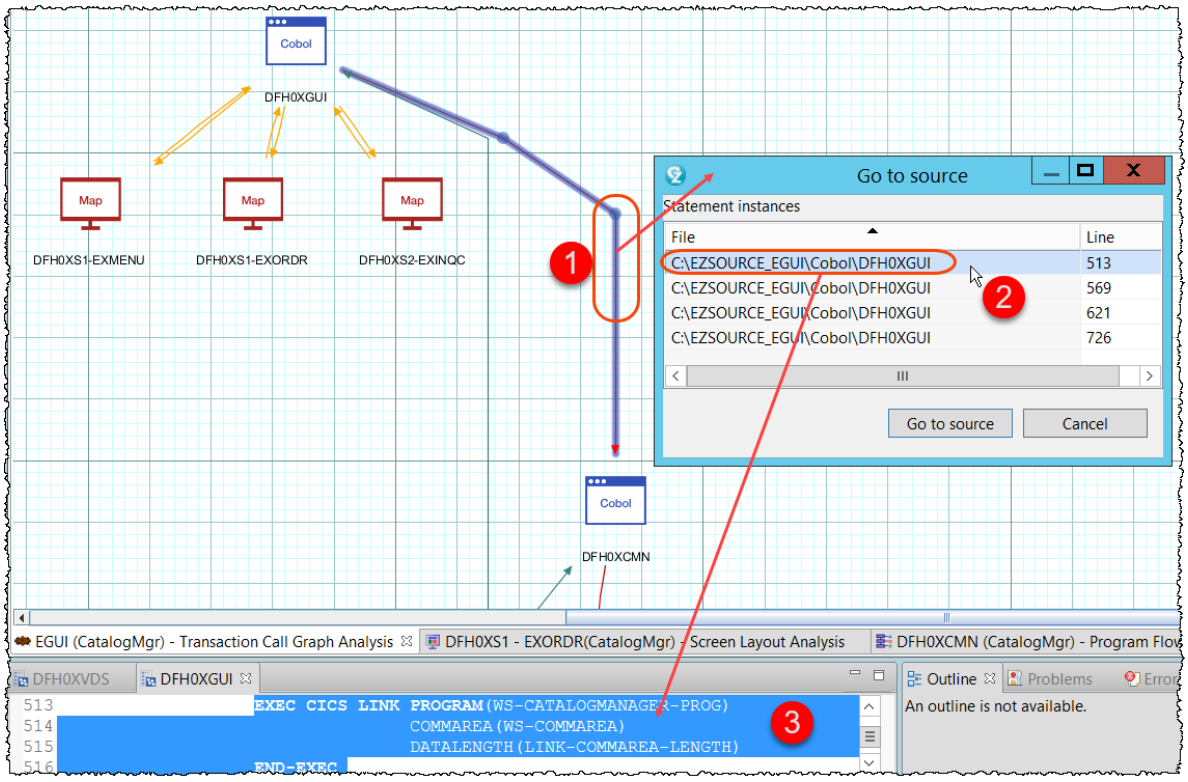
2.5.2 ⏭ Click on the arrow that links **DFH0XGUI** to the program that we would like to expose as an API : **DFH0XCMN.**
Observe the call type from the Properties tab, indicating that the program is invoked via a **CICS :DYNAMIC LINK**.



2.5.3 ⏭ To check if the call is via a COMMAREA or a CHANNEL, **double click on the link** and select the statement from the pop-up to view the source that represents the link.

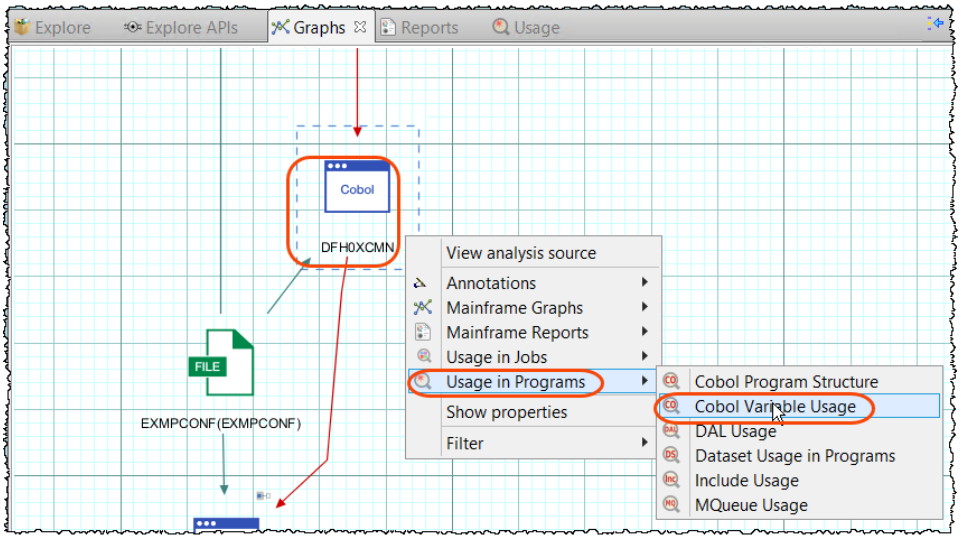
⏭ Click on **line 513** of the dialog *Go to source*
The program LINK statement opens to reveal that the call is via a **COMMAREA**

⏭ Click **Go to source**.

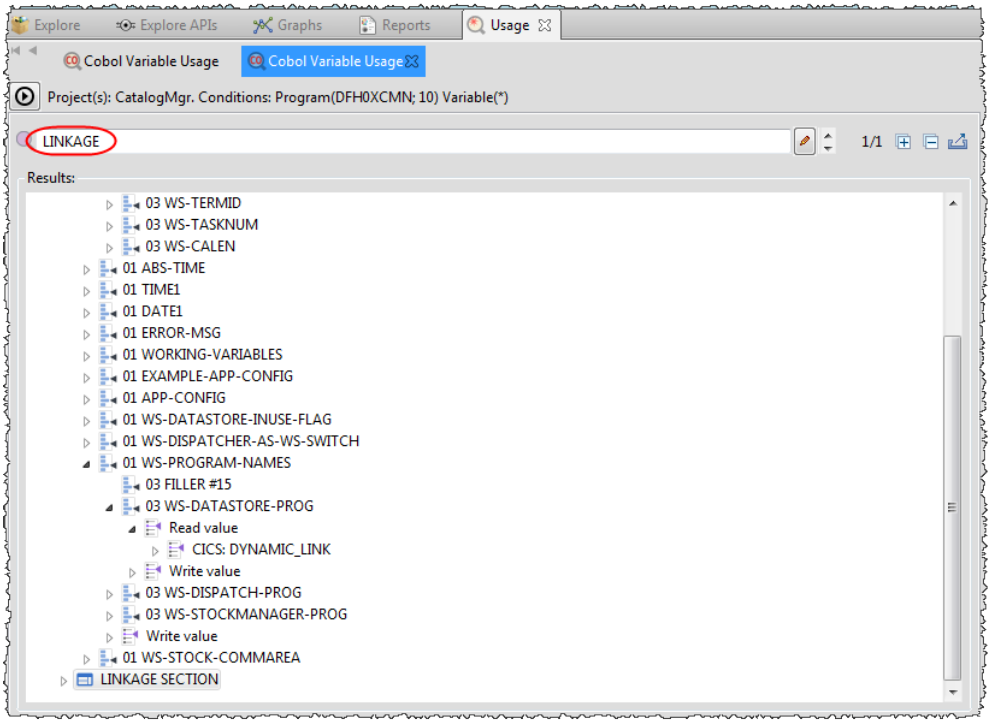


With this exercise, we have seen how to Identify the type of interface that is used in the Linkage section of a program.

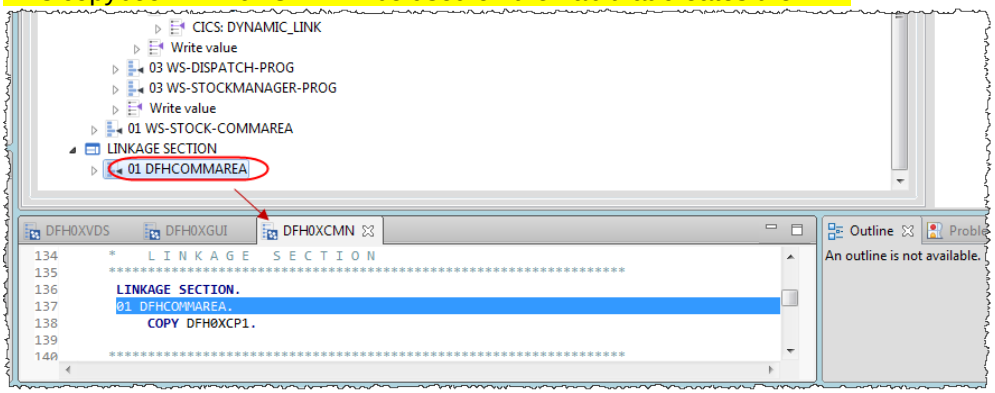
2.5.4 To understand the interface structure used to invoke DFH0XCMN, right click on the program **DFH0XCMN** and select **Usage in Programs > Cobol Variable Usage**



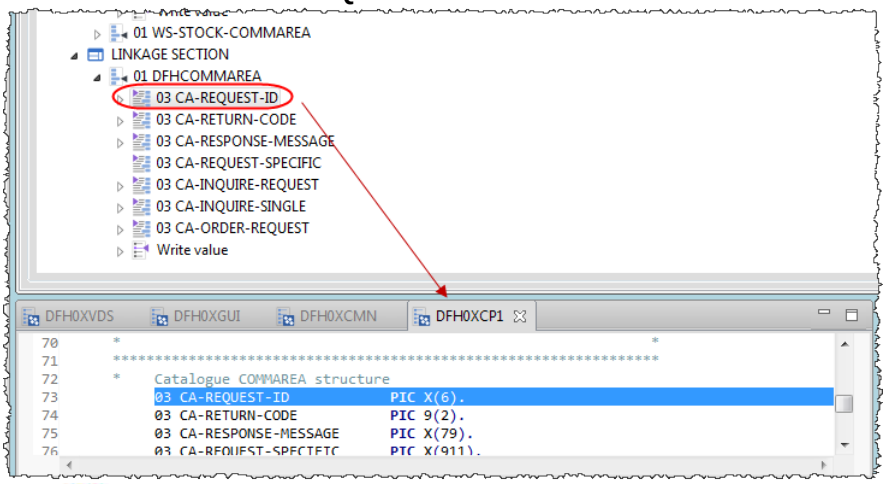
2.5.5 In the Cobol variable usage tab that pops up, enter the text **LINKAGE** to navigate to navigate to the linkage section. This will show the record structure using which DFH0XCMN is invoked



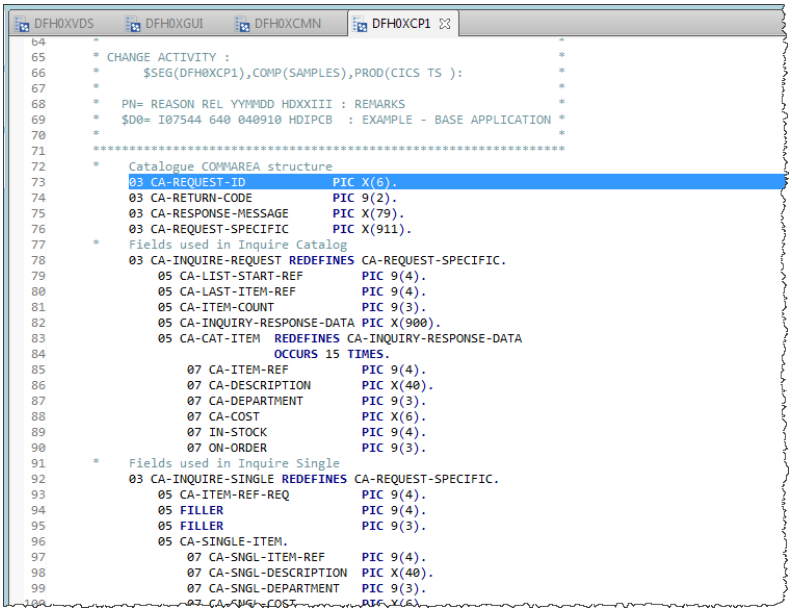
2.5.6 Expand the **LINKAGE SECTION** and double click on **DFHCOMMAREA**
The copybook structure used to invoke DFH0XCMN is shown under the Linkage Section as **DFH0XCP1**
This copybook **DFH0XCP1** will be used on the Lab that creates the API**.**



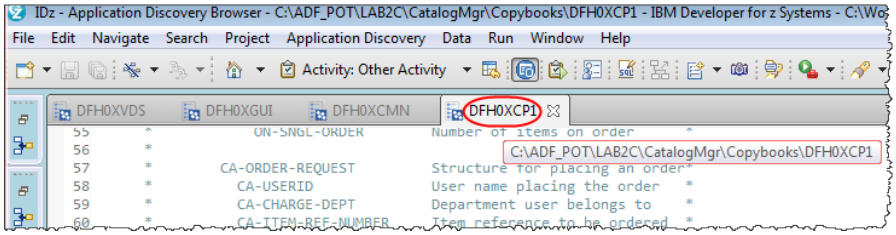
2.5.7 To view the copybook, expand the **DFHCOMMAREA** from the Cobol Variable Usage section and double click on the first field **CA-REQUEST-ID** variable.



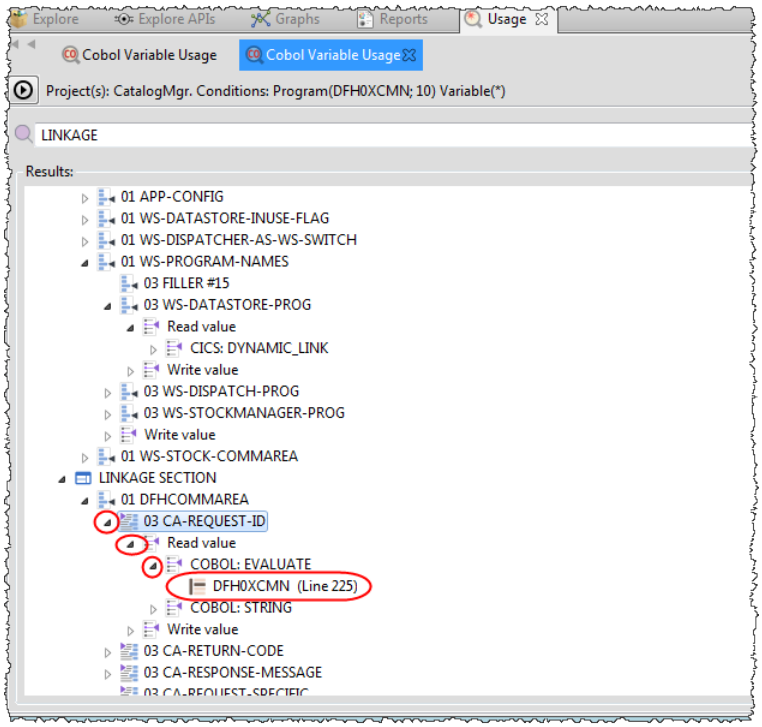
2.5.8 Maximize the view (double click on the title) of the source of the **DFH0XCP1** copybook to see that it contains a number of **REDEFINES.**



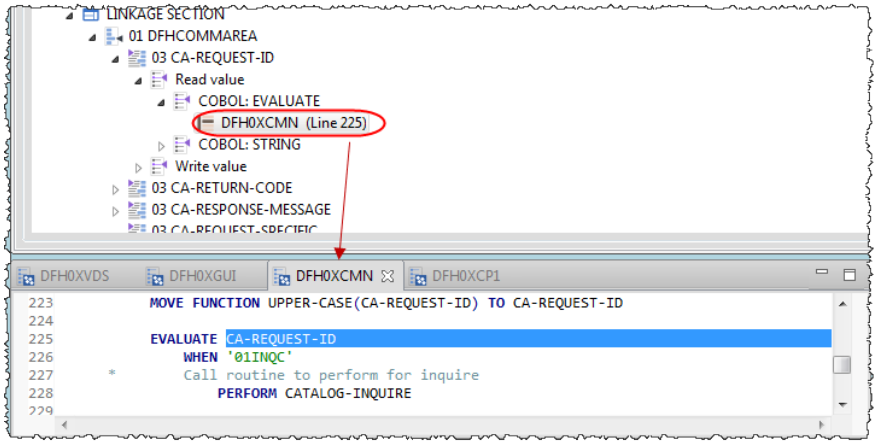
2.5.9 Double click on the **DFH0XCP1** title to return as before



2.5.10 Explore how the **CA-REQUEST-ID** is read and evaluated when **DFH0XCMN** is called



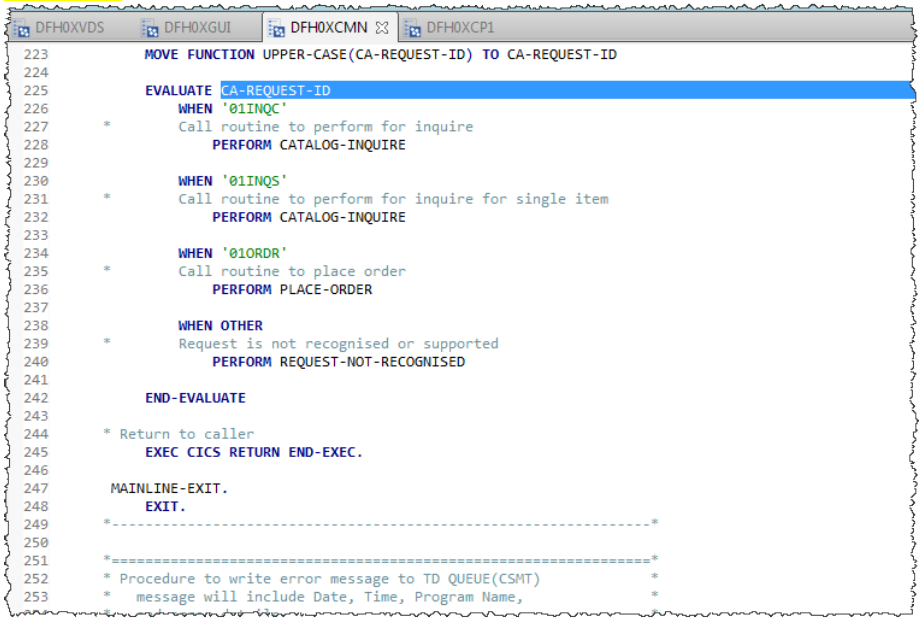
2.5.11 Double click on the line of source in **DFH0XCMN** where the CA-REQUEST-ID is evaluated. You should see in the source below how the CA-REQUEST-ID is used to determine what routine in the program to call.



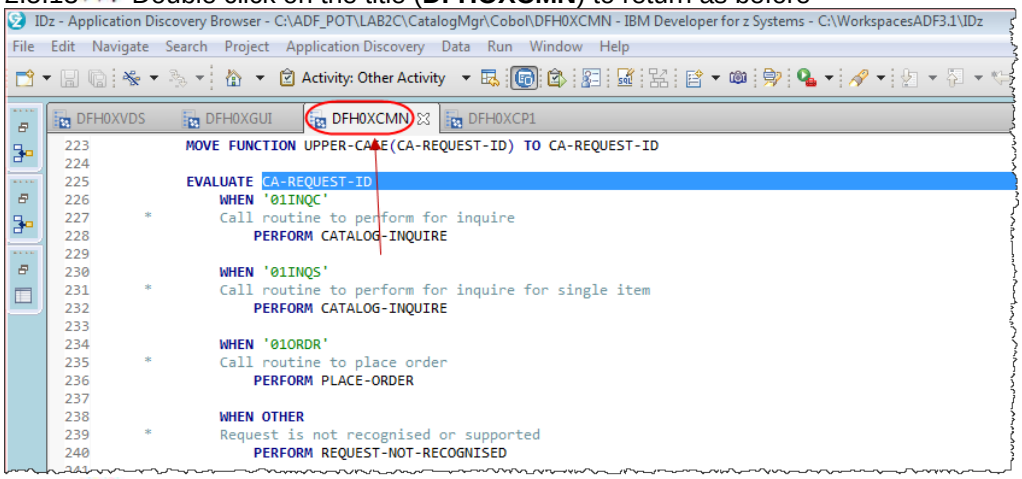
2.5.12 Maximize the program source double clicking on the title (**DFHOXCMN**).
We see that DFH0XCMN expects one of three values to be passed in CA-REQUEST-ID:

- **01INQC** when the program is called for an inquire request
- **01INQS** when the program is called for an inquire for a single item
- **01ORDR** when the program is called for an order request

We will also use this information when we create an API using the z/OS Connect EE API toolkit in the next section of the lab.



2.5.13 Double click on the title (**DFHOXCMN**) to return as before



2.5.14 Use **Ctrl + Shift + F4** or right click on tab and click "**Close all**" for close all opened editors

With this exercise, we have seen how you can explore the structure of the interface for the business logic program **DFH0XCMN.**
This helps us to understand the interface structure that can be used to create an API.
We have now explored how a mainframe legacy application can be discovered using IBM Application Discovery. In other LAB (LAB 2D) we will API enable the application

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

Updated August 13 2019 by Wilbert, Reviewed by Regi

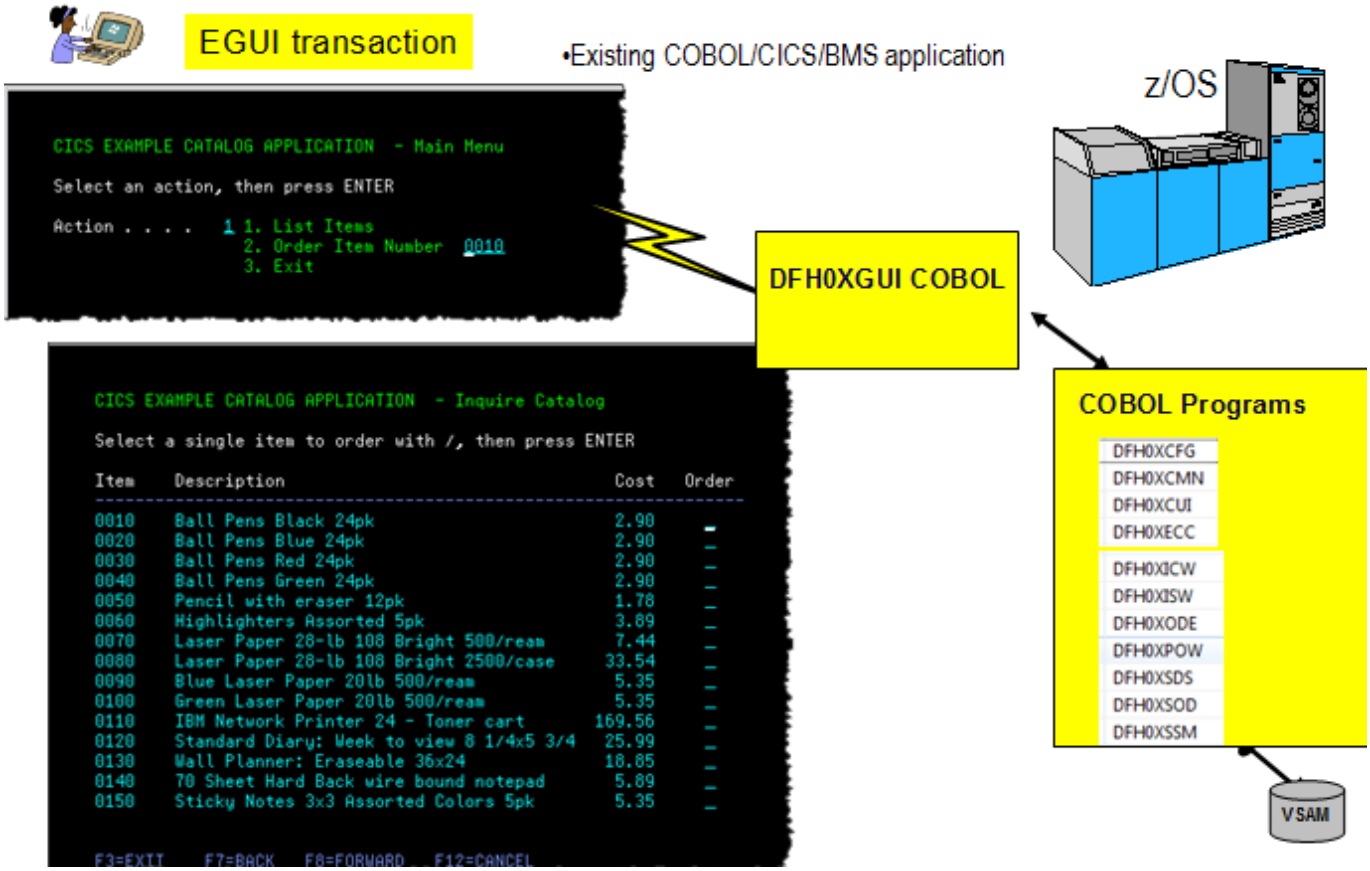
Acknowledgments:
We would like to thank the original authors of this exercises:
Aymeric Affouard, Eric Phan, Paul Pilotto & Nigel Williams.

A previous LAB (LAB 2C) shows how to use Application Discovery to identify components that are API candidates.

Overview of development tasks

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

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



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

z/OS Connect EE

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

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

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

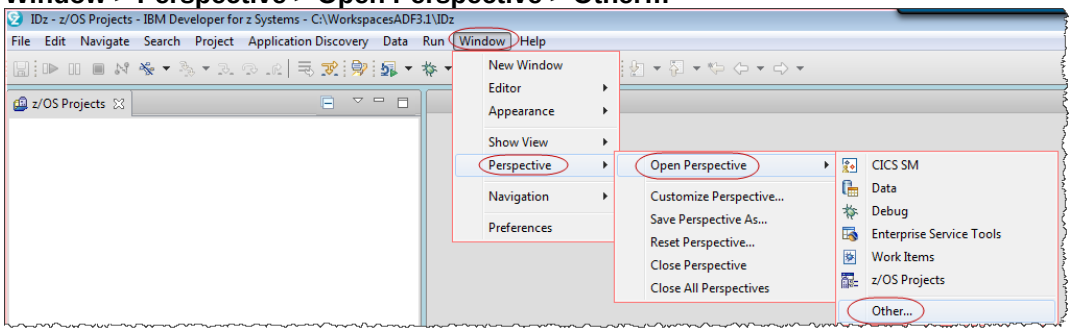
You will now create a *z/OS Connect EE Service* project and use the COBOL copybook to create a service. This copybook was identified as a candidate to invoke an API from the existing EGUI transaction.
To create the *z/OS Connect EE Service* you need to use the "*z/OS Connect Enterprise Edition*" perspective

___ 1.1 Create the z/OS Connect service project

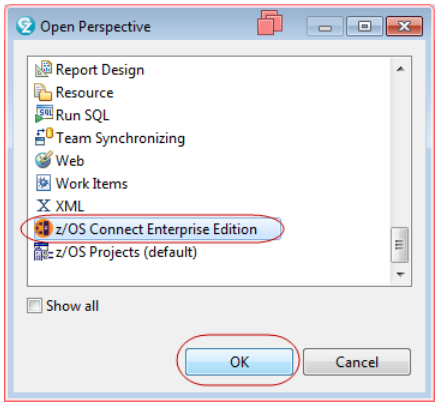
1.1.1 Start *IBM Developer for z Systems* if it is not already started,.
⏭ Double click on **IDz 14.1** icon

Be Patient, this first startup may take few minutes…

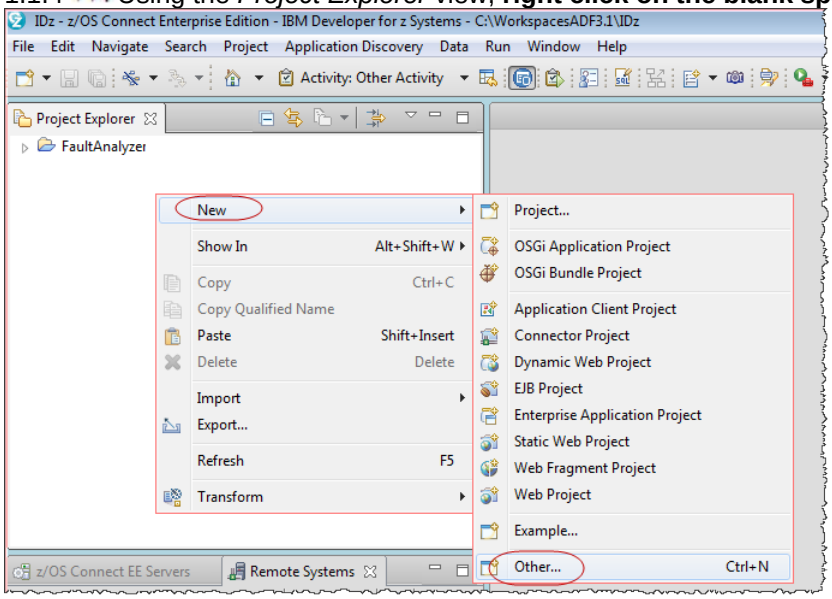
1.1.2. ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > Other…



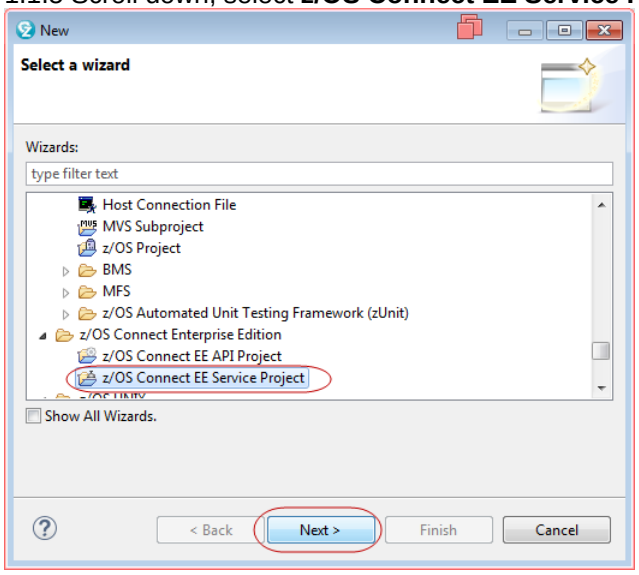
1.1.3 Select **z/OS Connect Enterprise Edition** and click **OK**



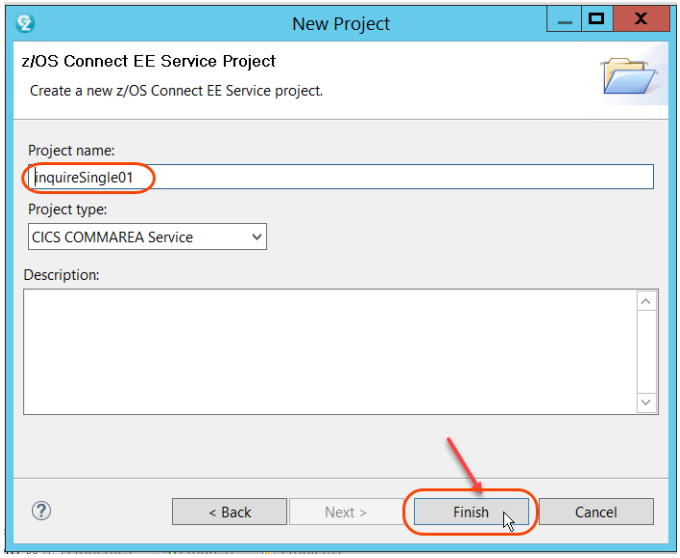
1.1.4 Using the *Project Explorer* view, **right click on the blank space** and select **New > Other..**



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



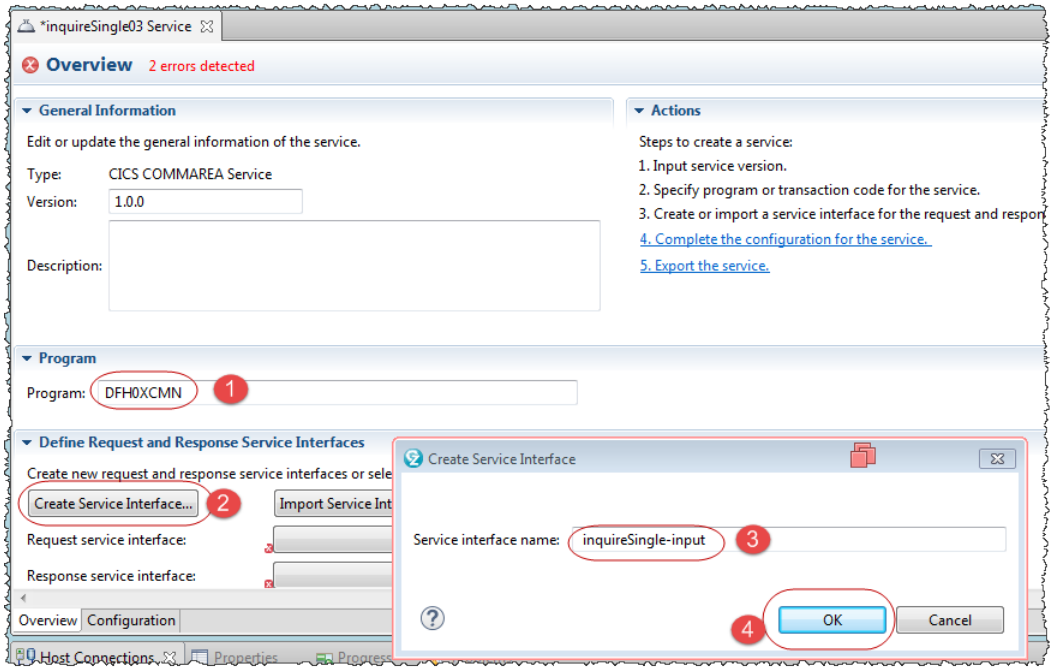
1.1.6 ⏭ Fill in the Project name, this name will be the name of the service.
Use **`inquireSingle01`**
We know from the Application Discovery lab that the project should be based on a CICS Commarea
⏭ Click **Finish**.



___ 1.2 Service configuration

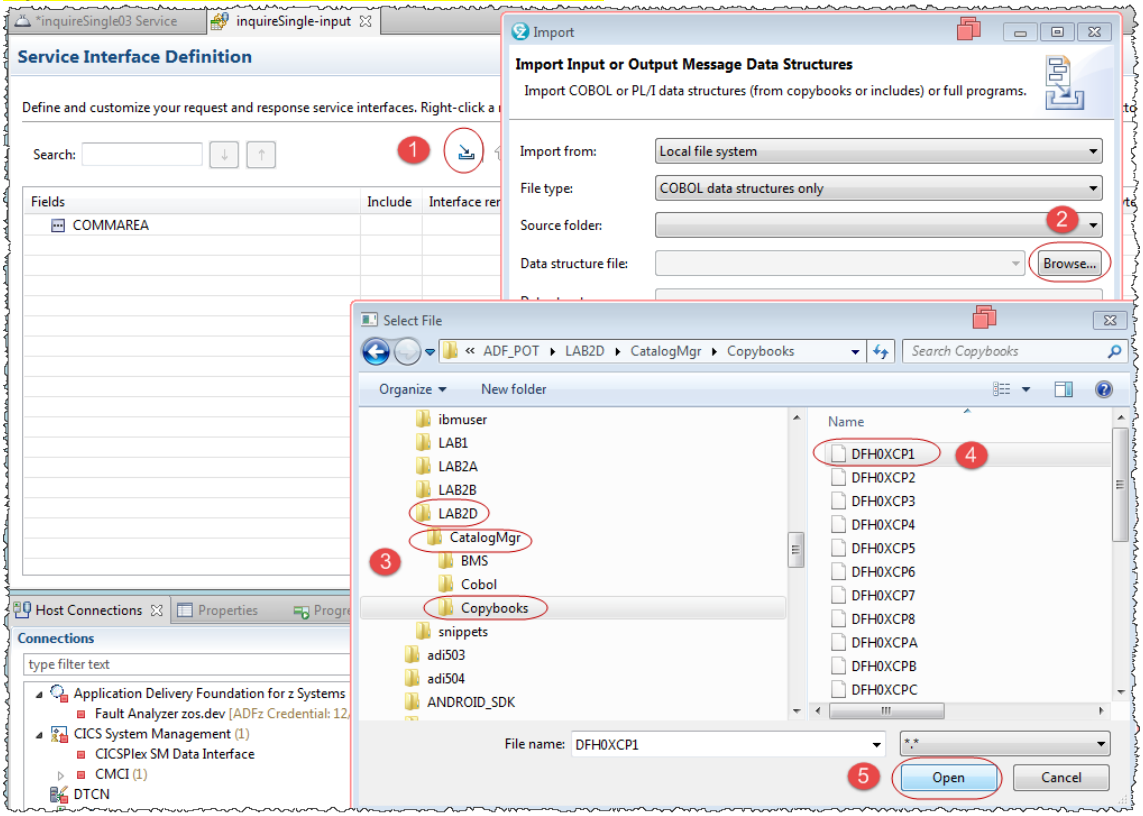
Once the project is created, there are 4 steps to perform: specify the CICS program to call, import copybook, define service interfaces and specify the IPIC connection reference

1.2.1 ⏭ Fill in the Program information.
We know from the Application Discovery lab that the program is **`DFH0XCMN`** (where 0 is number zero).
⏭ Click on **Create Service Interface...** and call it **`inquireSingle-input`** and click **OK**

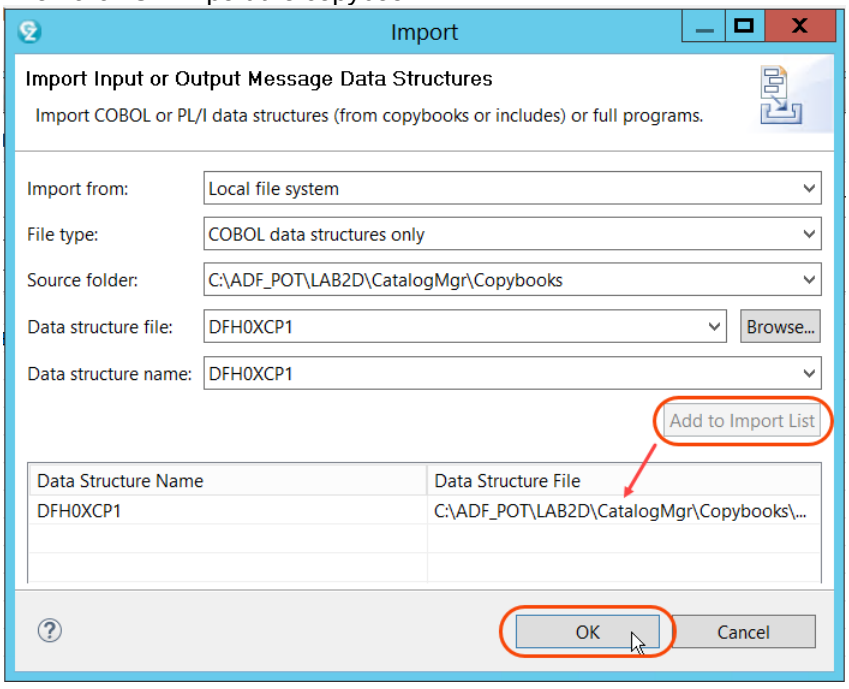


1.2.2 You will arrive in the Service Interface Definition window and you need to Import the copybook **DFH0XCP1**.

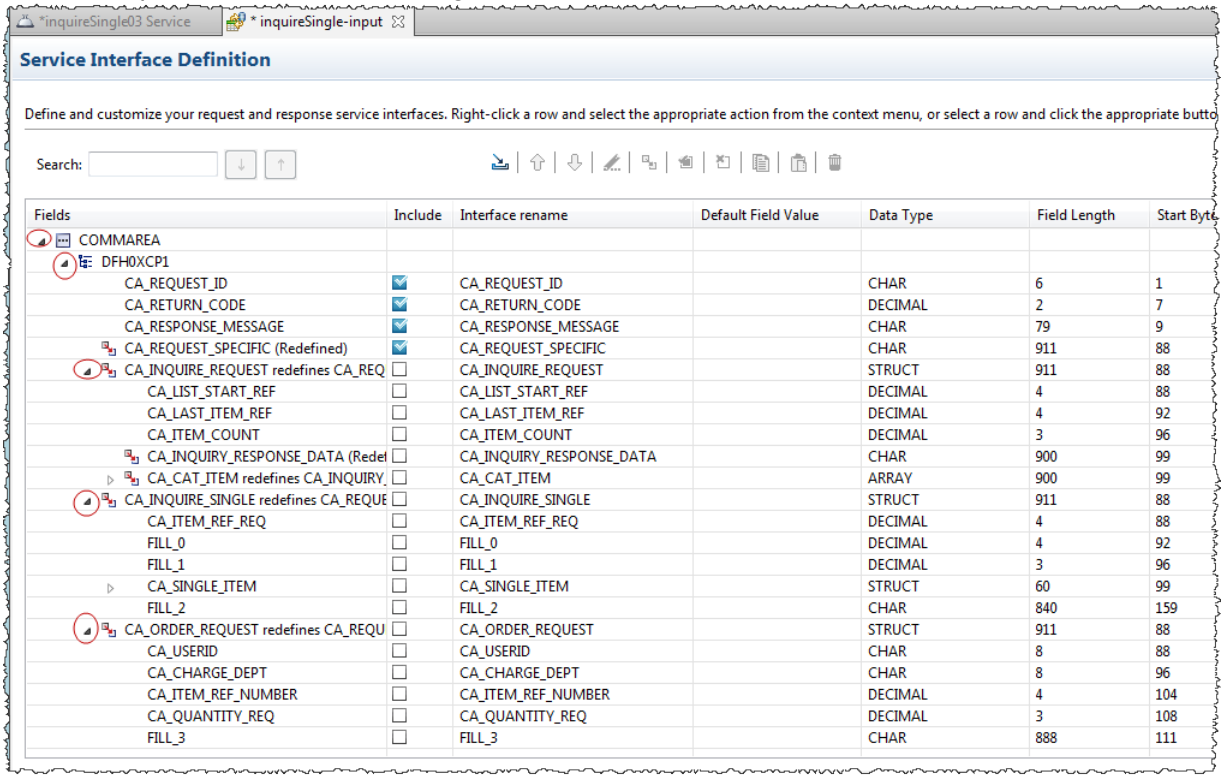
⏭. Click on icon and using the **Browse.**. button navigate to **`c:\ADF_POT\LAB2D\CatalogMgr\Copybooks\DFH0XCP1`** and click **Open.**
If you DO not find this folder, contact the instructor.



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



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

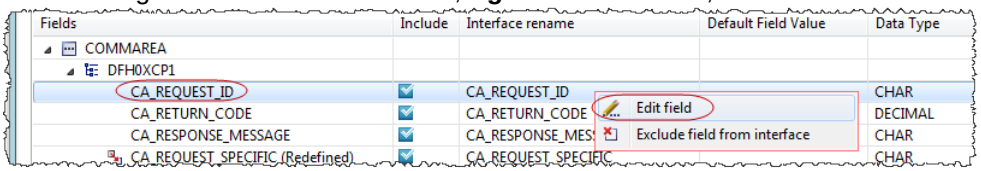


Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	☑	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	☑	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	☑	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefined)	☑	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines CA_REQ	☐	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_LIST_START_REF	☐	CA_LIST_START_REF		DECIMAL	4	88
CA_LAST_ITEM_REF	☐	CA_LAST_ITEM_REF		DECIMAL	4	92
CA_ITEM_COUNT	☐	CA_ITEM_COUNT		DECIMAL	3	96
CA_INQUIRY_RESPONSE_DATA (Redef	☐	CA_INQUIRY_RESPONSE_DATA		CHAR	900	99
CA_CAT_ITEM redefines CA_INQUIRY_	☐	CA_CAT_ITEM		ARRAY	900	99
CA_INQUIRE_SINGLE redefines CA_REQUE	☐	CA_INQUIRE_SINGLE		STRUCT	911	88
CA_ITEM_REF_REQ	☐	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	☐	FILL_0		DECIMAL	4	92
FILL_1	☐	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	☐	CA_SINGLE_ITEM		STRUCT	60	99
FILL_2	☐	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines CA_REQU	☐	CA_ORDER_REQUEST		STRUCT	911	88
CA_USERID	☐	CA_USERID		CHAR	8	88
CA_CHARGE_DEPT	☐	CA_CHARGE_DEPT		CHAR	8	96
CA_ITEM_REF_NUMBER	☐	CA_ITEM_REF_NUMBER		DECIMAL	4	104
CA_QUANTITY_REQ	☐	CA_QUANTITY_REQ		DECIMAL	3	108
FILL_3	☐	FILL_3		CHAR	888	111

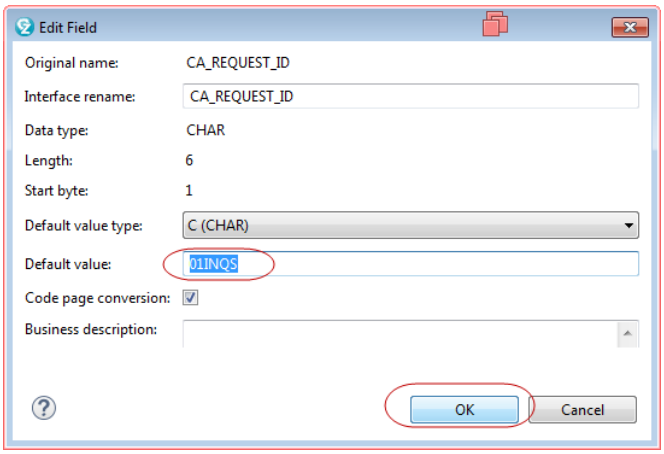
For the **inquireSingle-input** service, define a mapping that:

1.2.5 Excludes **CA_REQUEST_ID** and assign the default value **`01INQS`**
(Identified at the Application Discovery Lab)

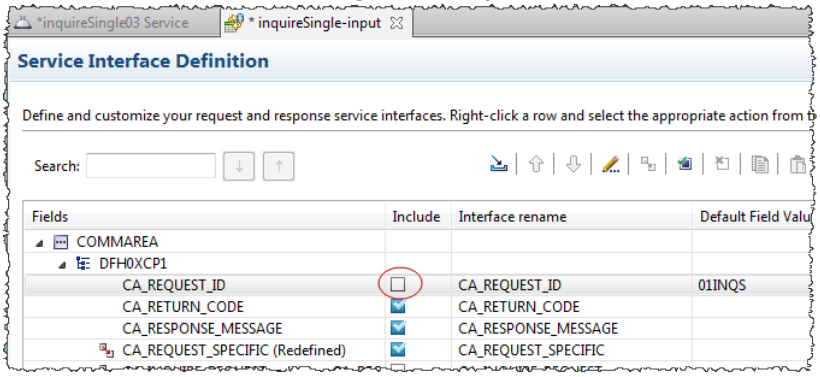
⏭ To assign a default value to a field, **right-click the field**, then choose **Edit field**,



1.2.6 ⏭ Specify **`01INQS`** as default value and click **OK**

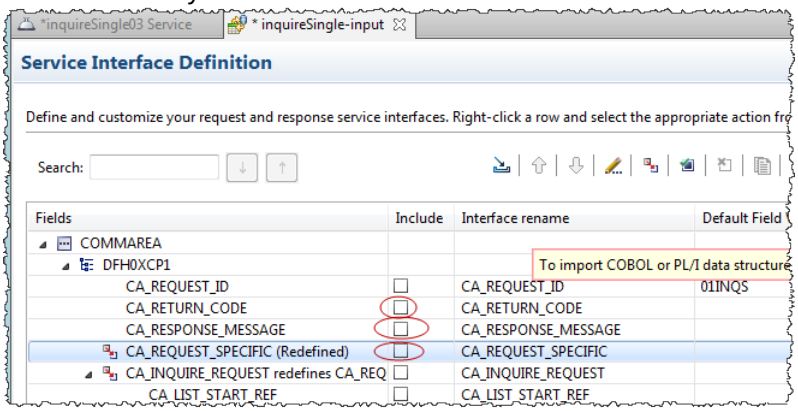


1.2.7 ⏭ To exclude **CA_REQUEST_ID** you need to **uncheck the include box.**



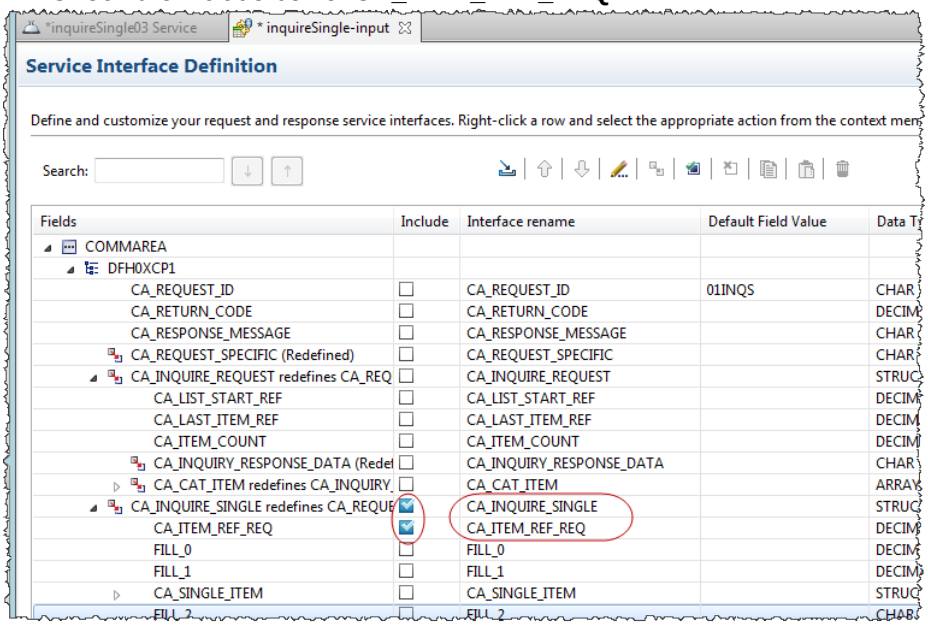
1.2.8 Excludes **CA_RETURN_CODE**, **CA_RESPONSE_MESSAGE** (these are response fields). Excludes **CA_REQUEST_SPECIFIC** (this field is redefined by CA_INQUIRE_SINGLE)

⏭ To exclude you need to **uncheck the include box**.



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

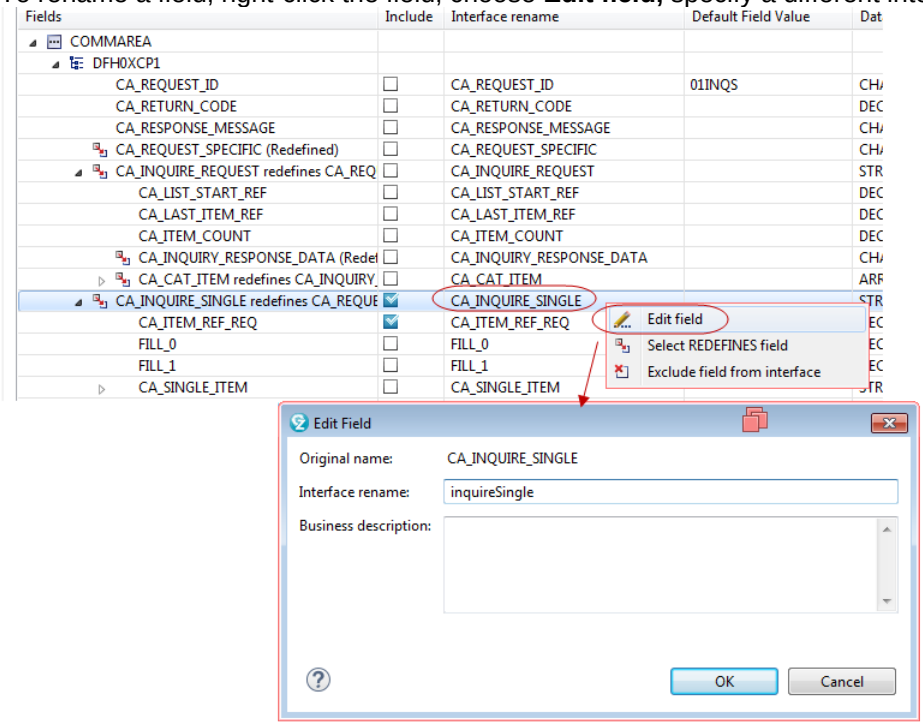
⏭ **Check** the include box of **CA_ITEM_REF_REQ** .



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

⏭ Rename the **CA_INQUIRE_SINGLE** as **`inquireSingle`**,

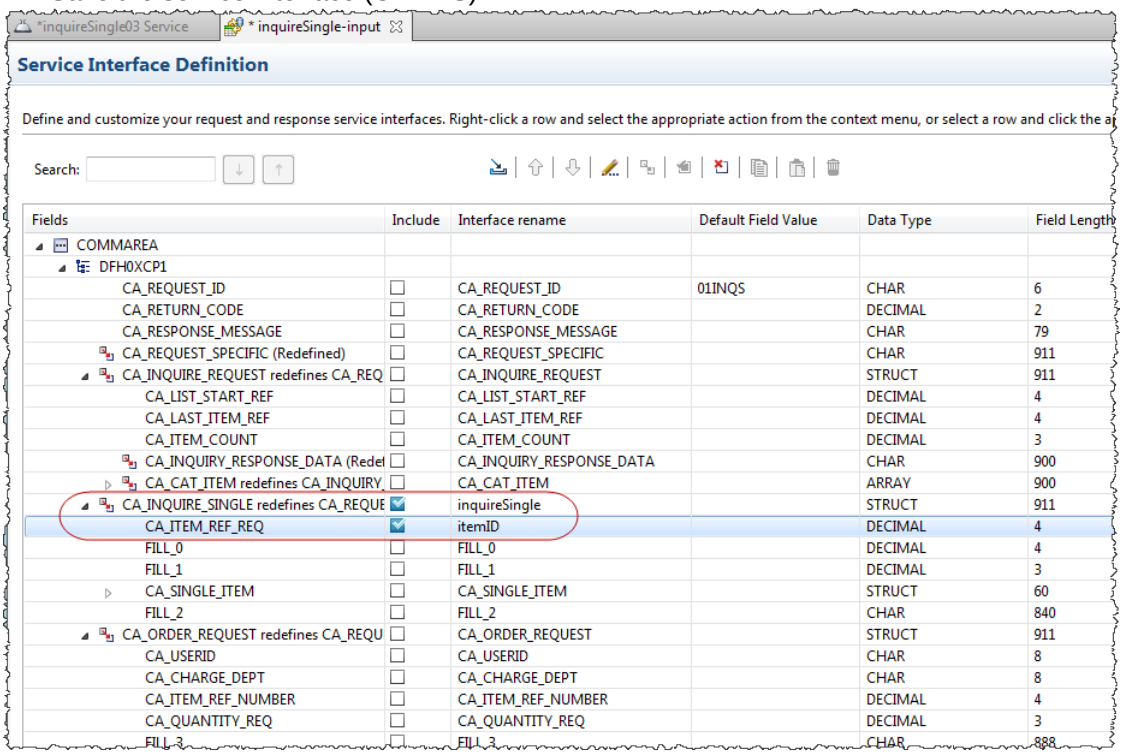
To rename a field, right-click the field, choose **Edit field,** specify a different interface rename and click **OK**.



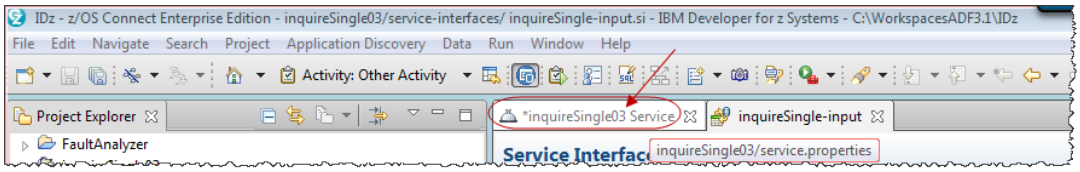
⏭ Rename the **CA_ITEM_REF_REQ** field as **`itemID`** (follow the lower/upper case letters)

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

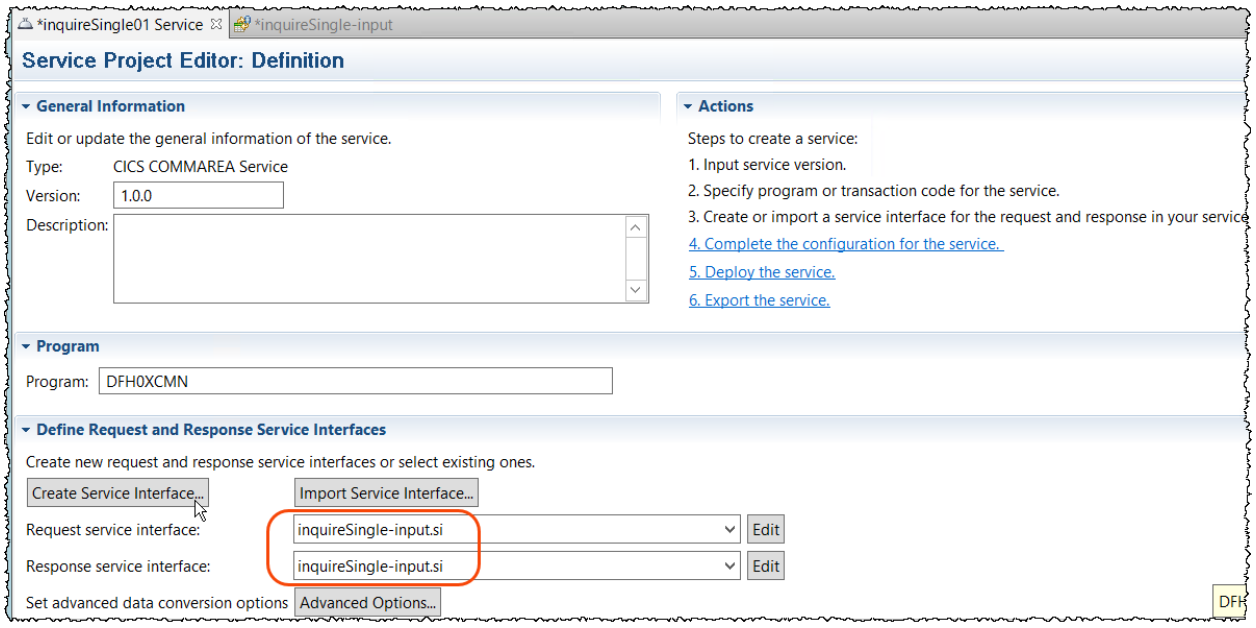
⏭ Save the service interface (**CTRL S**)"



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



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

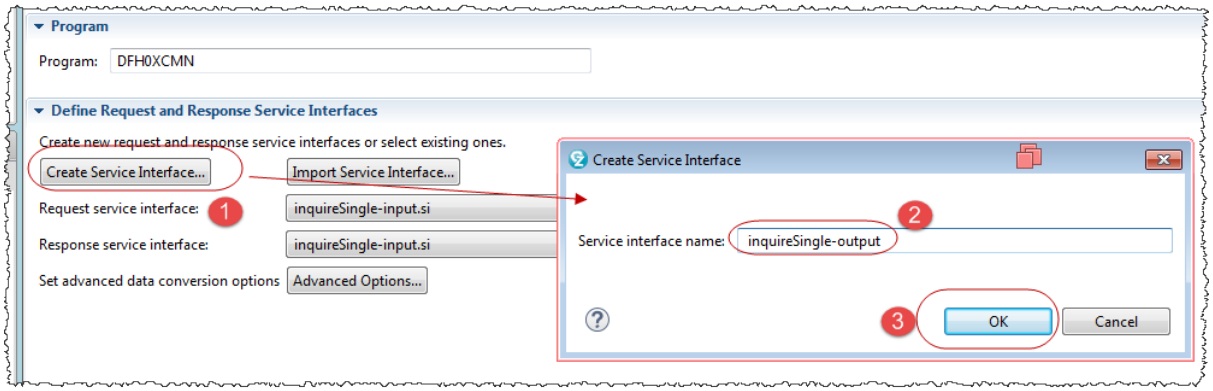


Let's create a new service interface for the output: **inquireSingle-output**.

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

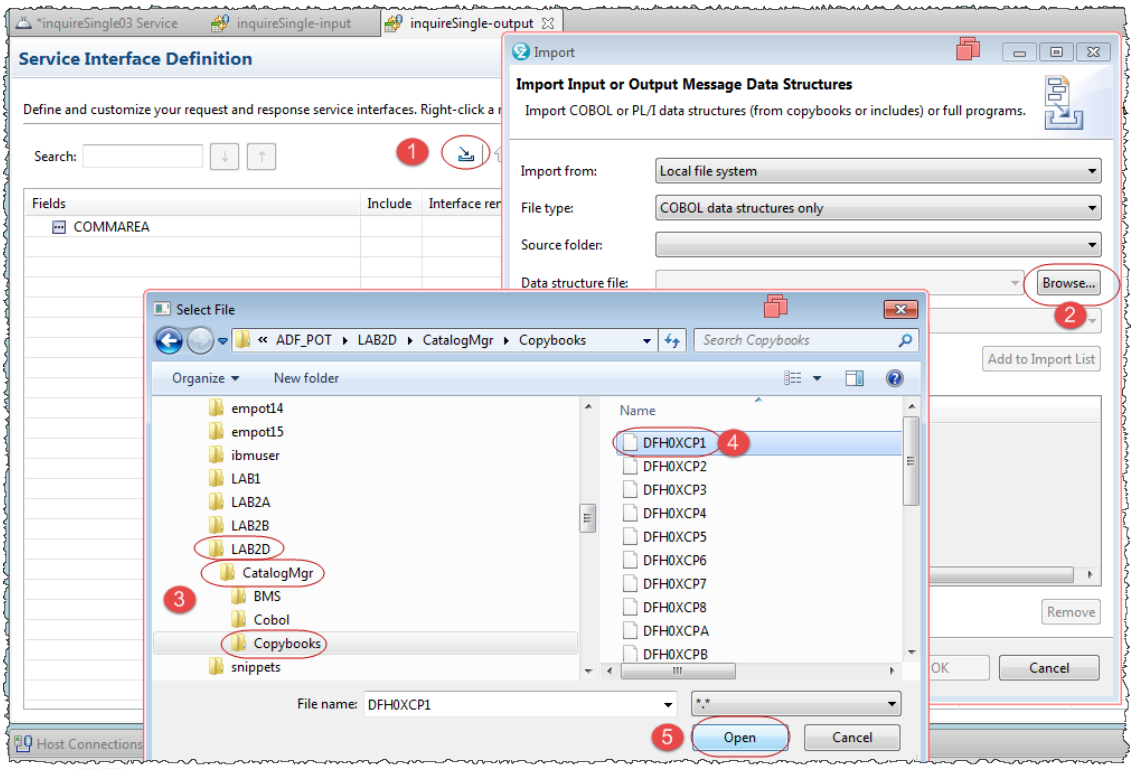
- Import the **DFH0XCP1** copybook
- Include and exclude fields
- Optionally rename exposed fields to give names that make more sense

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

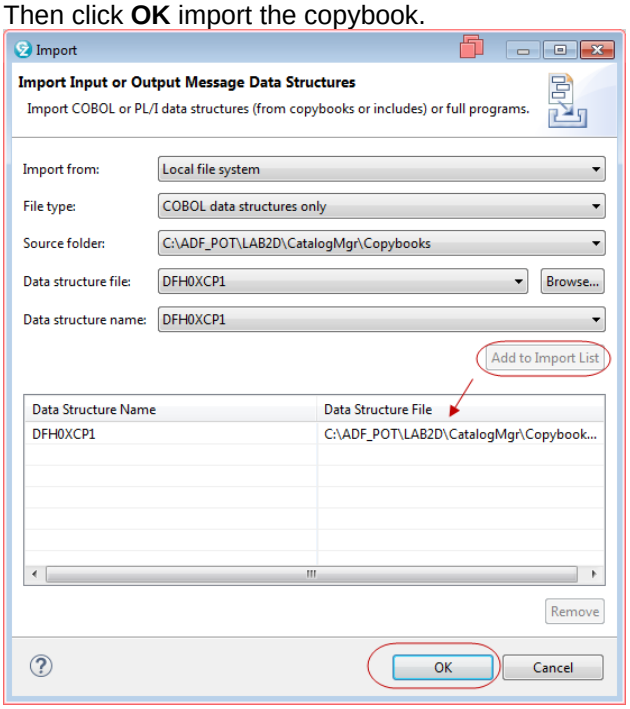


1.2.15 You will arrive in the Service Interface Definition window and you need to Import again the copybook **DFH0XCP1**.

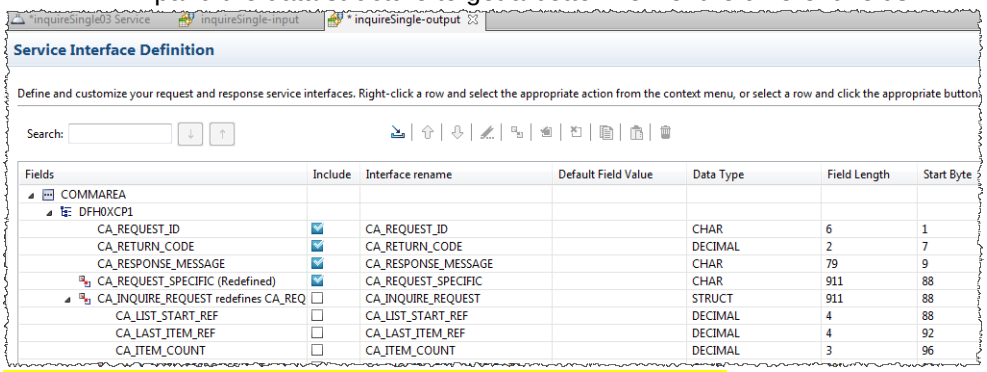
⏭. Click on icon  click on source folder to select the value already entered there and using the **Browse.**. button navigate to **`c:\ADF_POT\LAB2D\CatalogMgr\Copybooks\DFH0XCP1`** and click **Open.**



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



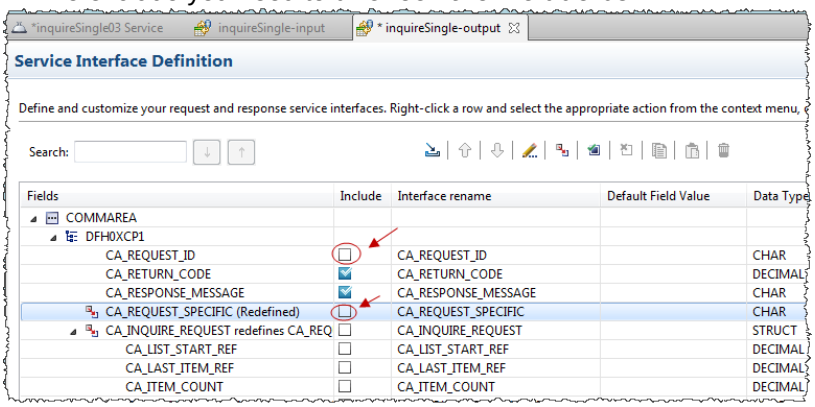
1.2.17 ⏭ Expand the data structure to get a better view of the different fields



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

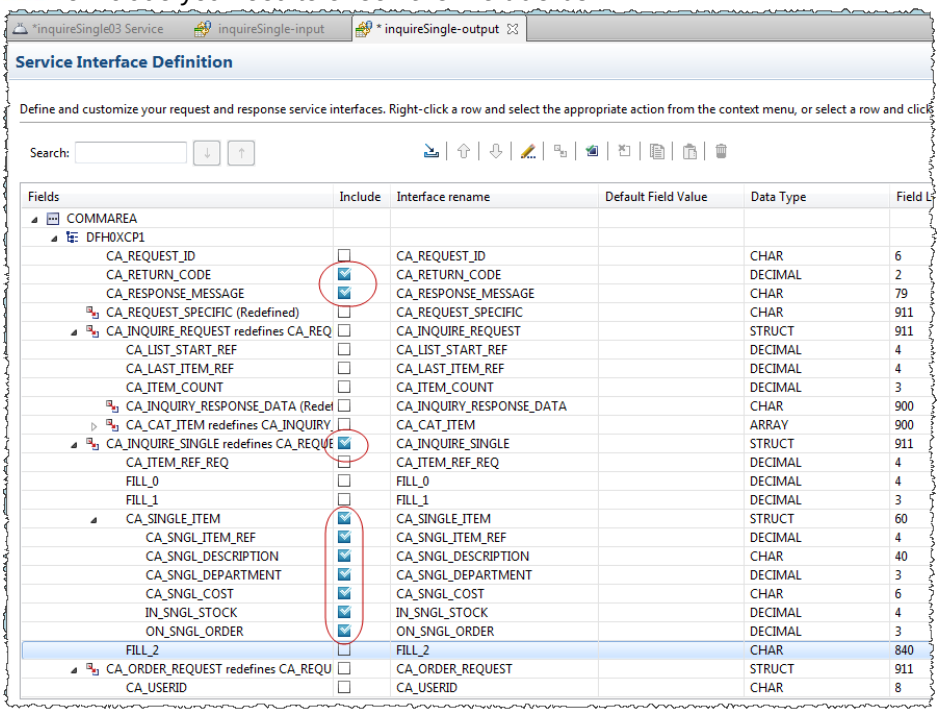
1.2.18 Excludes **CA_REQUEST_ID** and **CA_REQUEST_SPECIFIC**

⏭ To exclude you need to **uncheck the include box**.



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

⏭ To include you need to **check the include box**.

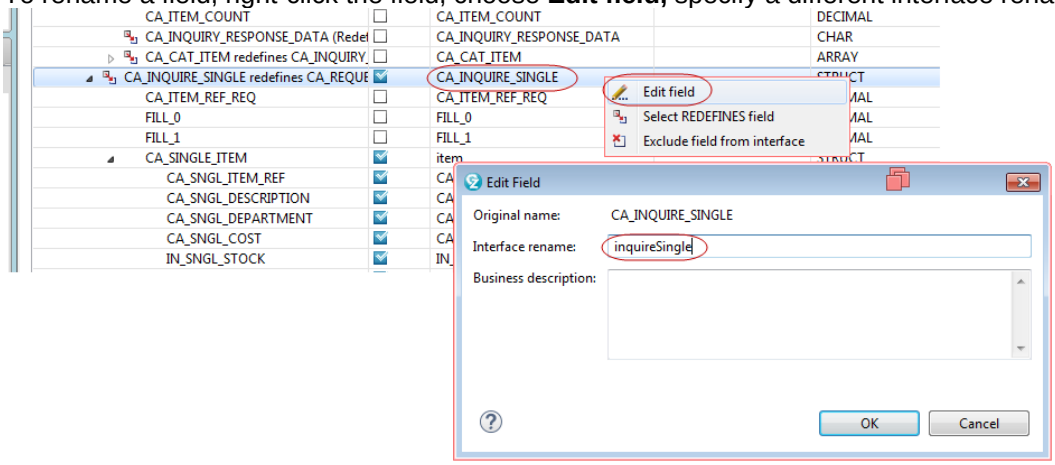


1.2.20 The best practice is to rename the fields to better expose the service interface..
But if you are running late or don't want to spend time here just jump to 1.2.22

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

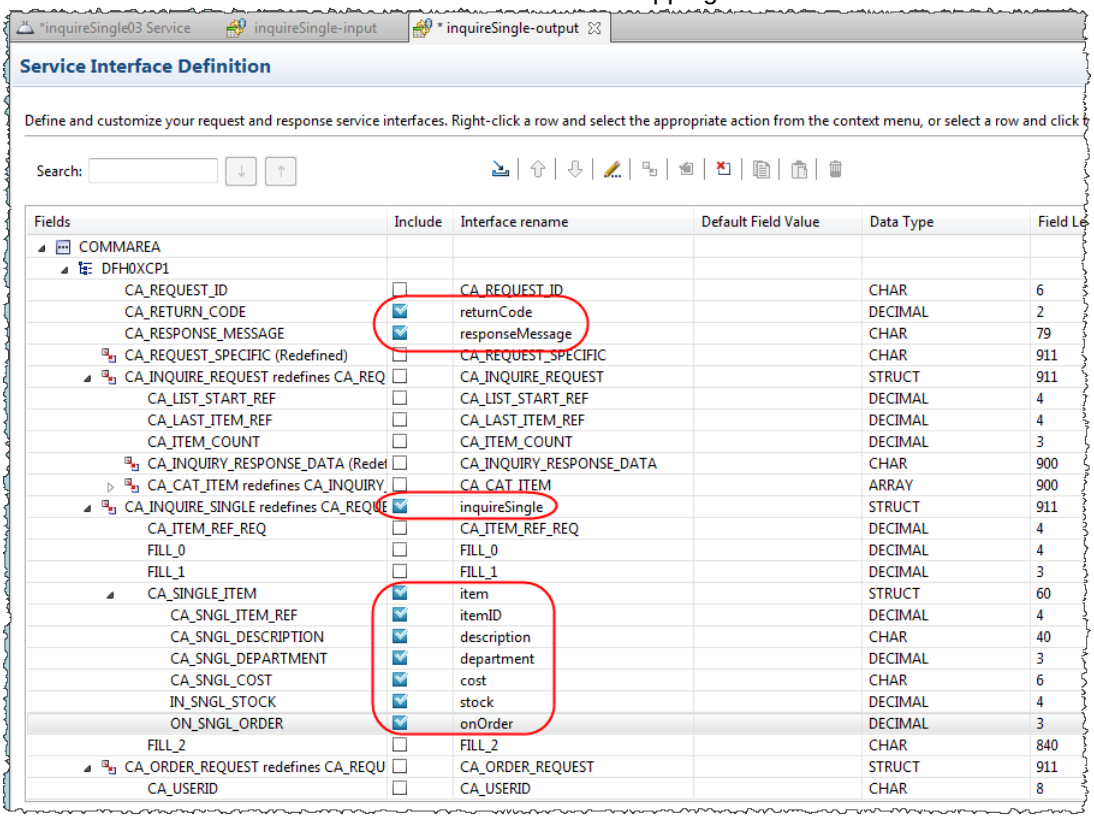
⏭ Rename the **CA_INQUIRE_SINGLE** as **`inquireSingle`**,

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



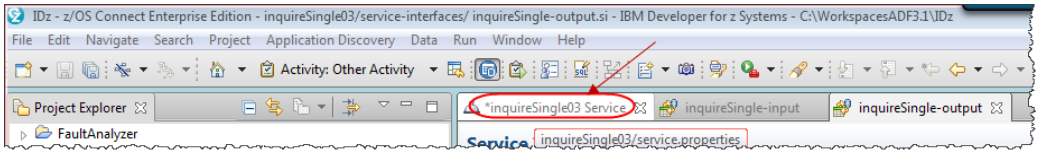
1.2.21 ⏭ (Optional) Rename the **CA_RETURN_CODE** as **returnCode, CA_RESPONSE_MESSAGE** as **responseMessage CA_SINGLE_ITEM** as **item, CA_SNGL_ITEM_REF** as **itemID, CA_SNGL_ITEM_DESCRIPTION** as **description, CA_SNGL_ITEM_DEPARTMENT** as **department, CA_SNGL_COST** as **cost, IN_SNGL_STOCK** as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

⏭ Your service interface should look similar to the mapping below

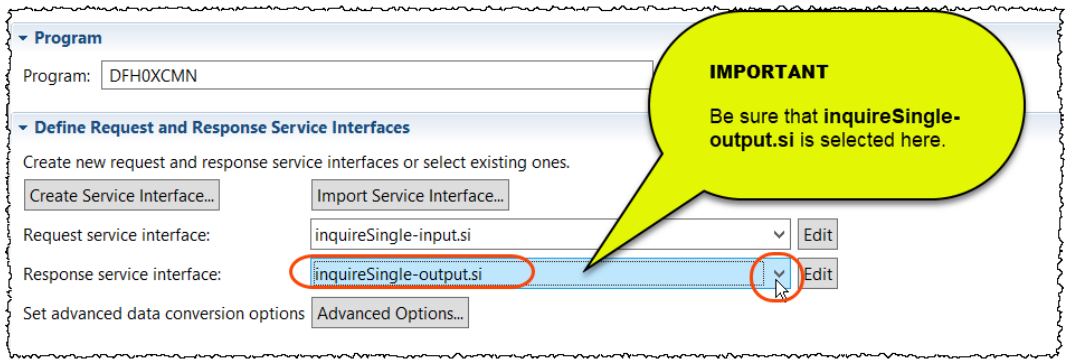


1.2.22 Save the **`inquireSingle-output`** service interface (**CTRL S**)"

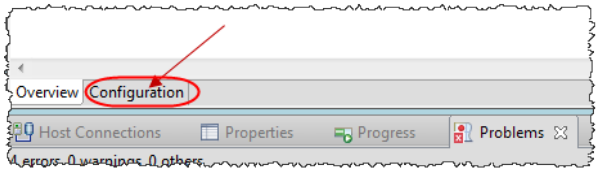
1.2.23 Go back to the **inquireSingle01** Service tab by clicking on it



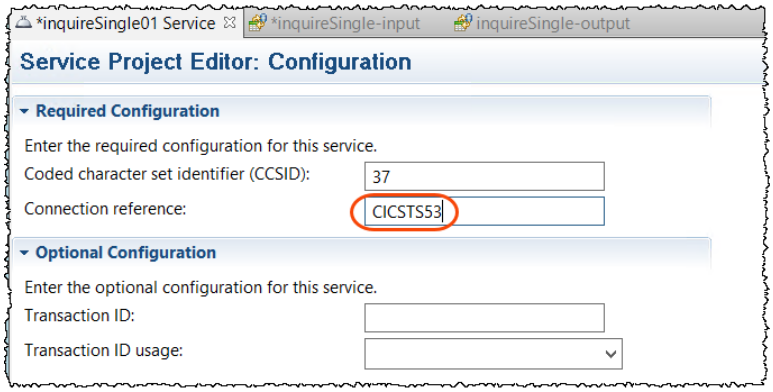
1.2.24 Click on the second drop down ("*Response service interface*") and select **inquireSingle-output.si**



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



1.2.26 Type **`CICSTS53`** for the *Connection reference*
This connection name is a logical name that represents the IPIC connection to a CICS region.



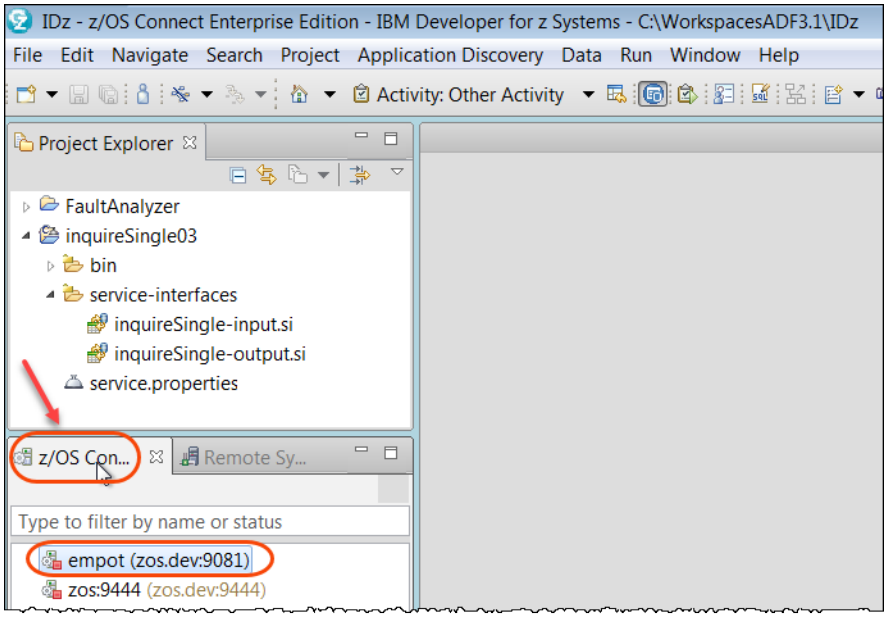
1.2.27 Save the service interface (**CTRL S**)"

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

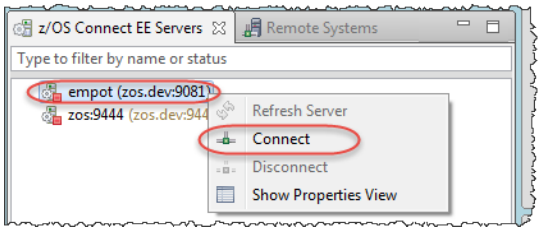
___ 1.3 Connecting to a z/OS Connect EE server

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

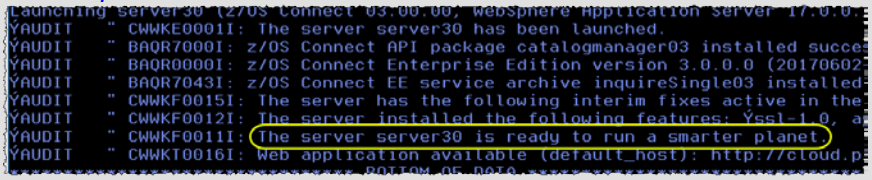
1.3.1 Still using the *z/OS Connect Enterprise Edition* perspective click on **z/OS Connect EE Servers** view. You should see the connection **zos.dev:9081** already defined



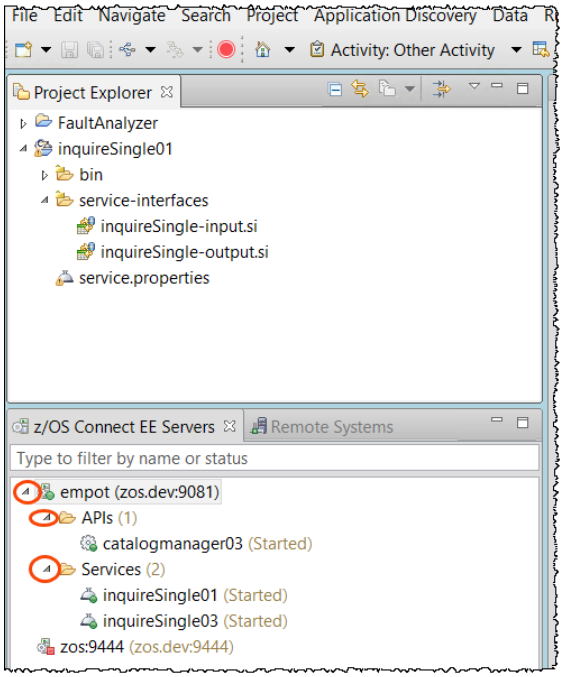
1.3.2 Right click **empot(zos.dev:9081)** and select **Connect**.



In case of connection errors…
If you have errors during the connection it could be because z/OS connect is not running. Call the instructor.
Instructor: *use the z/OS master console or SDSF DA and verify that you have the* ***BBGZANGL*** *and* ***BAQSTRT*** *started tasks running.*
If not: issue a ***S BBGZANGL****, then* ***S BAQSTRT***
For BAQSTRT, make sure you see the message "The server server1 is ready to run a smarter planet



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

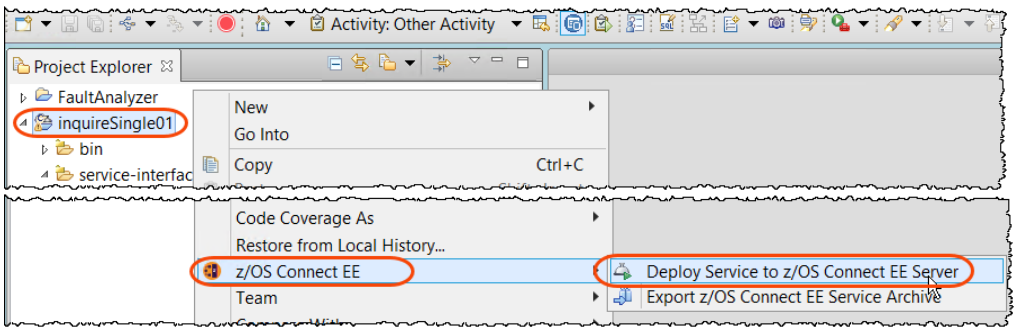


___ 1.4 Deploying the Service to z/OS Connect

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

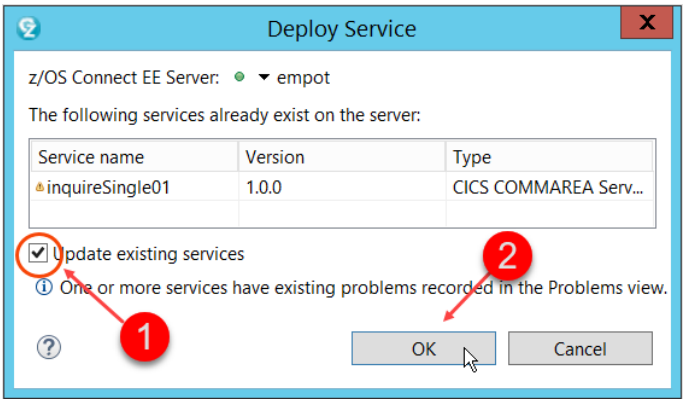
Automatic Deploy versus Manual deploy
Depending on the level of the **z/OS Connect EE V3** installed you could do the Automatic deploy.

1.4.1 We will deploy the service to the z/OS Connect EE server.
⏭ Using *Project Explorer* view, right-click on the project **inquireSingle01**, then **z/OS Connect EE > Deploy Service to z/OS Connect EE Server.**

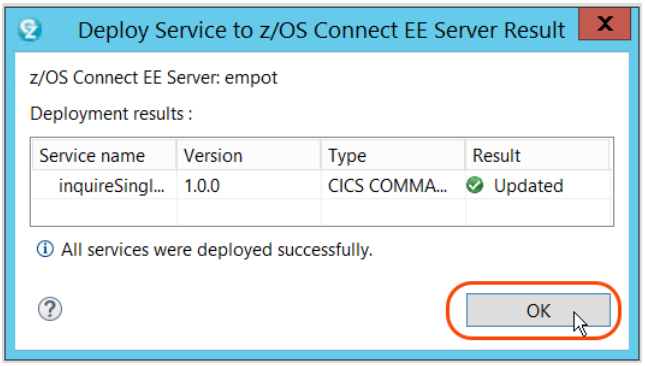


1.4.2 You will get the following *Deploy Service* dialog.

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

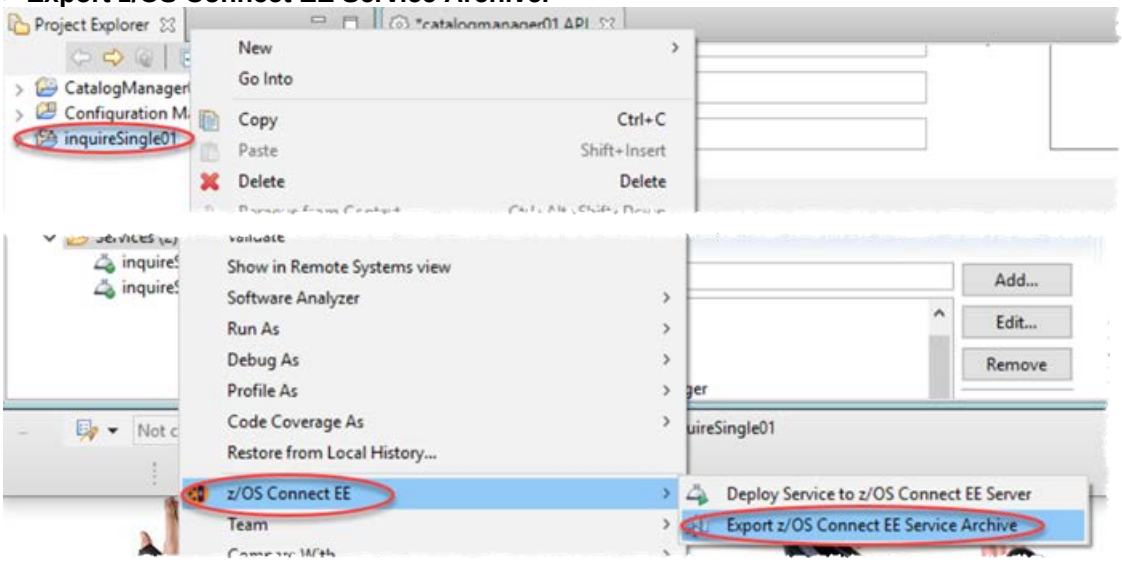


1.4.3 You will get the *Deploy Service … Result* dialog. The result could be “Created” or “Updated”.
⏭ Click **OK**.

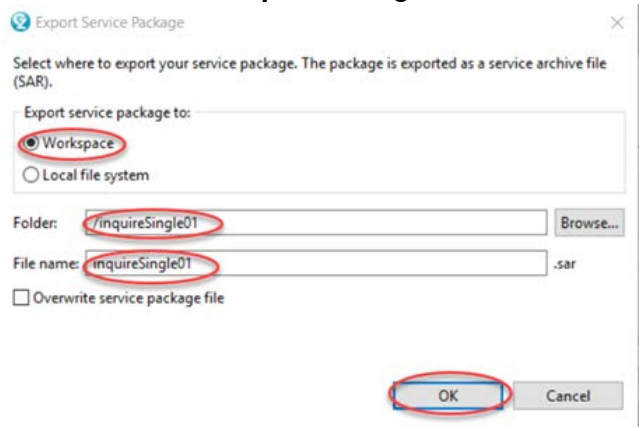


1.4.4 You will now export the service archive file for use in the next section.

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



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

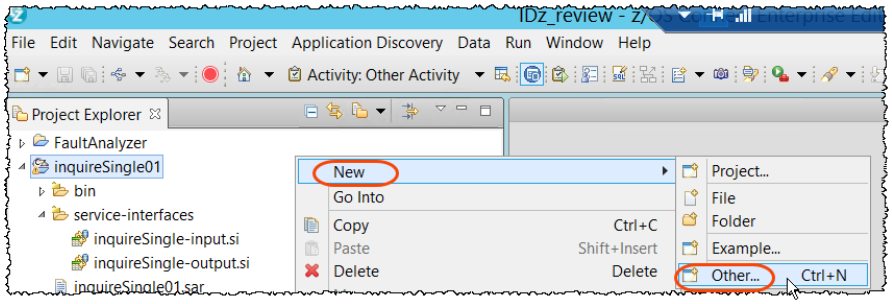


Section 2 – Create and configure a z/OS Connect API Project

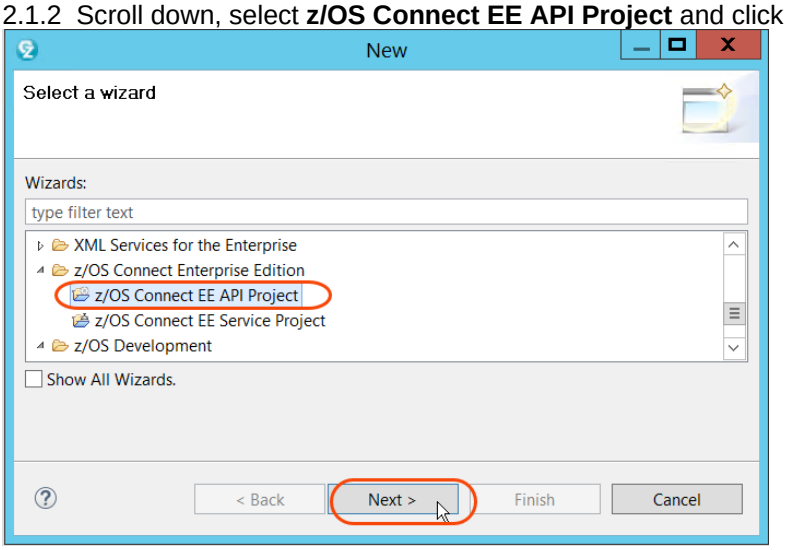
In this step, you will create and API to invoke a z/OS Connect Service created before.

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

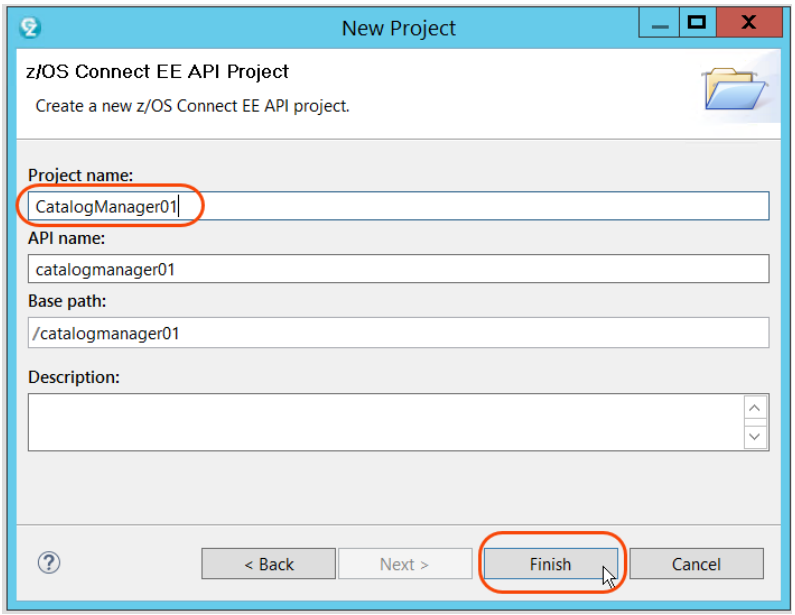
2.1.1. ⏭ Still under *z/OS Connect Enterprise Edition* perspective and using the *Project Explorer* view, **right click on the blank space** and select **New > Other..**



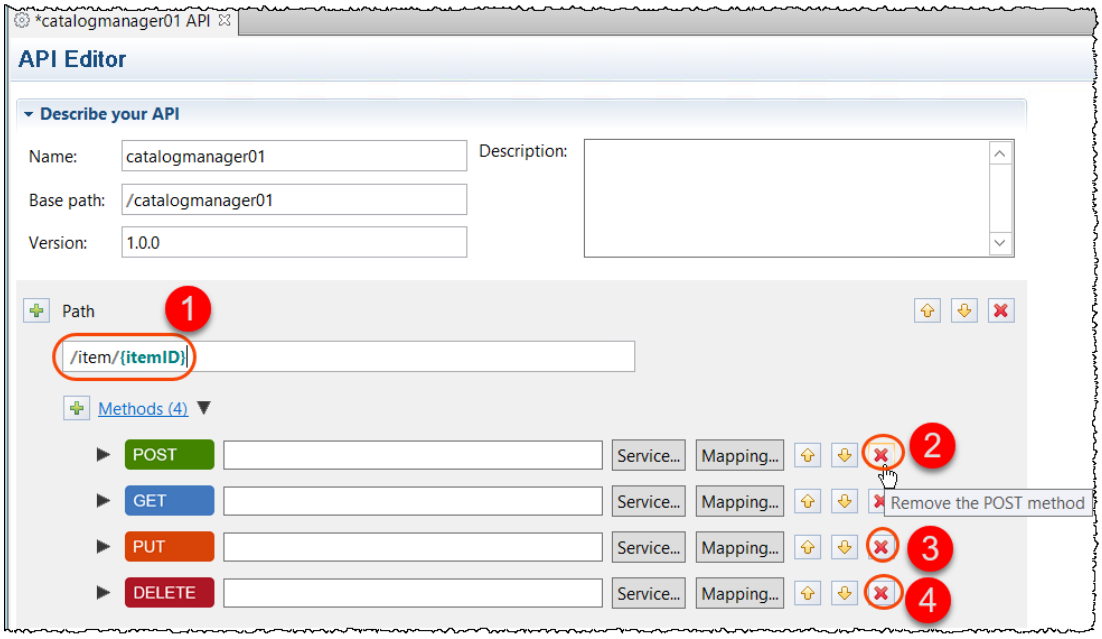
2.1.2 Scroll down, select **z/OS Connect EE API Project** and click **Next**



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

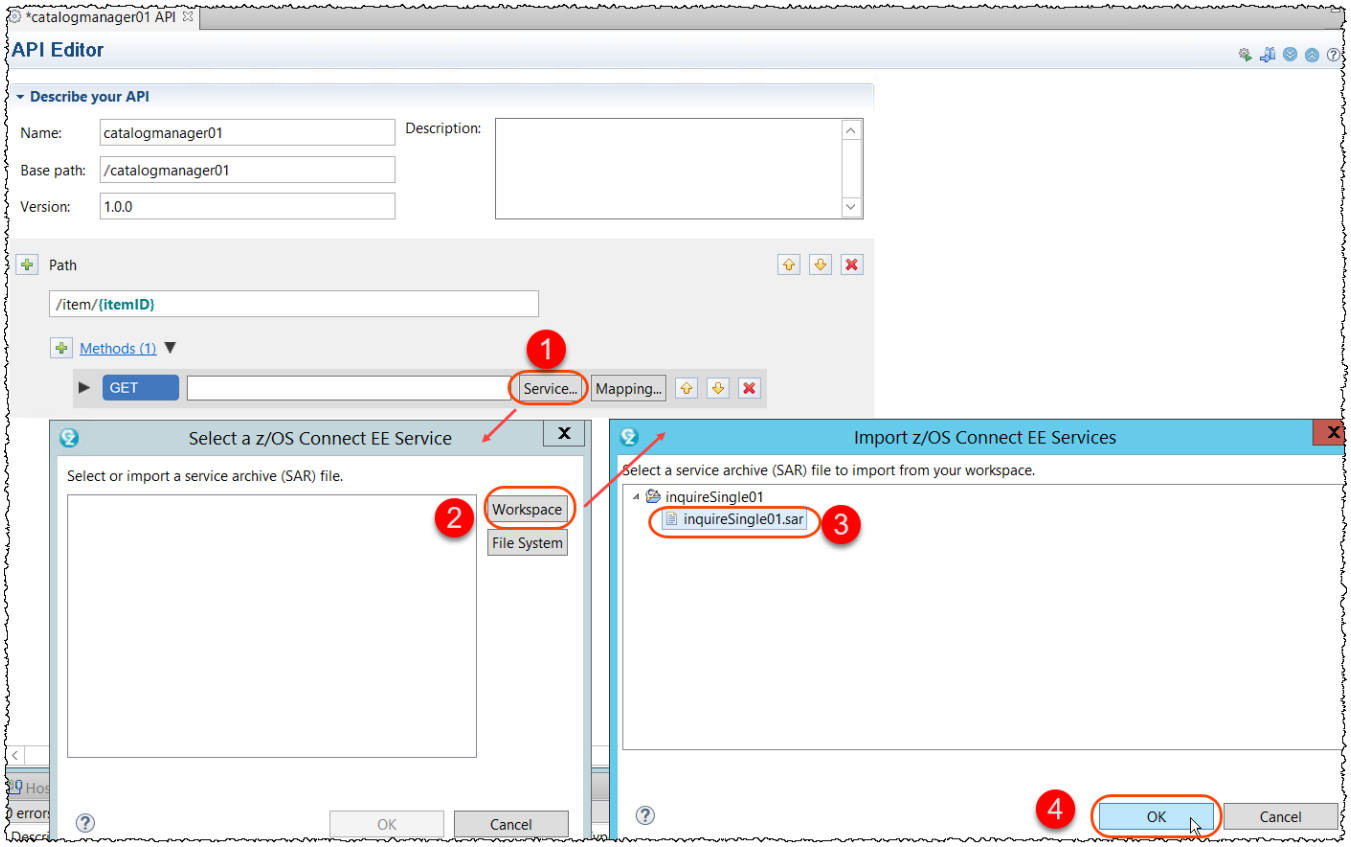


2.1.4 You will arrive in the z/OS Connect EE API Editor
⏭ in **Path** specify **`/item/{itemID}`**
Tip: (itemID was the interface renamed on step 1.2.10)
⏭ You want only GET a value from the service. Use the icon ✖ to delete the methods **POST**, **PUT** and **DELETE**



2.1.5 ⏭ Click **OK** for the dialogs confirming the delete.

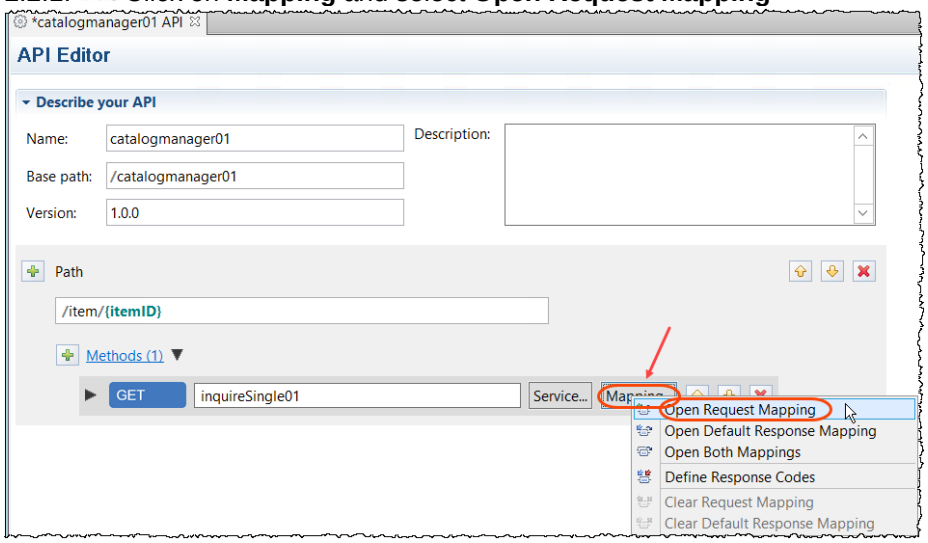
2.1.6 ⏭ Click on **Service**… button and on *Select a z/OS Connect EE Service* click **Workspace** button. Navigate to your Service Project and find the SAR file (**inquireSingle01.sar**) that you exported before and ⏭ Click **OK** three times.



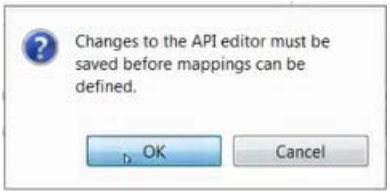
___ 2.2 Mapping the API to the service

Now we need to do the mappings

2.2.1. ⏭ Click on **Mapping** and select **Open Request Mapping**



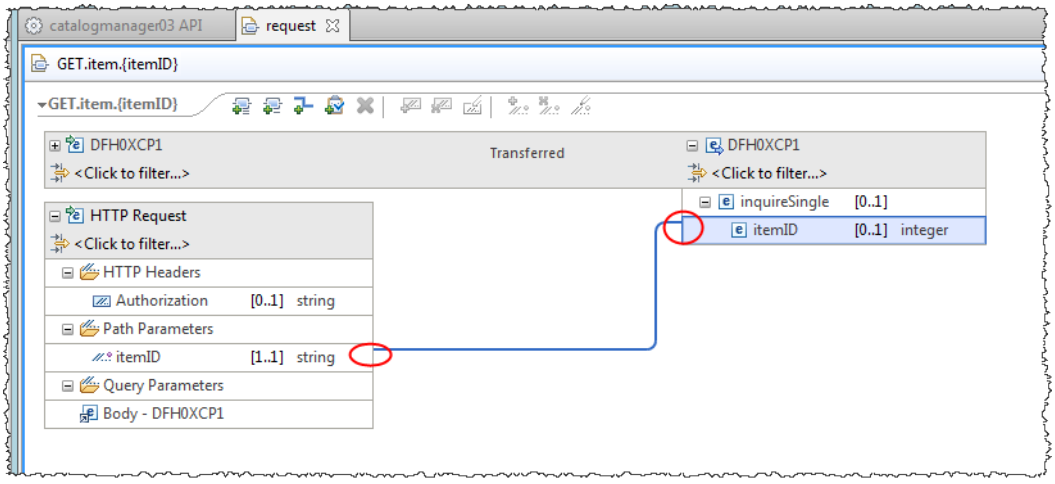
2.2.2 A dialog will ask for save the changes. Click **OK** to save it..



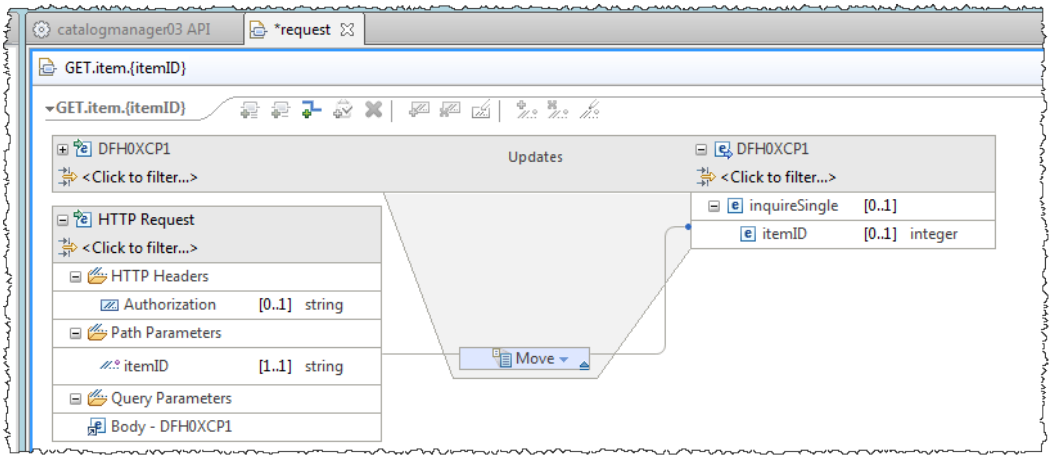
2.2.3 The Mapping editor opens. s

Expand **inquireSingle** on right.

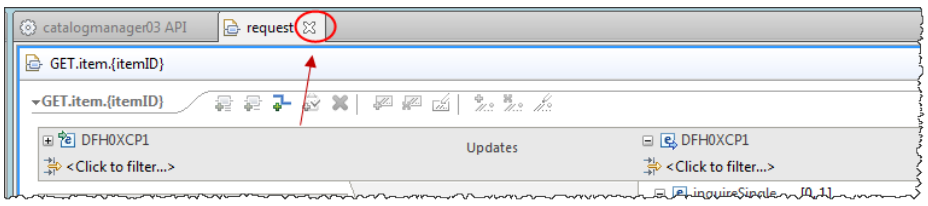
Use the mouse to drag **itemID** and drop to **itemID** under **Path Parameters**



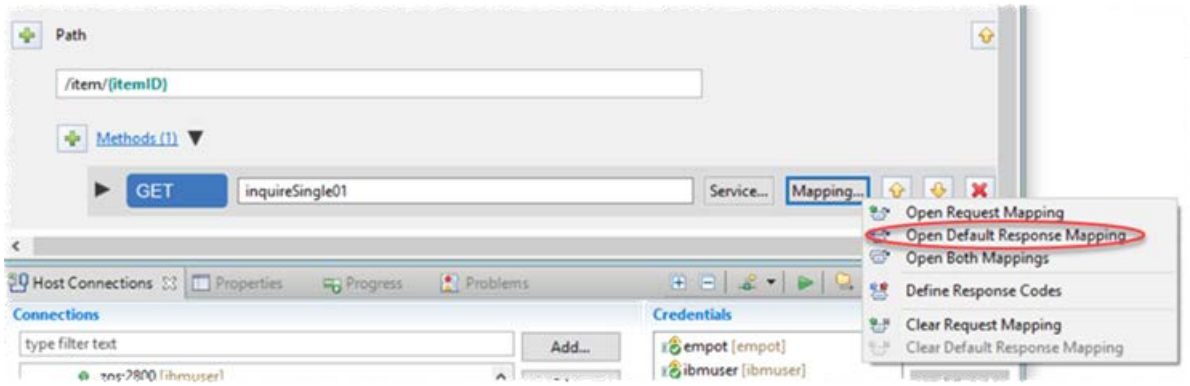
It shows as:



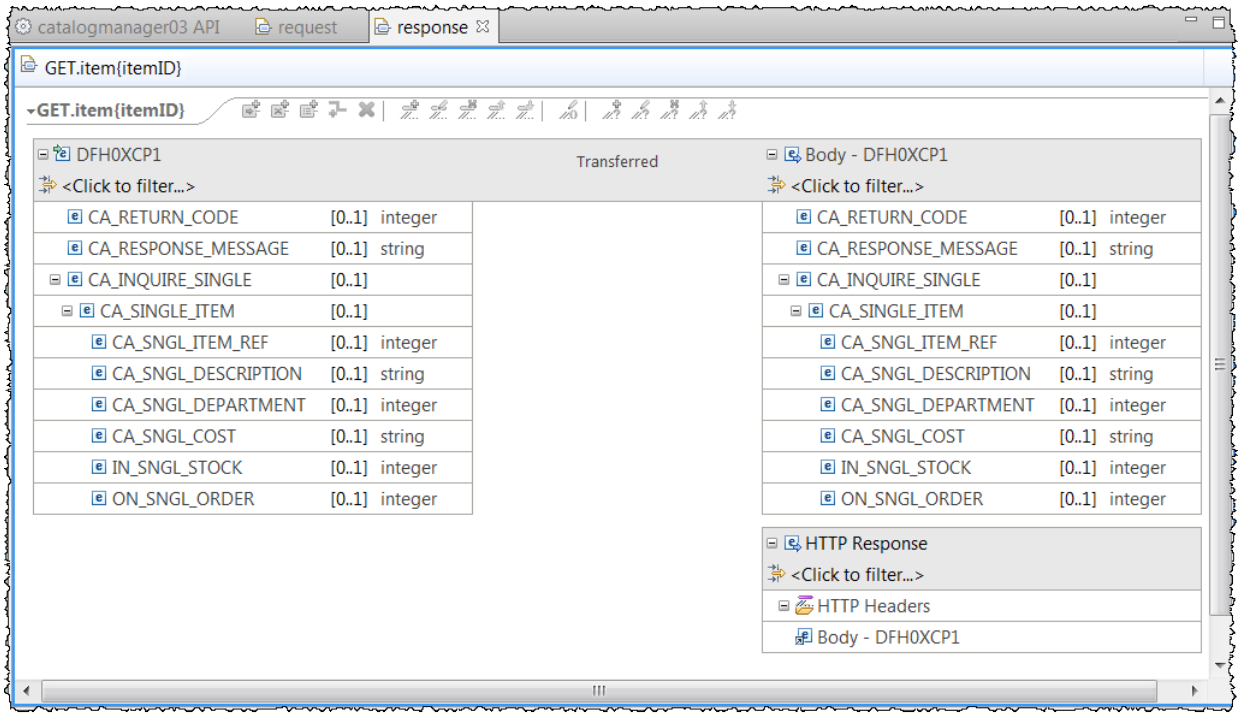
2.2.4 Use **Ctrl + S to** save this mapping and **close** this mapping editor



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



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



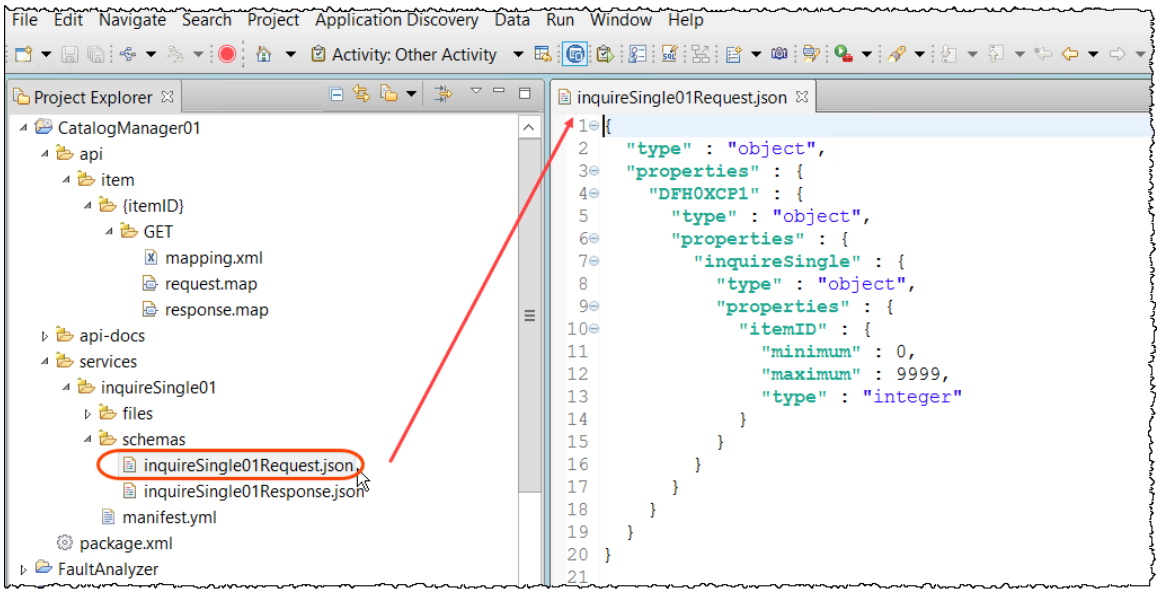
2.2.7 Use **Ctrl + Shift + F4** to close all editors

Section 3 – Deploy the API

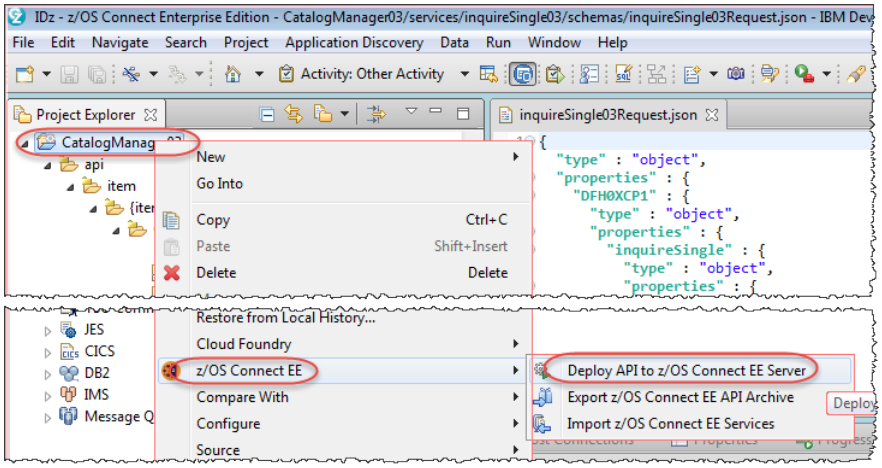
In this step, you would deploy the API to z/OS Connect EE Server. Your API may only contain a single path, but for an initial experience with the API Editor and the APIs deployment, it is sufficient.

___ 3.1 Deploy API to z/OS

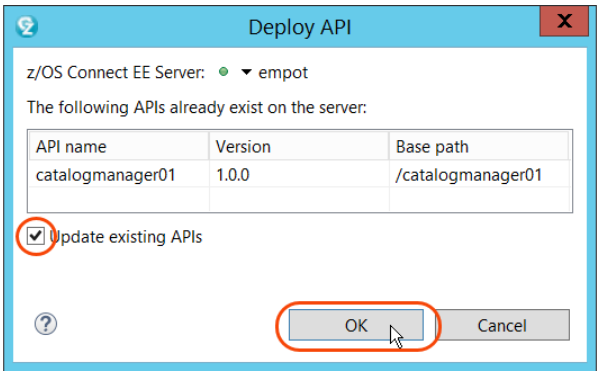
3.1.1. Still under *z/OS Connect Enterprise Edition* perspective and using the *Project Explorer* view, fully expand the **CatalogManager01** project and double click on **inquireSingle01Request.json.**
You can see the json request created



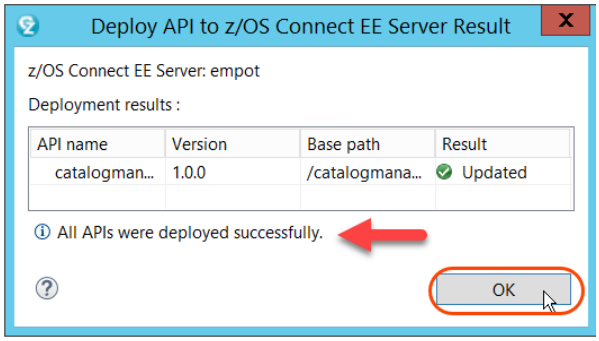
3.1.2. To deploy this API, right click on **CatalogManager01** and select **z/OS Connect EE -> Deploy API to z/OS Connect EE Server**



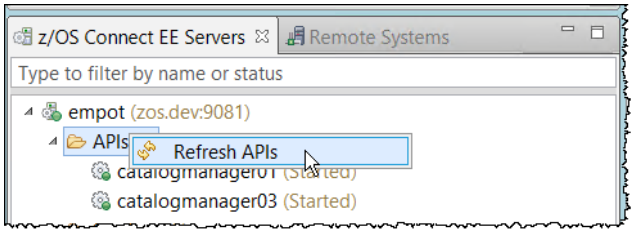
3.1.3 Select Update existing APIs and click **OK**



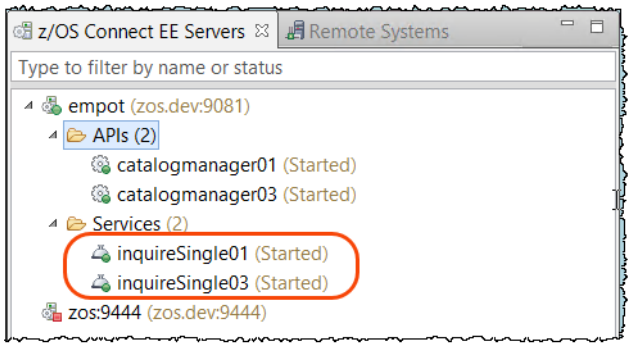
3.1.4 Once deployed, the result is displayed
Check that "*All APIs were deployed successfully*" then click on **OK**:



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



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

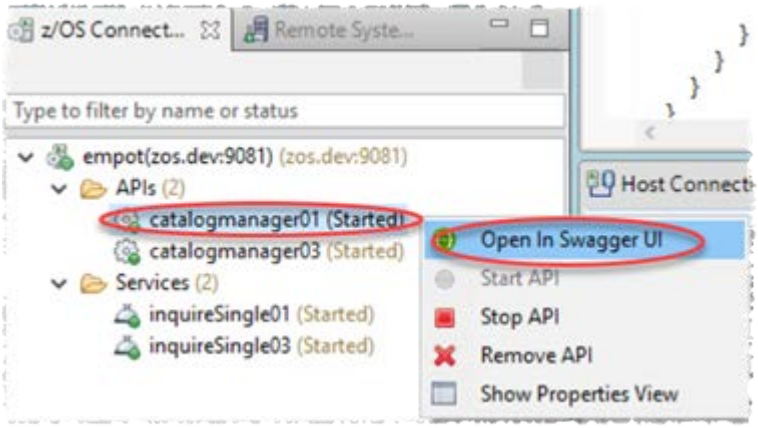


Section 4 – Testing deployed API using Swagger

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

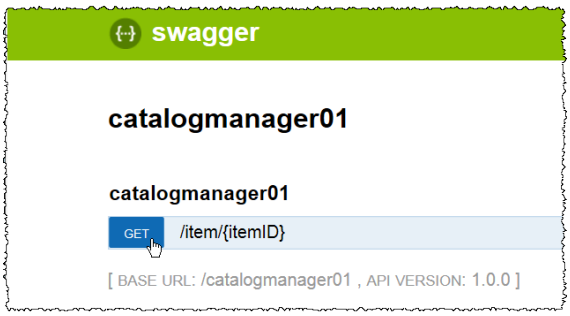
___ 4.1 Invoking Swagger to test the API

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

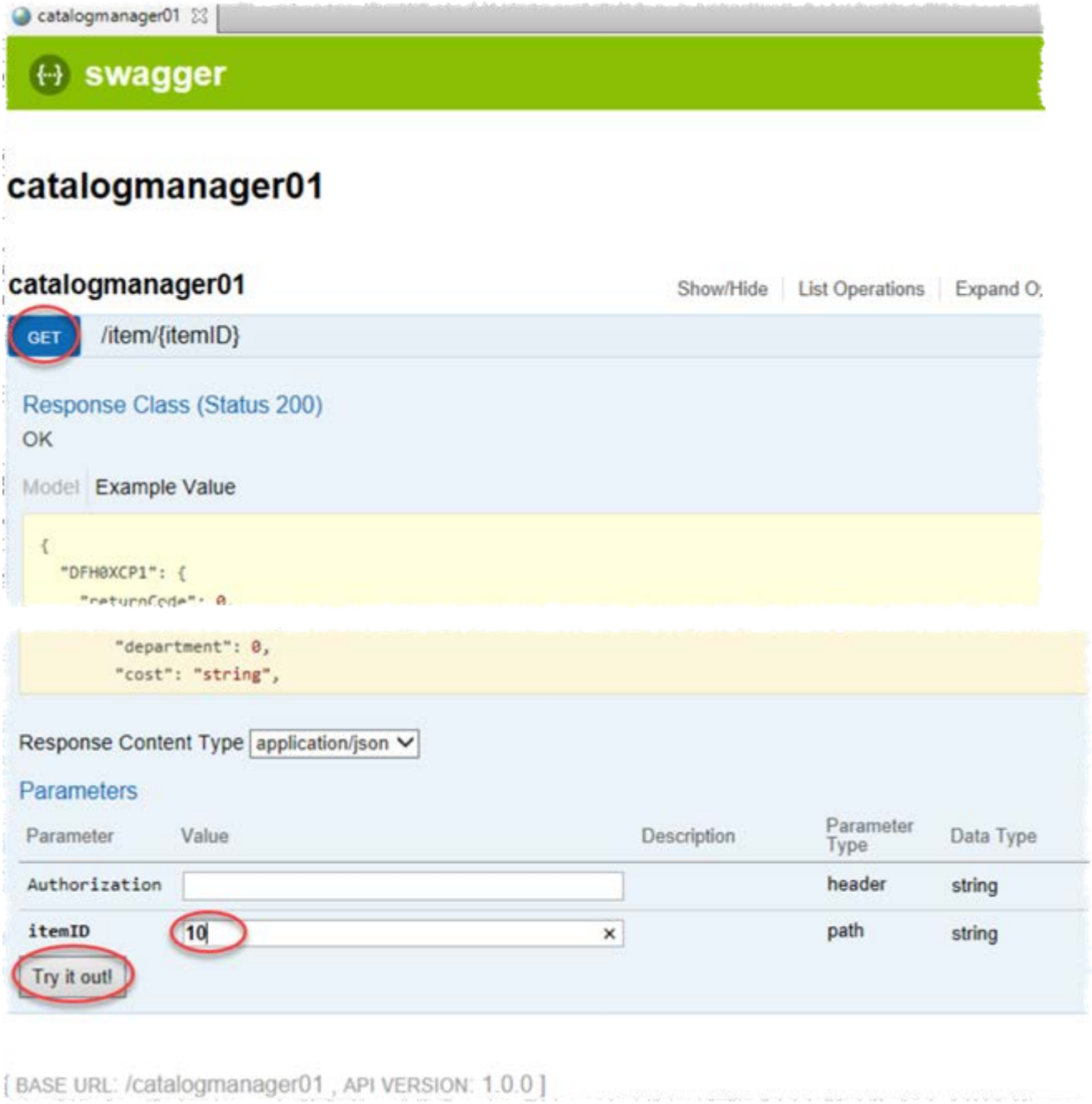


4.1.2. It will open the Swagger UI inside IDz.
⏭ Click on **catalogmanager01** to see the **GET** API

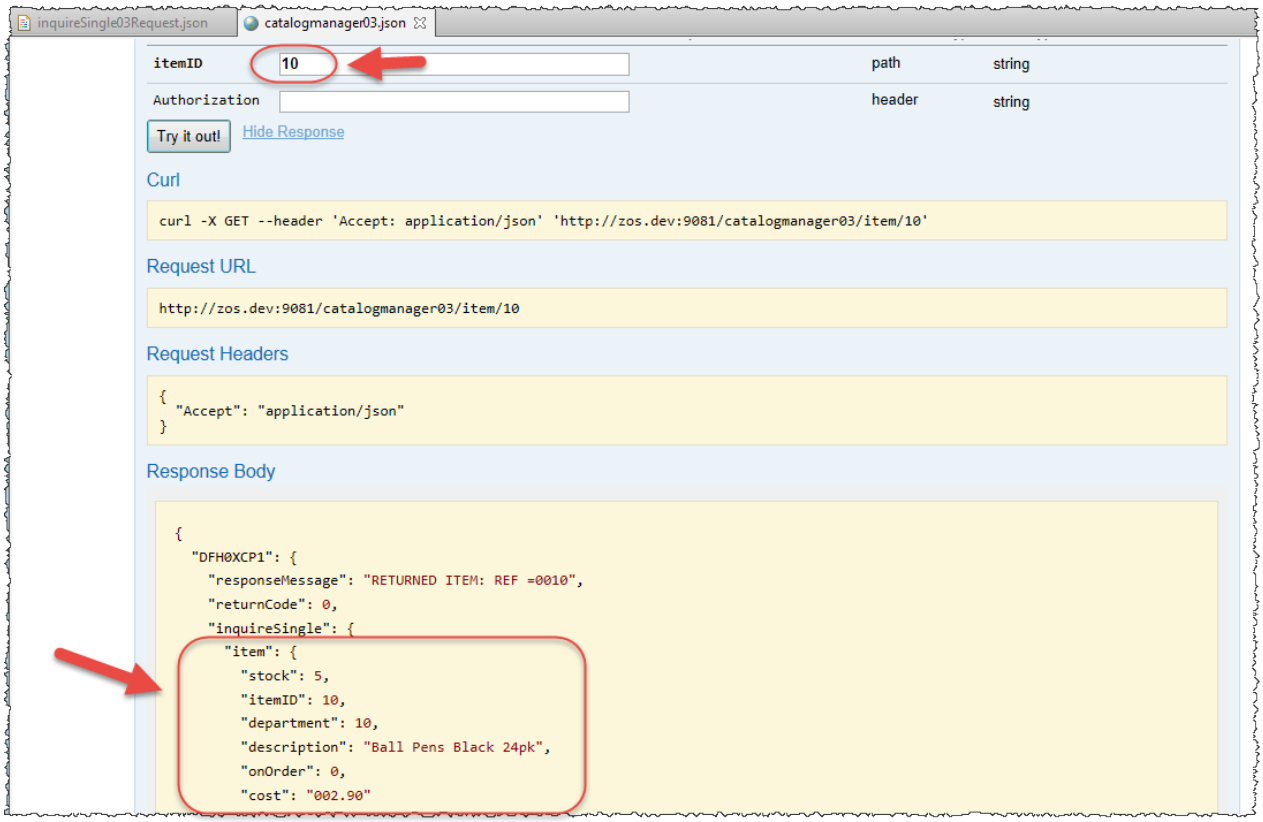
You will see:



4.1.3 ⏭ Click **GET /items/{itemID}**
You will see the description for the operation: the elements to send for the request and the format of the response. Test your inquire single operation by filling in the itemID (**10, 20, 30, …, 210**), then click "**Try it out!**".



4.1.4 Scroll down and check the results

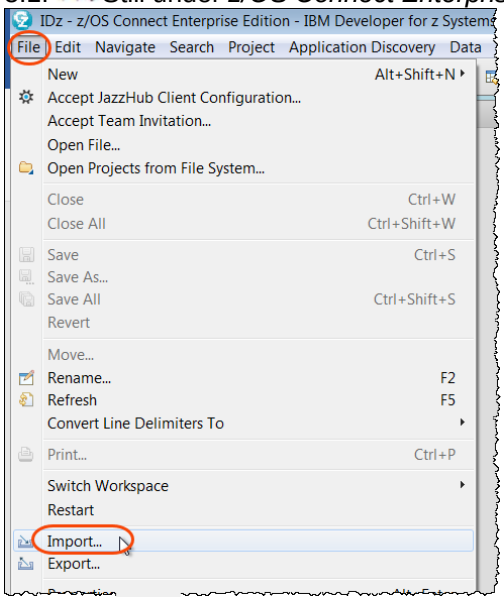


4.1.5 Use **Ctrl + Shift + F4** to close all opened editors.

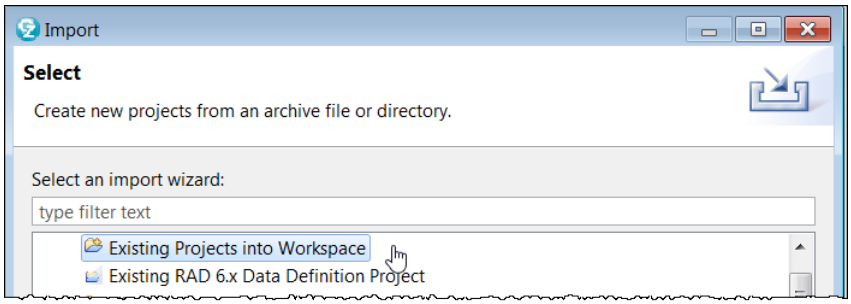
Section 5 –(OPTIONAL) Import the solution

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

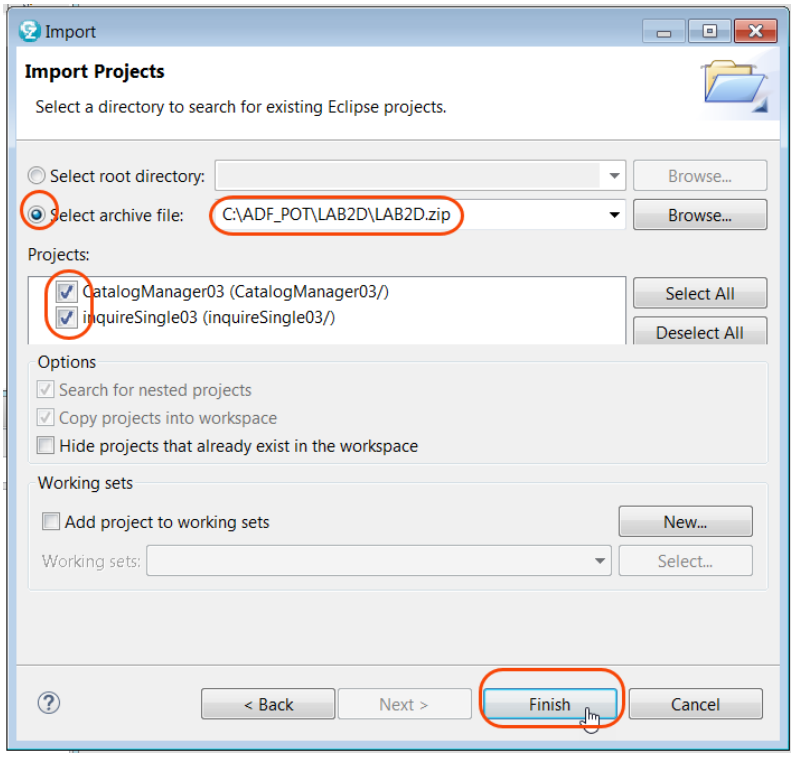
5.1. Still under *z/OS Connect Enterprise Edition* perspective select **File > Import…**



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



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



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

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

Updated July 16 2019 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.

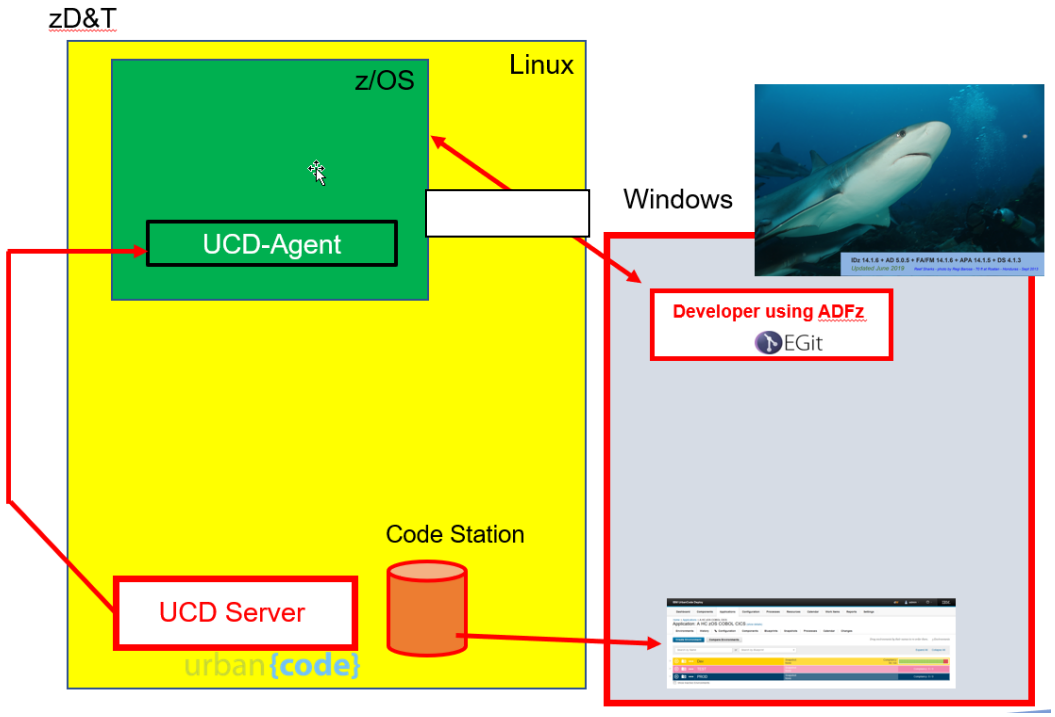
Part 3 – Deploy the application to z/OS CICS
On this part you will deploy the Application to the z/OS CICS.

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

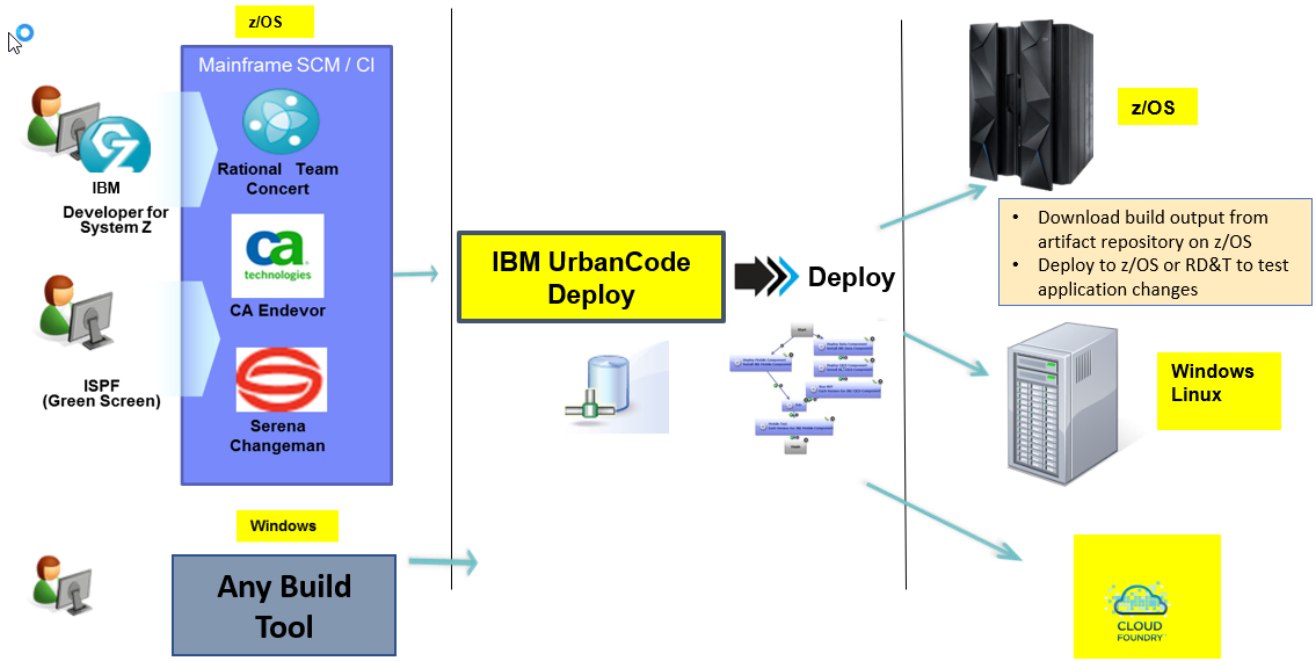
Lab Architecture and proposed scenario

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:

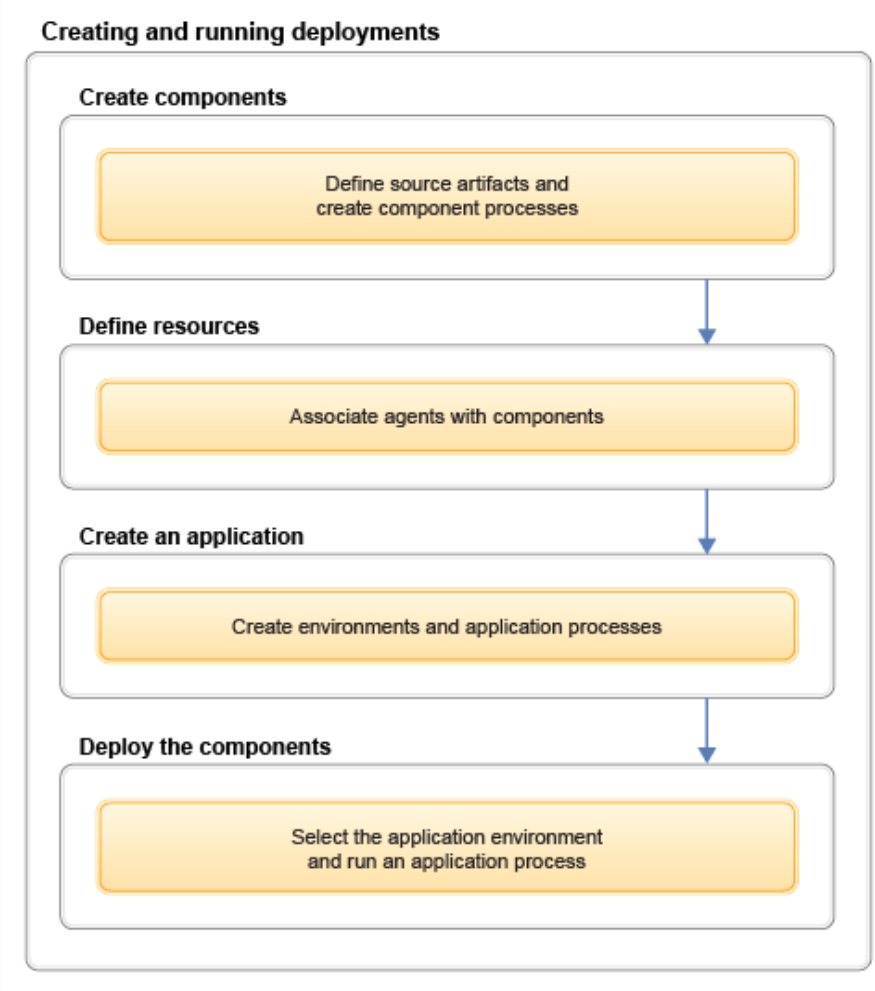


PART 1 – Create the UrbanCode application infrastructure

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:



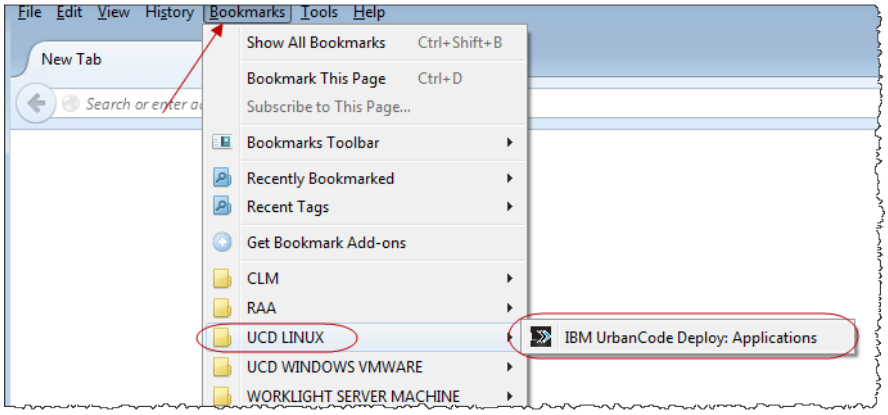
Task 1 – Logon to UCD server running on Linux and verify that the UrbanCode agents are running

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.**

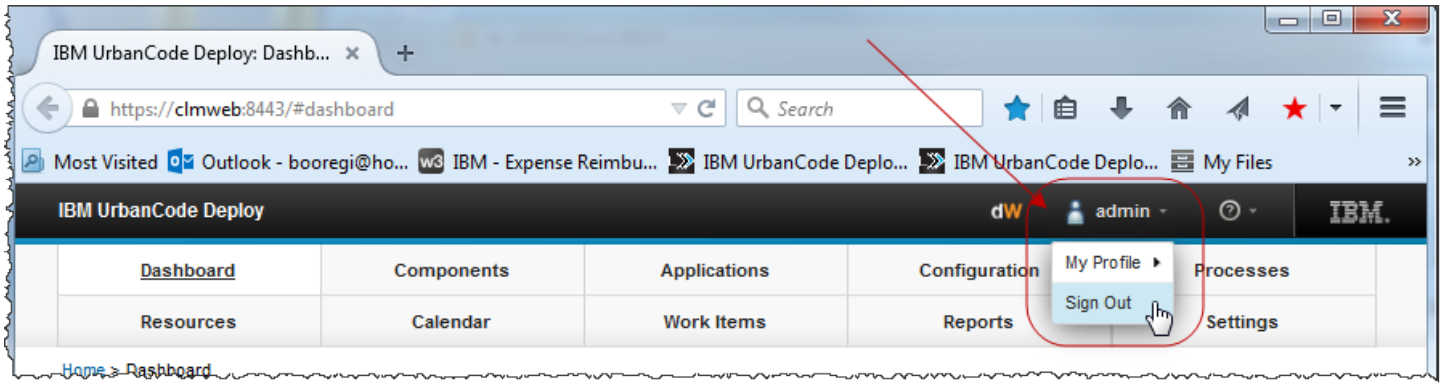
1.1 **Start the Firefox browser** to go to UCD server on Linux You can use the icon on the base of your screen Move the mouse to the base of your screen if you do not see this bar

Look for the **bookmark** below

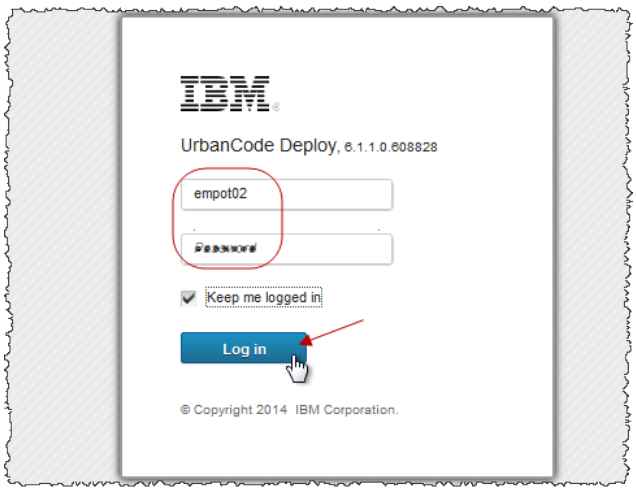


Or use: https://clmweb:18443/

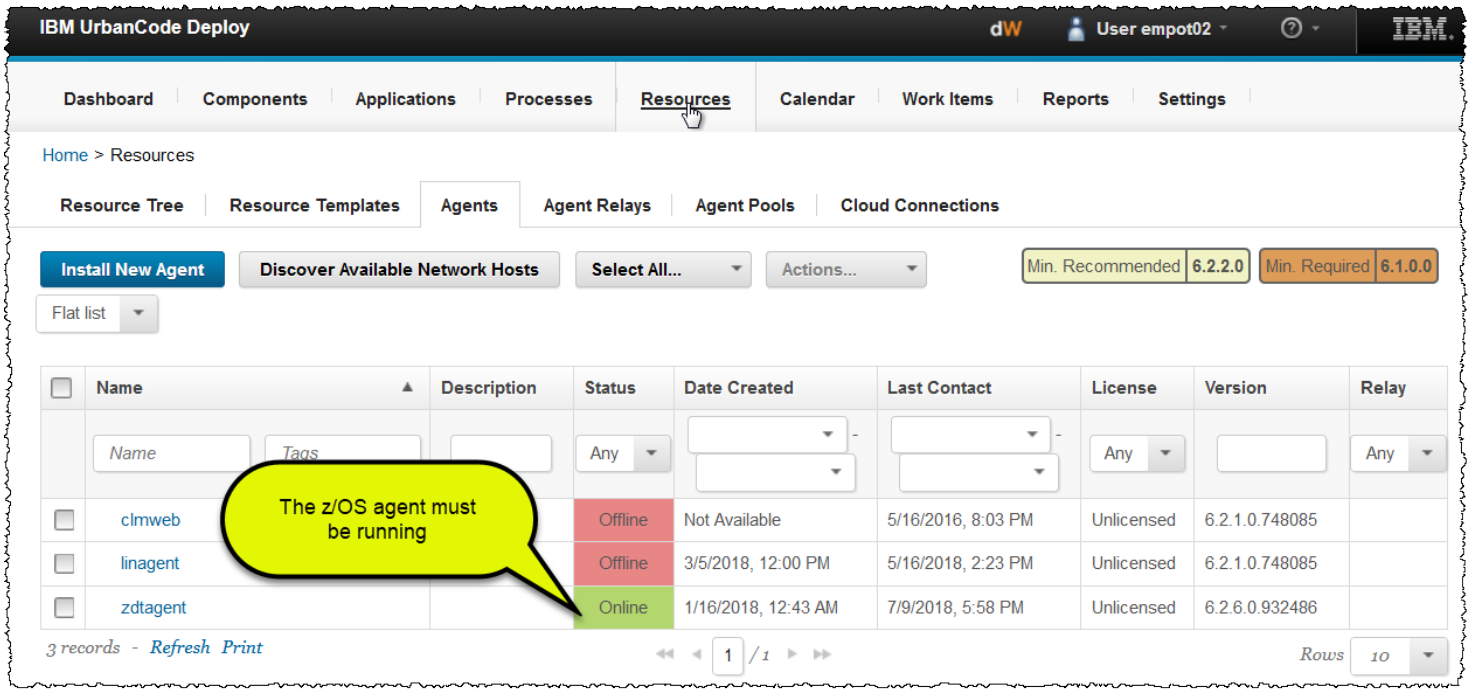
1.2. If you are already logged on, you must first logoff and logon with your right user id.
Click on **admin** (or whatever user id is logged in) and select **Sign Out**



1.3. Logon with **empot02** and password **empot02**
Remember that this is case sensitive here and lower case



1.4. Verify that the agent is installed by clicking **Resources > Agents**.
The Windows server and z/OS agents must appear in the list of agents with a status of ***Online.***
On this lab you will use only the z/OS agent **(zdtagent**)



Task 2 – Create a UCD component

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

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

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

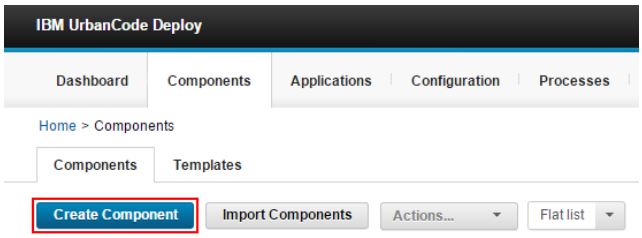
z/OS: What is a PDS?

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

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

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

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



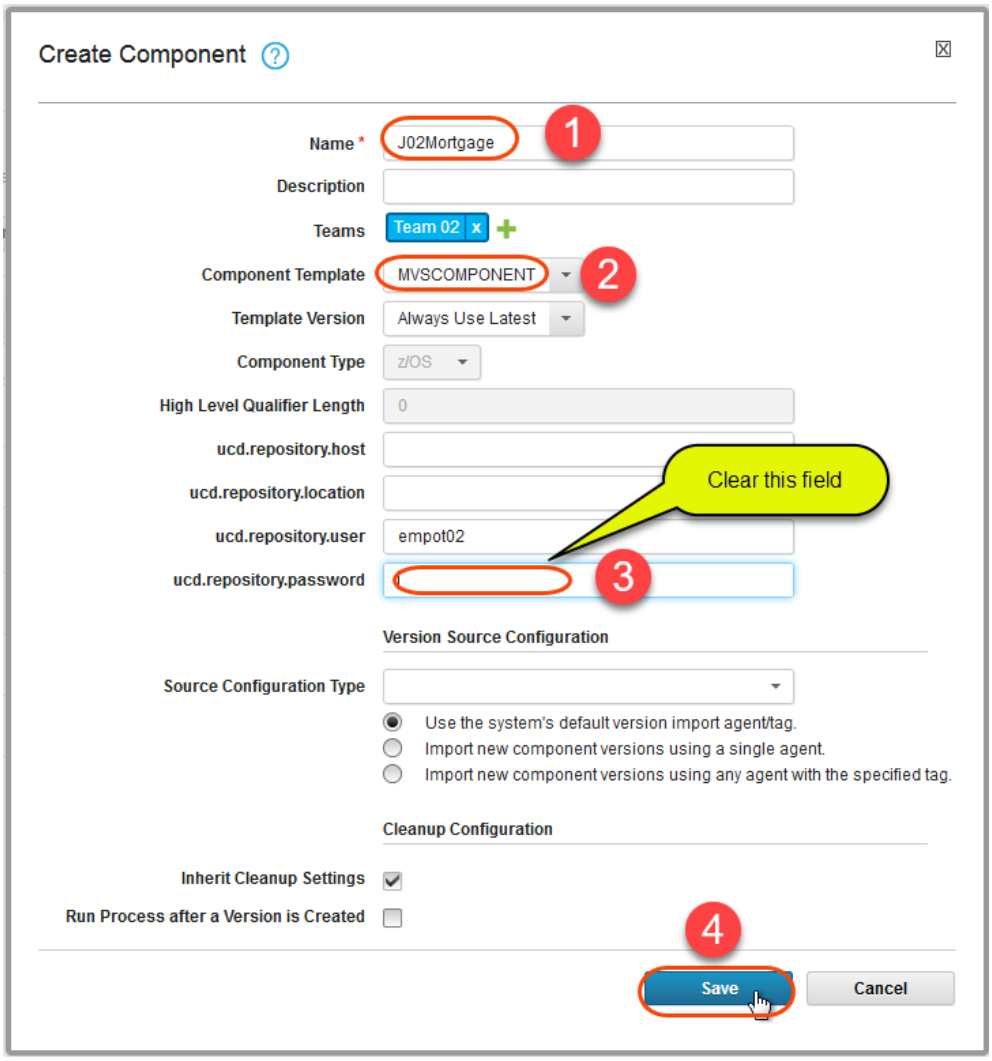
2.2 Provide values for the required fields on the **Create Component** dialog.

a. ⏭ Specify component Name as **J02Mortgage**
(Be careful since the name is case sensitive).

b. ⏭ 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

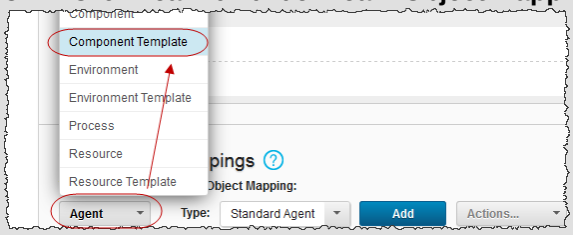
c ⏭ Clear the field *ucd.repository.password* and click **Save**.

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.

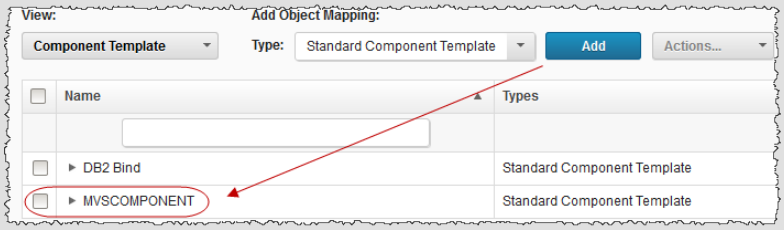


2.3 The component **J02Mortgage** is created

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

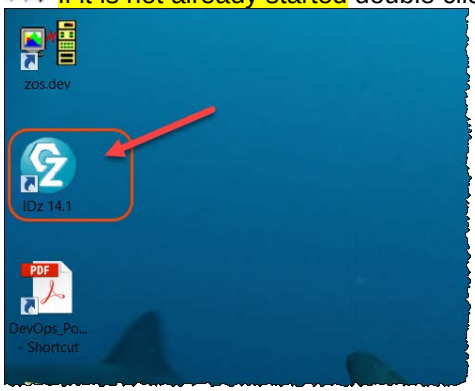
The z/OS **ship list file** specifies which files from the build to include in the new component version to deploy.

You must create a ship list file before you run z/OS deployment tools.

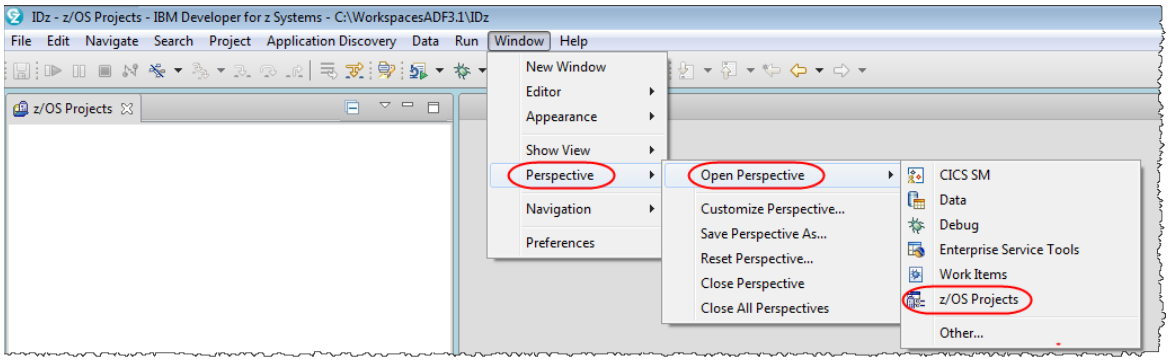
Ship list files are XML files that contain a list of files to be deployed on z/OS. For more information, refer to the UCD documentation:
http://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.doc/topics/zos_shiplistfiles.html

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* if it is not already started
if it is not already started double click on **IDz 14.1** icon

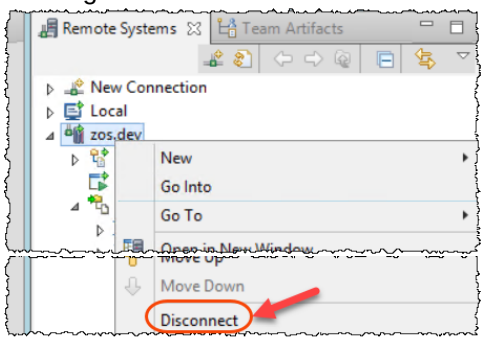


3.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Open Perspective > z/OS Projects .

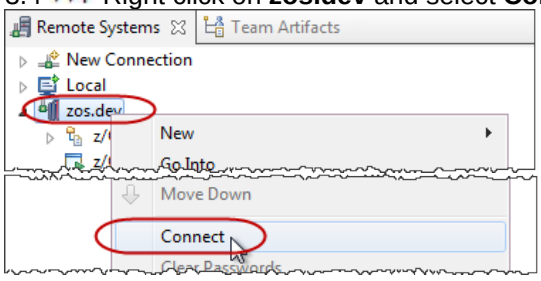


3.3 If you used userid *empot01* before you need to disconnect. Your new userid now will be **empot02**.

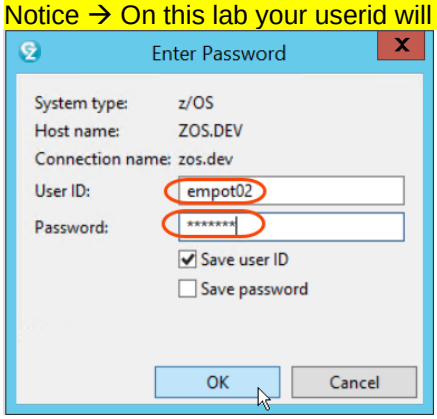
⏭ Right click on **zos.dev** and select **Disconnect**



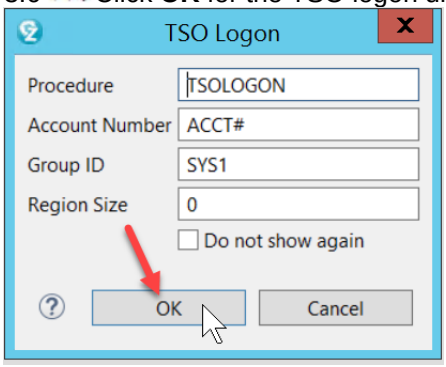
3.4 ⏭ Right click on **zos.dev** and select **Connect**



3.5 ⏭ Type **empot02** as userid (might be already there) and **empot02** as password.
Click **OK** to connect to z/OS.
Notice → On this lab your userid will be **empot02** (not empot01 used in previous labs)



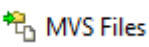
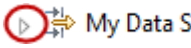
3.6 Click **OK** for the TSO logon dialog



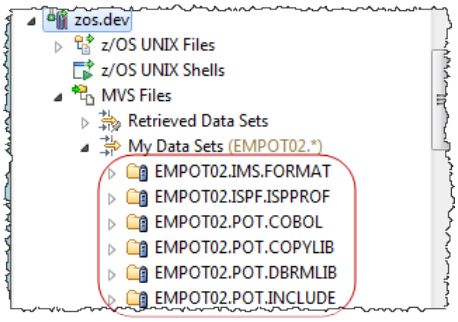
3.7 Notice that the folder icons changed once you are connected to z/OS. A small green arrow is added.,,

From…

To:

Expand  and  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.

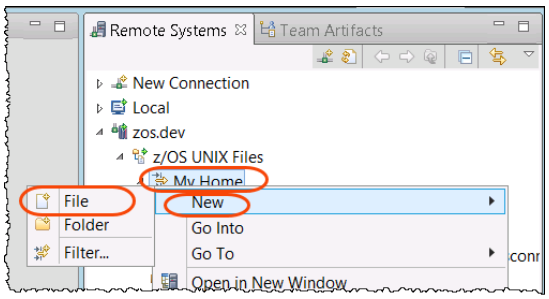


You are connected to a z/OS remote system. Now you will create a Ship List and submit JCL to create the executables to be deployed.

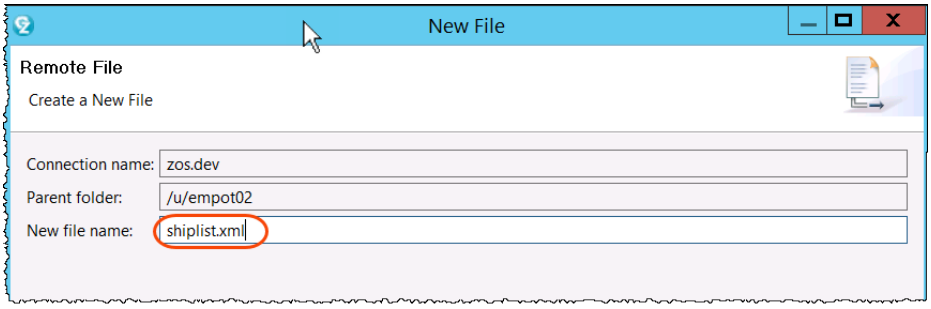
3.8 You now can create the UCD ship list. An XML file that has the list of the load modules that will be deployed.This xml could be created from an SCM build tool like RTC, Endevor etc..
For this lab we will create it manually. .

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.

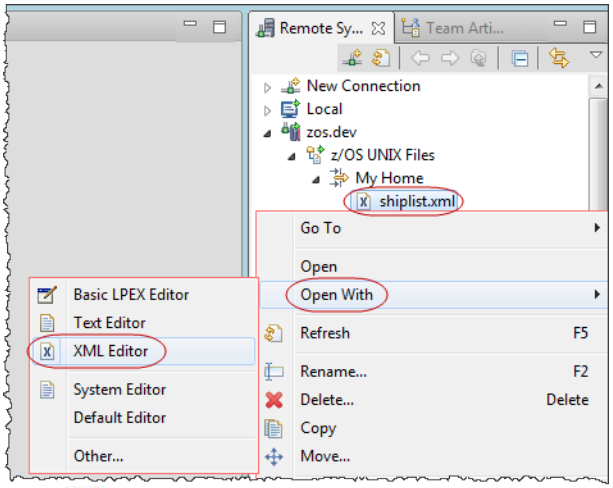
3.9 Using the *Remote Systems* view (on top right), right click on **My Home → New → File**



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 XML Editor**

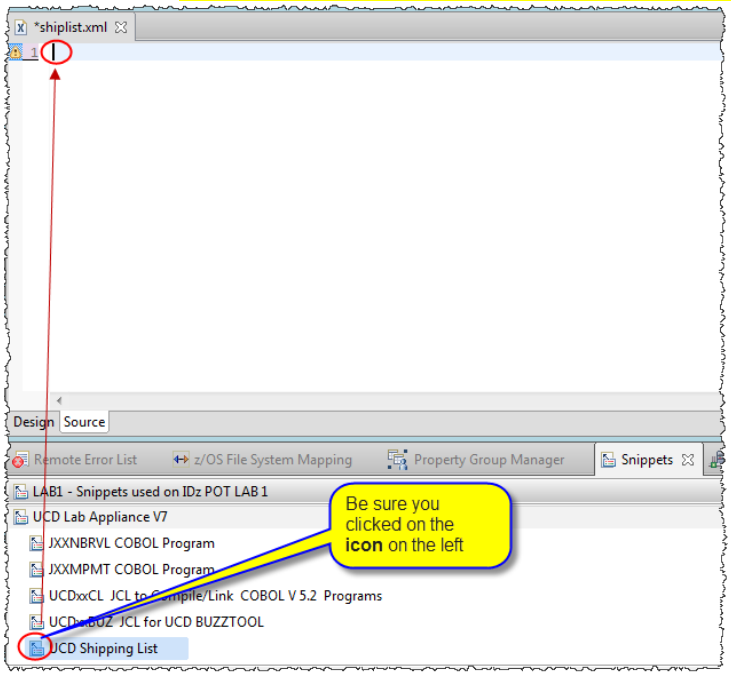


3.12 Click on **Source** tab

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

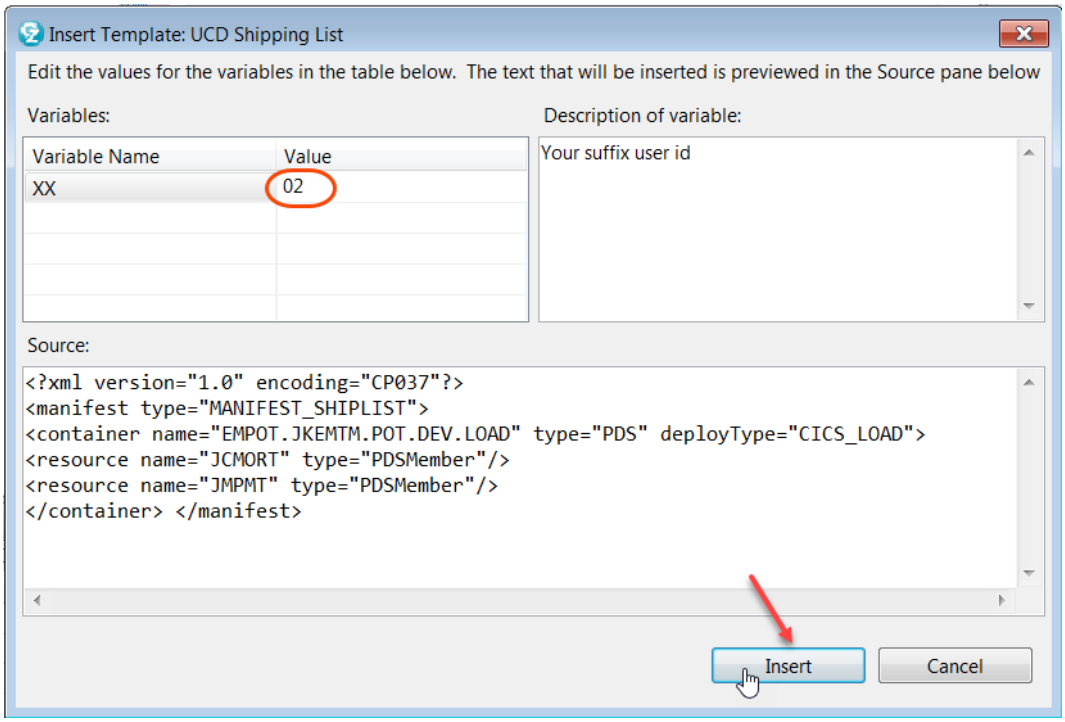
Use the **UCD Shiping List** snippets to drag and drop to the **FIRST position** of the file

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

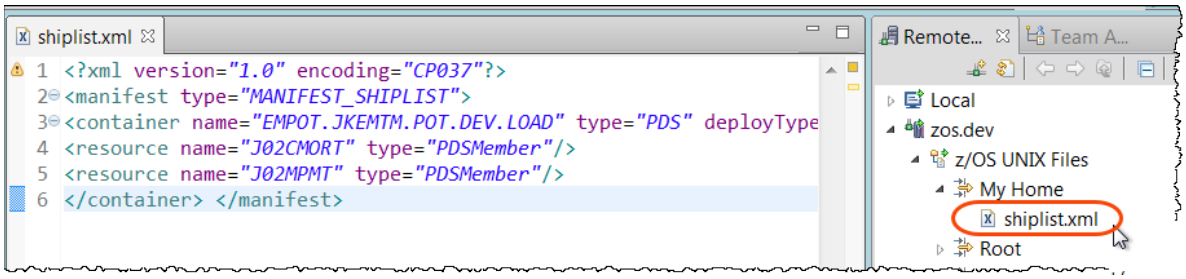


3.14 When the dialog opens **type *02***. ..

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



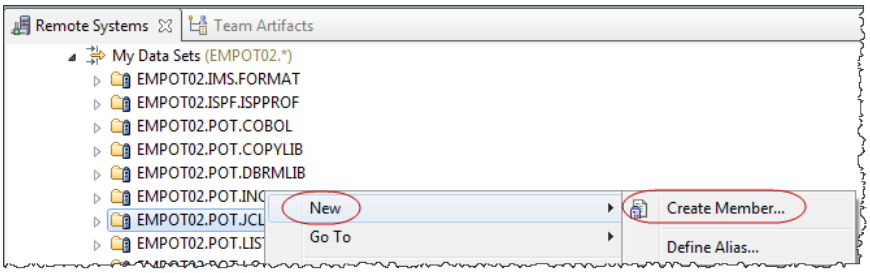
3.16 ⏭ Use **Ctrl + S** to save the changes



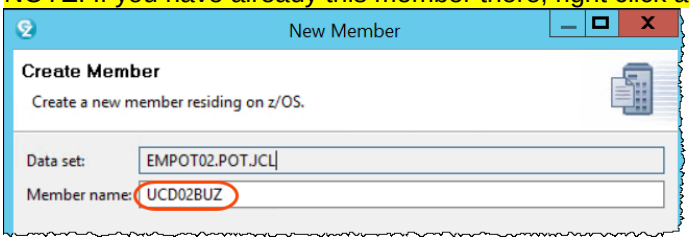
Task 4 - Create the UCD component version from JCL

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).

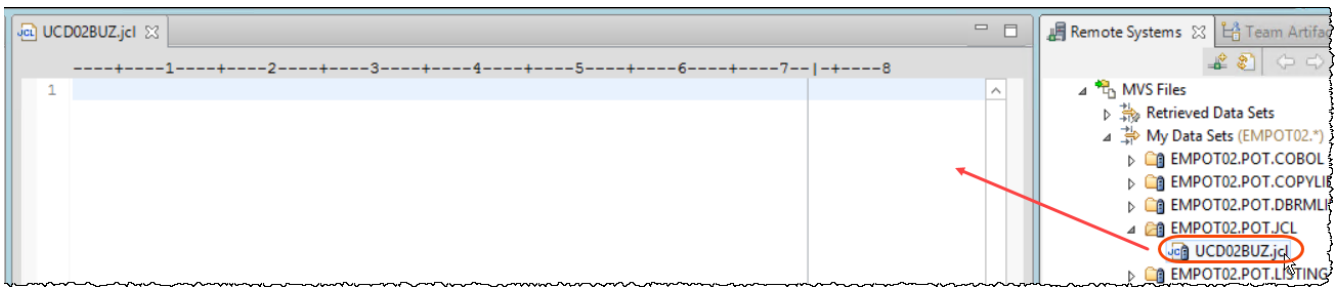
4.1 ⏭ Using *Remote System* view**, r**ight click on **EMPOT02.POT.JCL** and create a member .
If you don't have this PDS call the instructor.



4.2 ⏭ Name it **UCD02BUZ** and click **Finish**
NOTE: If you have already this member there, right click and delete it.

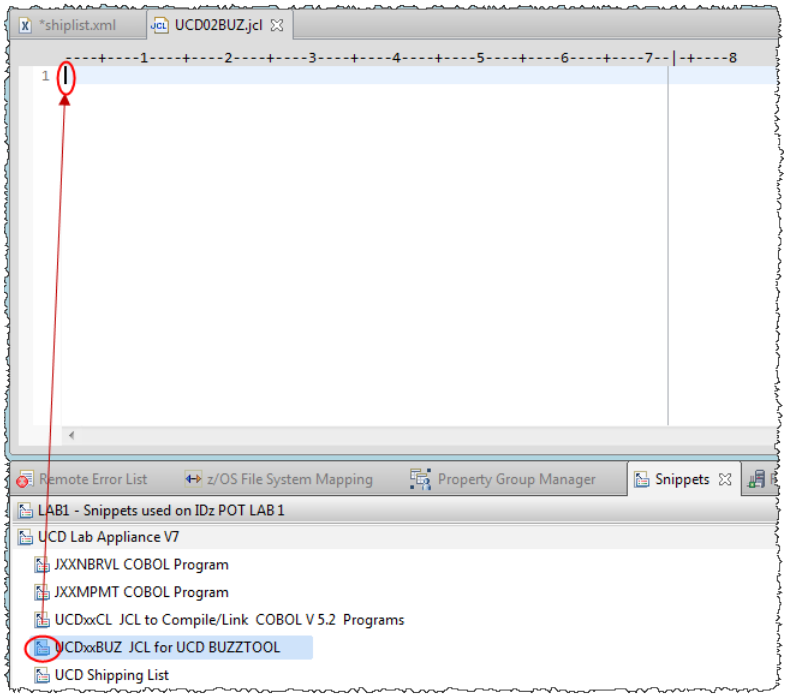


4.3 ⏭ Double click on **UCD02BUZ** to edit.



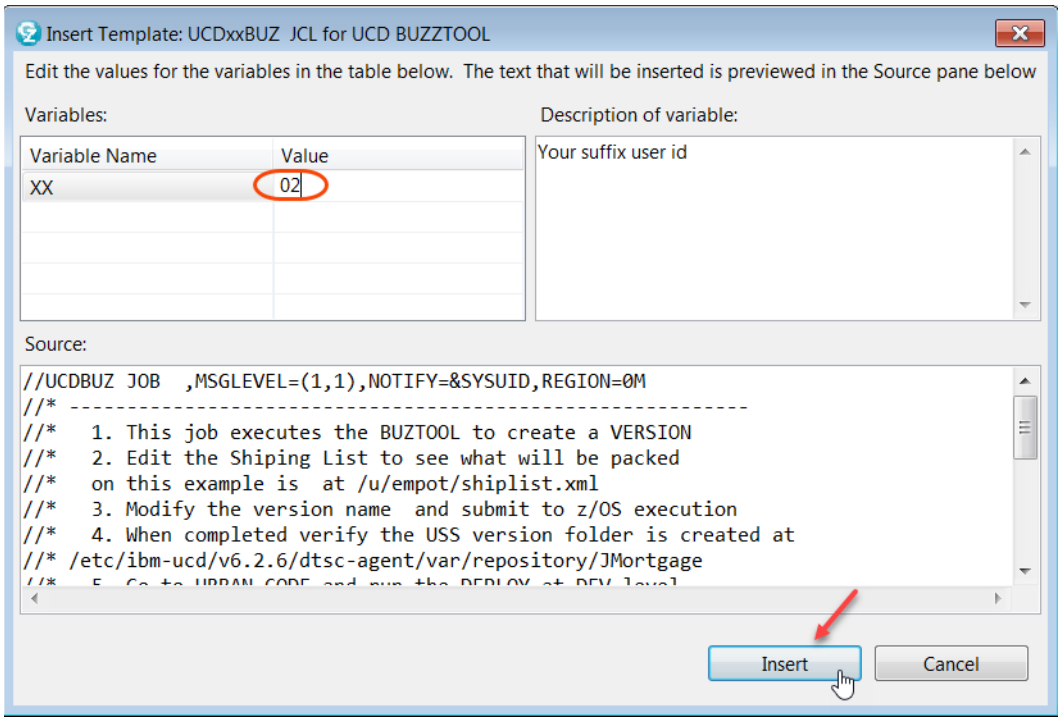
4.4 Use the **UCDxxBUZ** snippets to drag and drop to the **FIRST POSITION** of the file

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



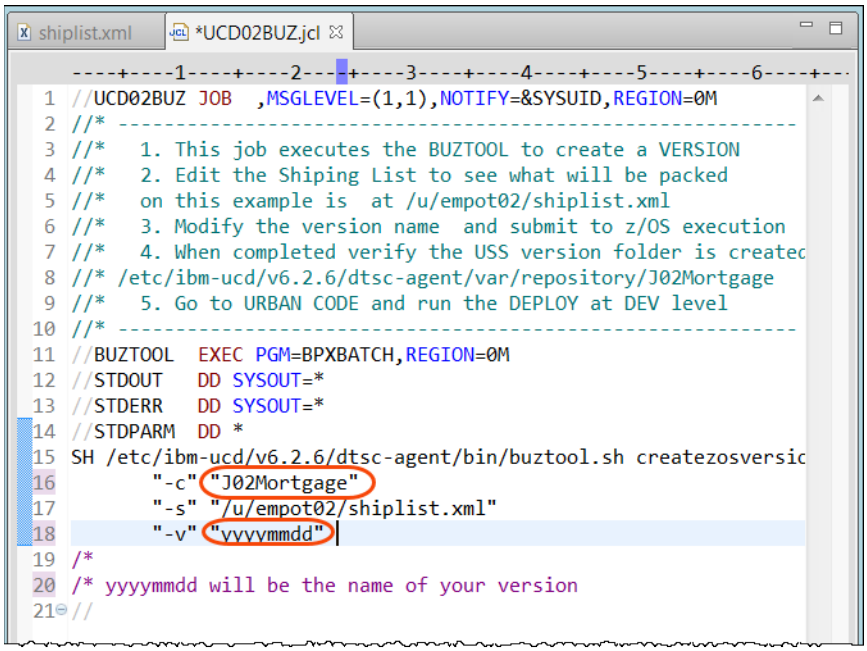
4.5 When the dialog opens **type 02**. ..

Click **Insert** to insert the lines. Note that the variables are replaced



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

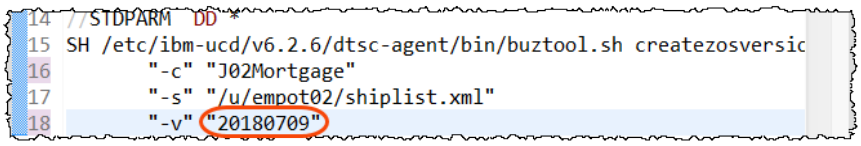
4.6 **Scroll down** and understand the parameters passed to the BUZTOOL



*Possible arguments for the buztool **createzosversion** command*

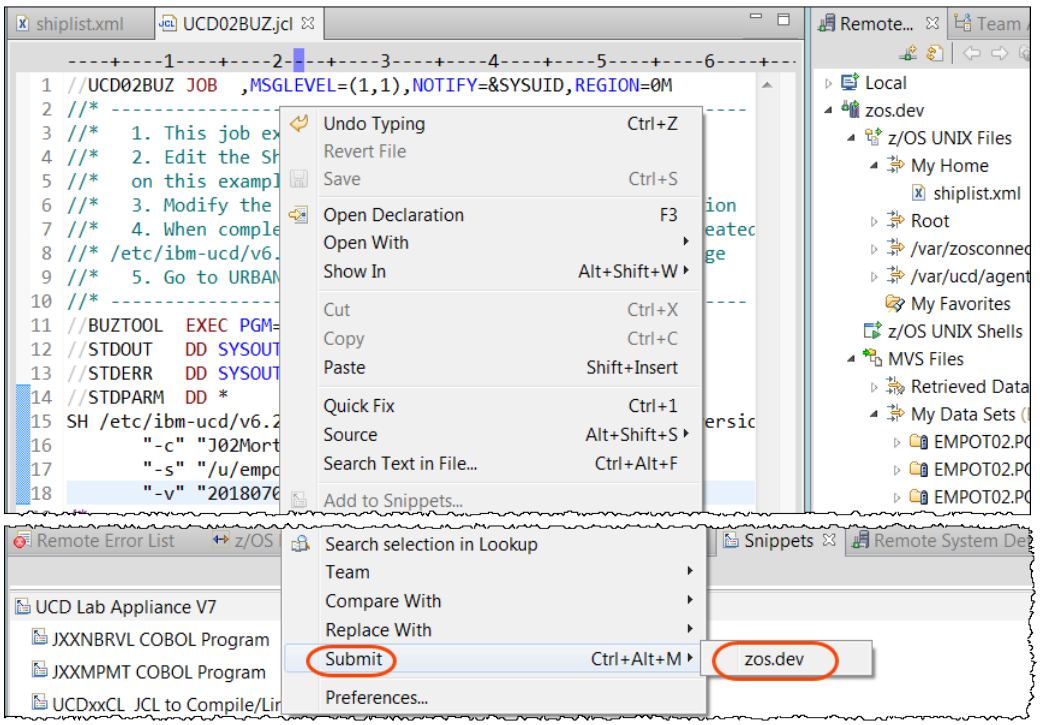
Parameter	Required	Description
-c	true	The name of the component in IBM UrbanCode Deploy. The component name can contain only letters, numbers, and spaces.
-v	false	The name of the version to create. If a version is not specified, a version name is generated from the current time stamp. The version name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



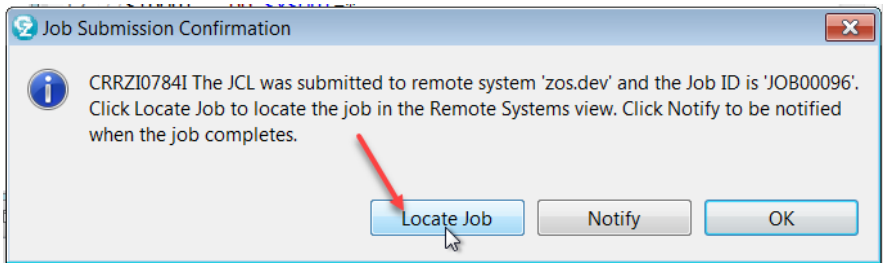
4.8 ⏭ Use **Ctrl + S** to save

⏭ **Right click** on the editor and select **Submit → zos.dev**



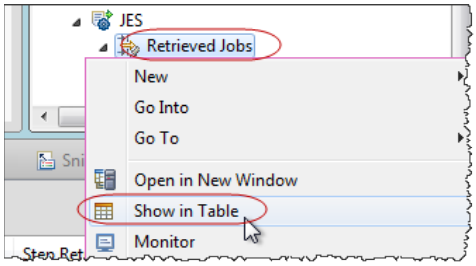
You do not find the option Submit?…
Be sure you are using the correct editor. Either LPEX or JCL editors will provide this capability when right clicking. Be sure you pasted the content on line 1 and column 1. If you still don't see, you can right click on the editor content and select ***Open with →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

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

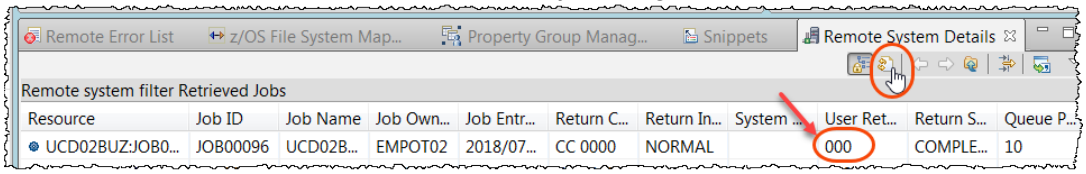


This job may take as long as 1 or 2 minutes.. Remember that your z/OS instance on cloud has limited power resources and is very slow.

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

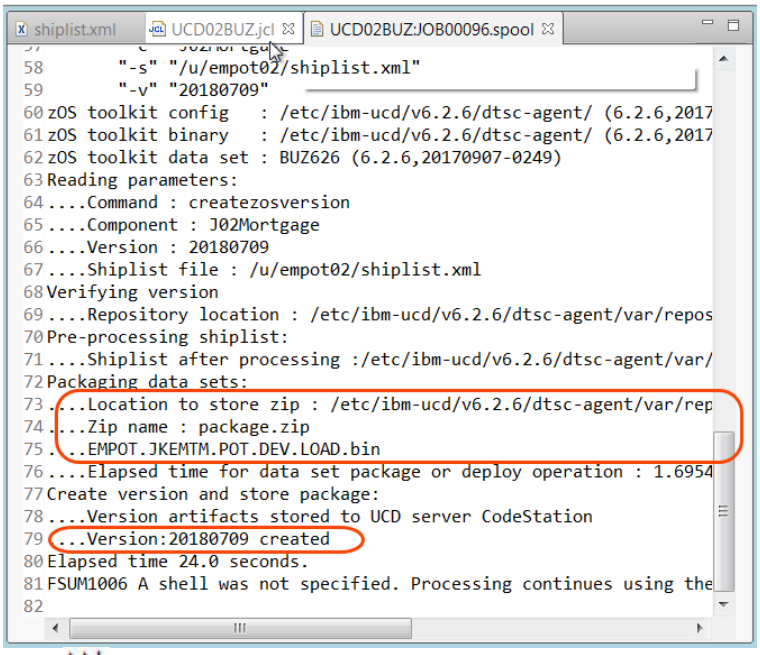


4.11 Use the **refresh icon** to see the job being executed



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

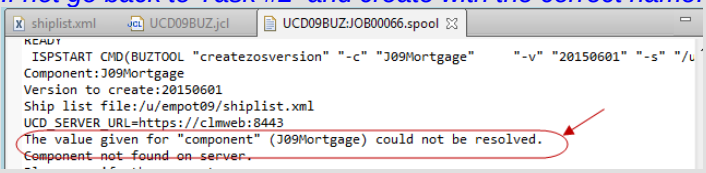


4.13 Use **Ctrl + Shift + F4** to close all editors.

Errors like the one below?.

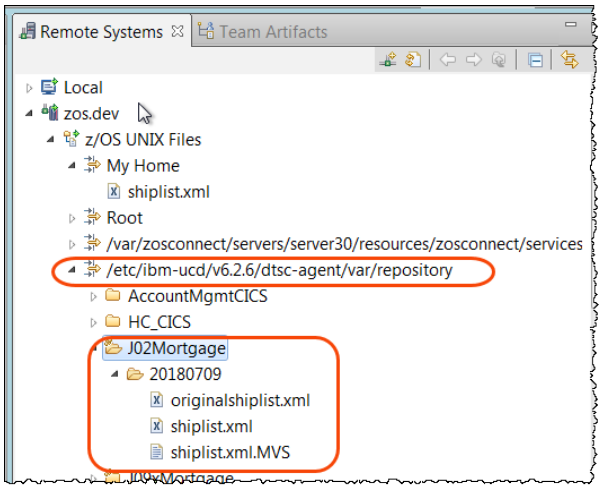
Be sure that you have created the component with the correct name (mixed case letters).

If not go back to Task #2 and create with the correct name.



4.14 **(Optional)**

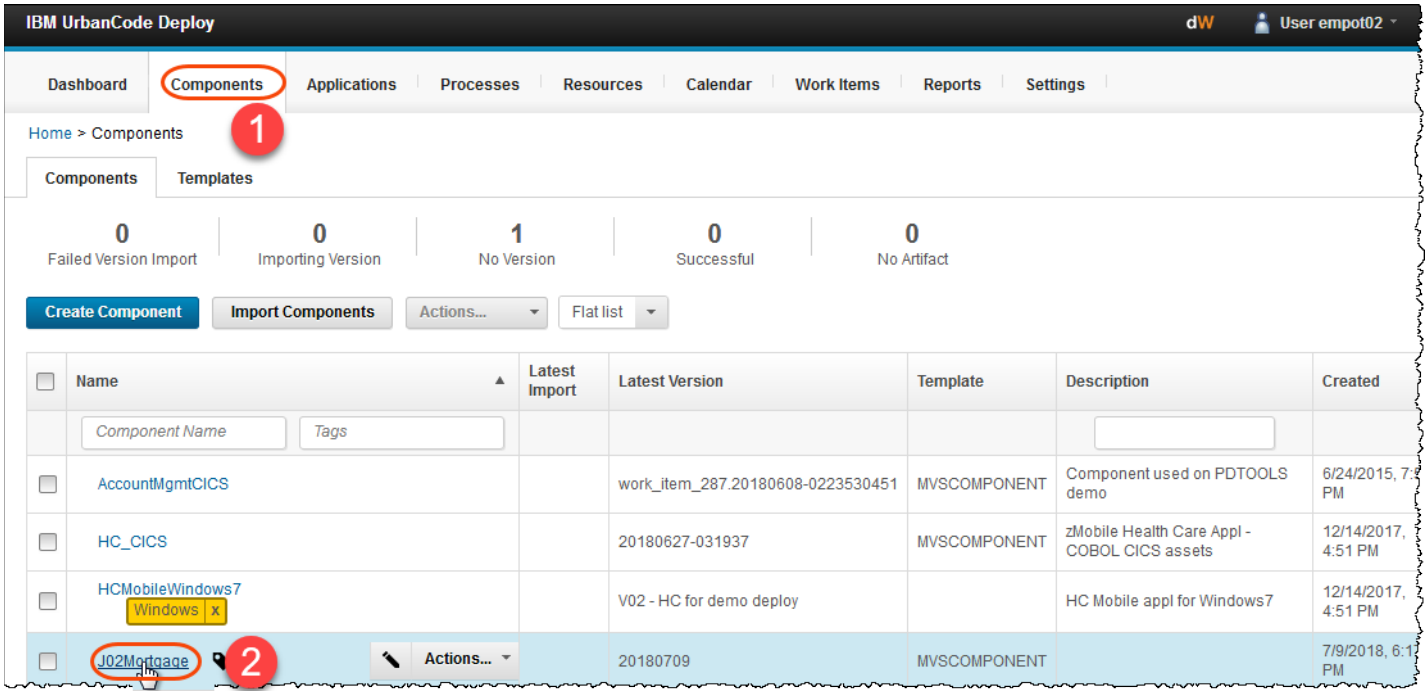
Under **z/OS UNIX Files** navigate to the filter
`/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository`
Expand **J02Mortgage** to see the UCD metadata.
Notice that the zip file is located on the UCD CODE STATION on Linux (not on z/OS).



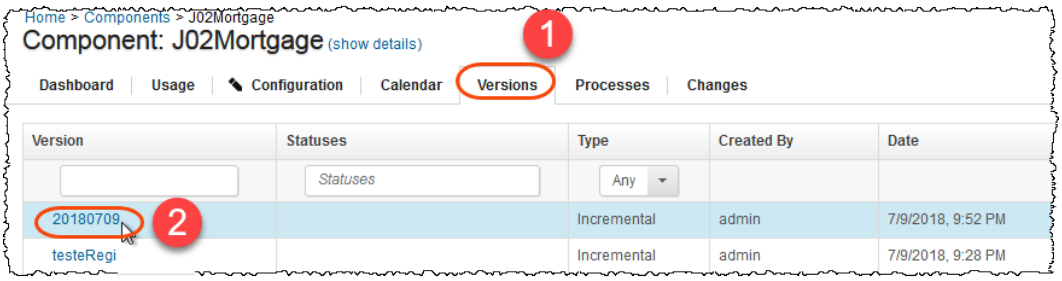
4.15 **Close** or minimize IDz, you will not need any more on this lab.

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

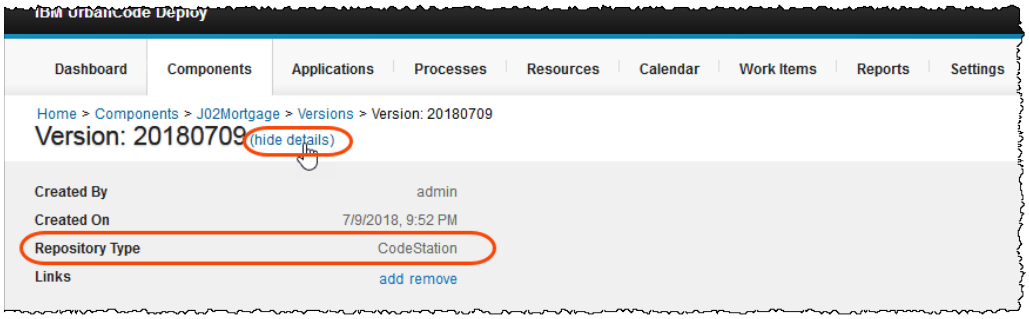
4.16 Now Using the UCD browser, click **Components -> J02Mortgage**



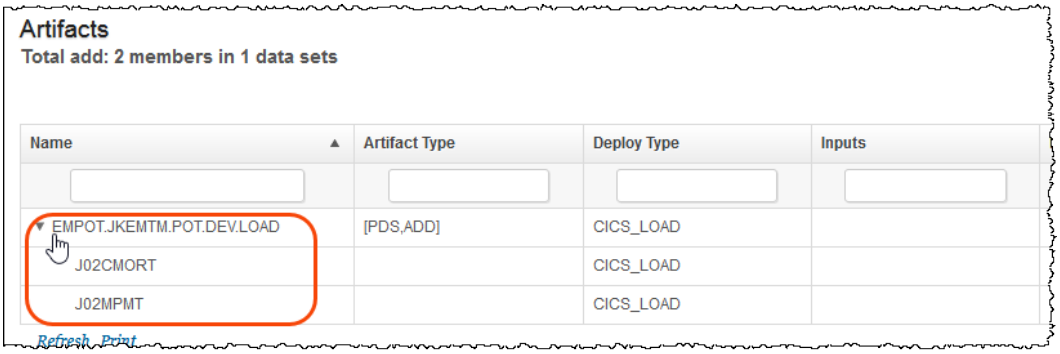
4.17 Click , 1 **Versions** and the 2 version that you created (**yyyymmdd**).



4.18 Click **Show Details**. You can verify that the repository is on UCD Server Code Station (not HFS)



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):

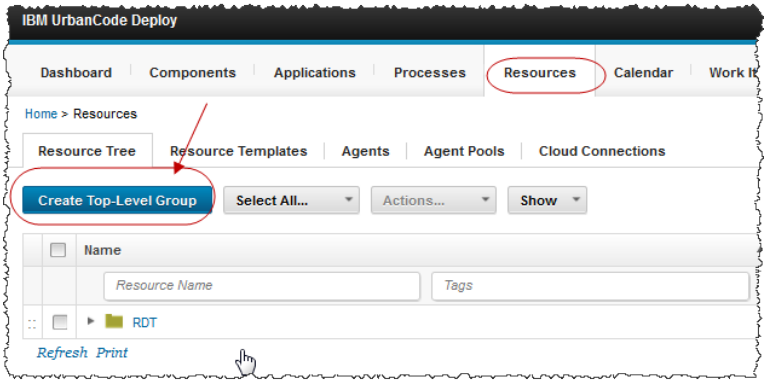


Task 5 - Create UCD Resources

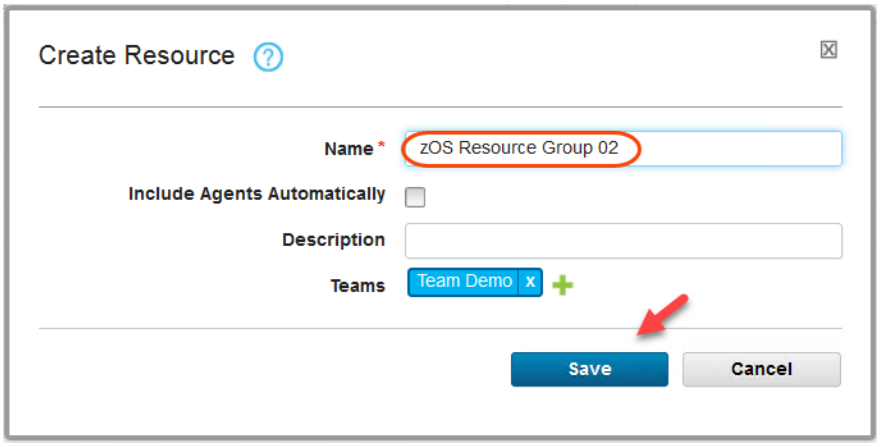
On this task you will create the UCD resources that is a user-defined construct.

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.

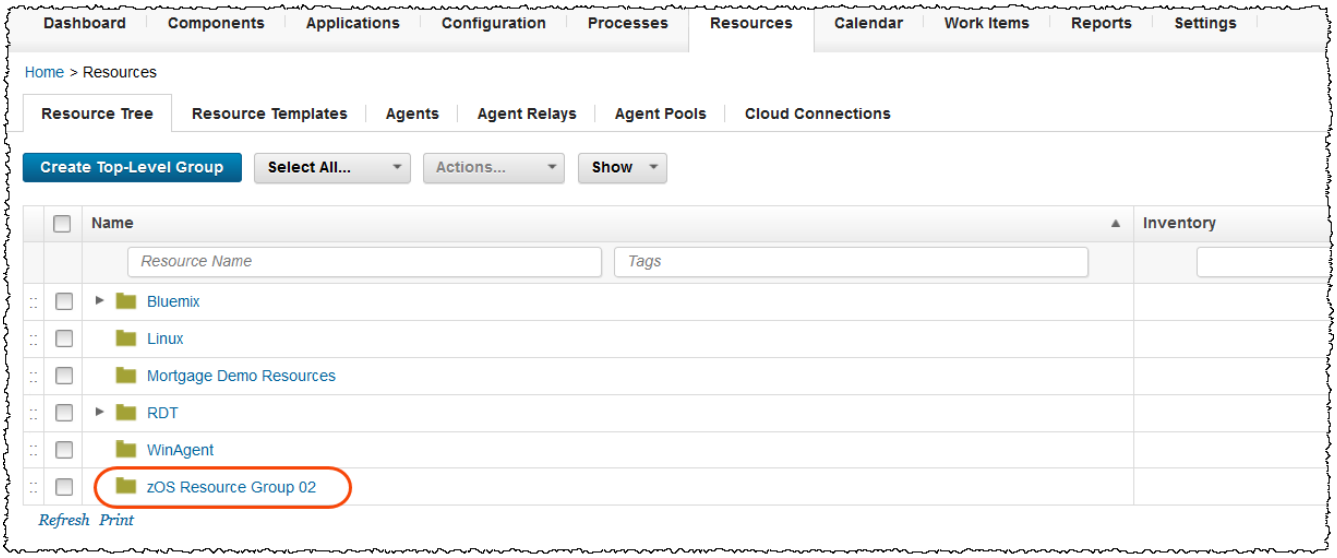
5.1 On the **Resources** tab, click **Create Top-Level Group**:



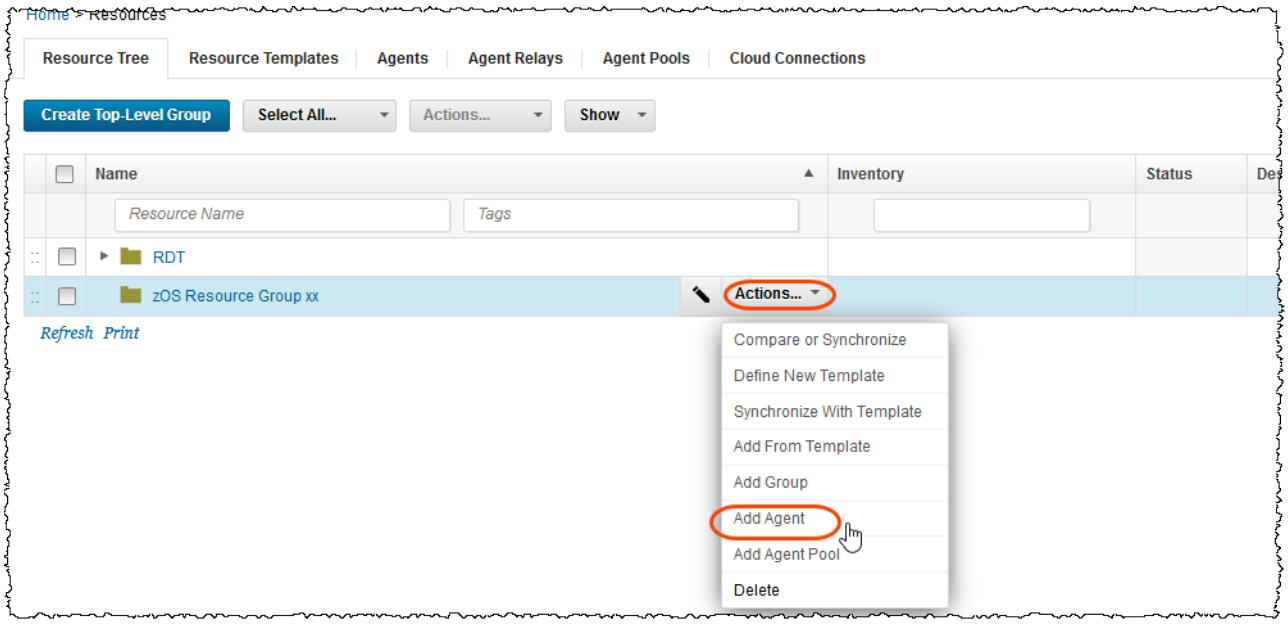
5.2 Use **zOS Resource Group 02** and click **Save.**



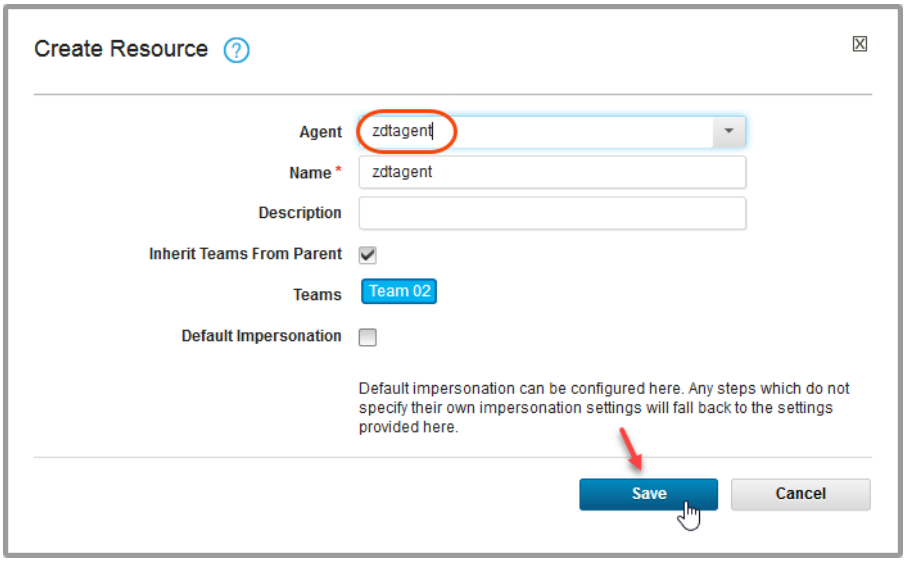
You should have:



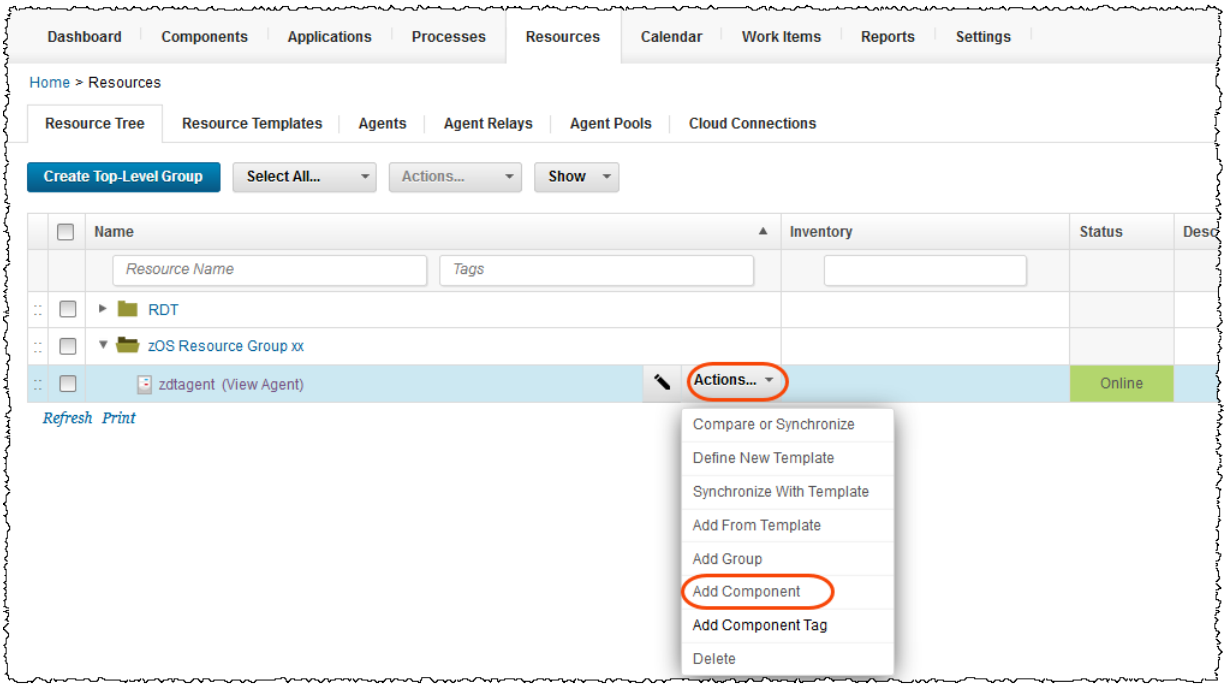
5.3 Click on **ZOS Resource Group**, click the **Actions** drop-down menu of the newly created Resource Group and then click **Add Agent**:



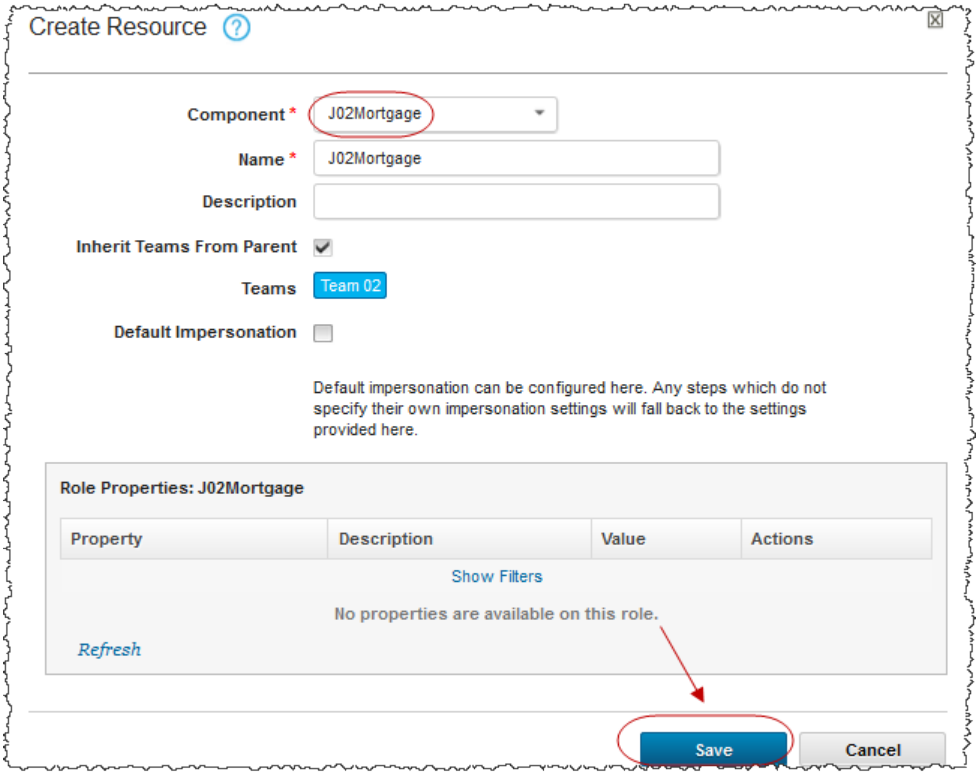
5.4 In the **Agent** drop down dialog, select **zdtagent** and click **Save**



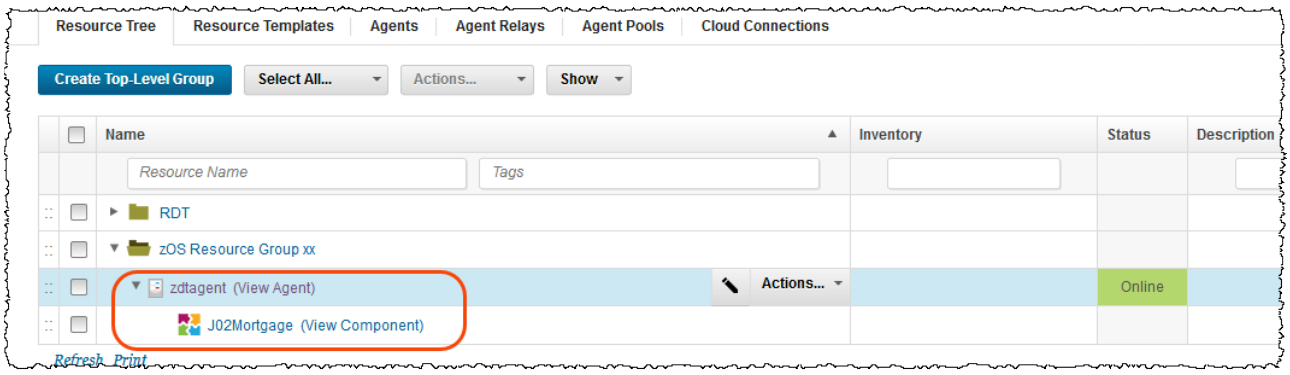
5.5 In the row **zdtagent,** click the **Actions** drop-down menu of the newly added Agent entry and then click **Add Component**:



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



You should have:



Task 6 - Create an UrbanCode Application

On this task you will create an UCD application.

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

UCD : What is a UCD Application?

Application is a module the contains the following elements:

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

UCD: Properties

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

You can refer to a property by scope:

${p:scope/propertyName} for example: ${p:environemt/DBconnect}

or without scope:

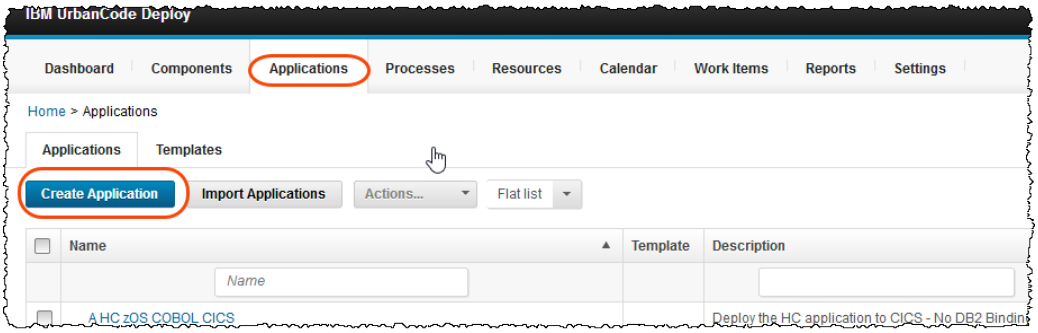
${p:propertyName}

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

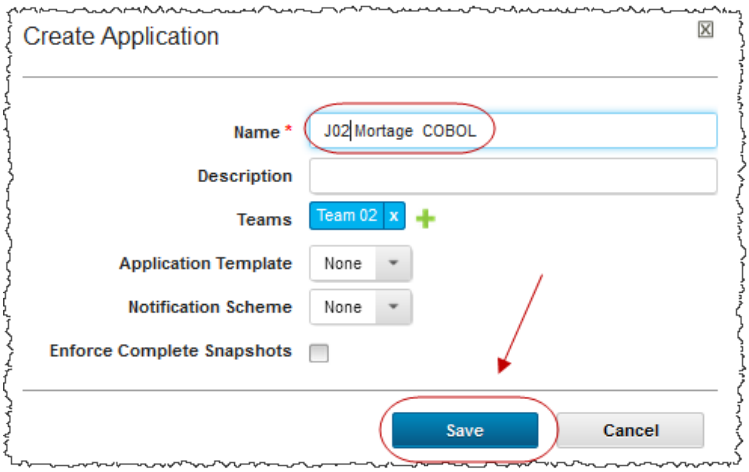
For more details, refer to the full documentation on Properties:

http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.reference.doc/topics/ud_properties.html

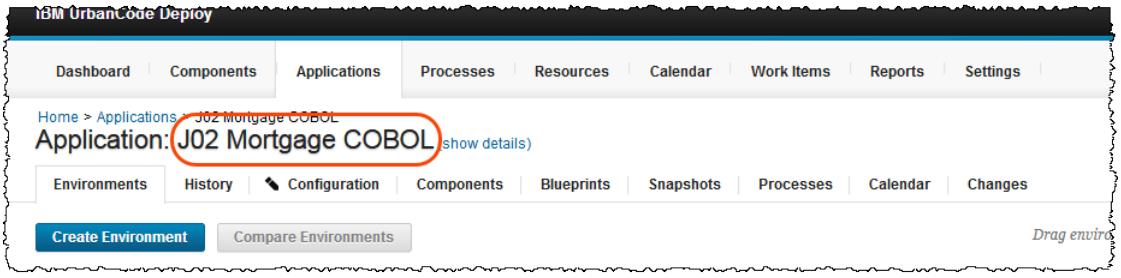
6.1 ⏭ On the **Applications** tab, click **Create Application**:



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:

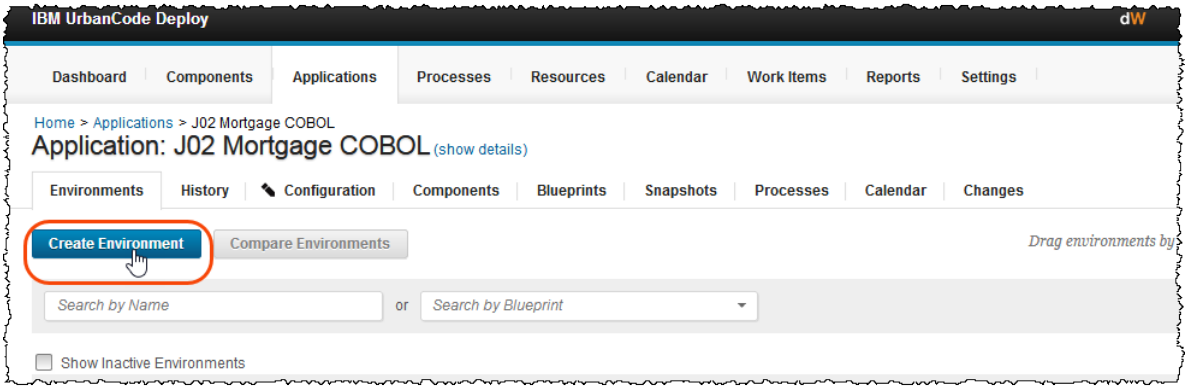


Task 7 - Create environment

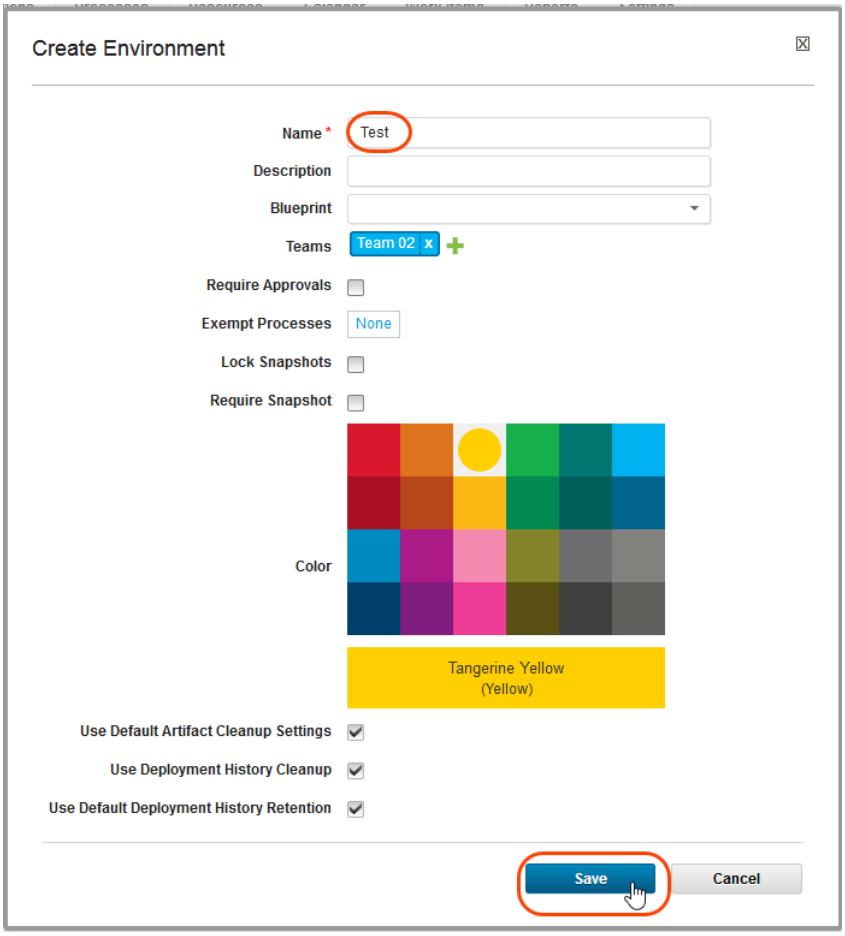
On this task you will you create The UCD environment that is a user-defined collection of resources that hosts an UCD application.
When you create an environment, you map resources to it that define where the parent application can run deployments.

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.

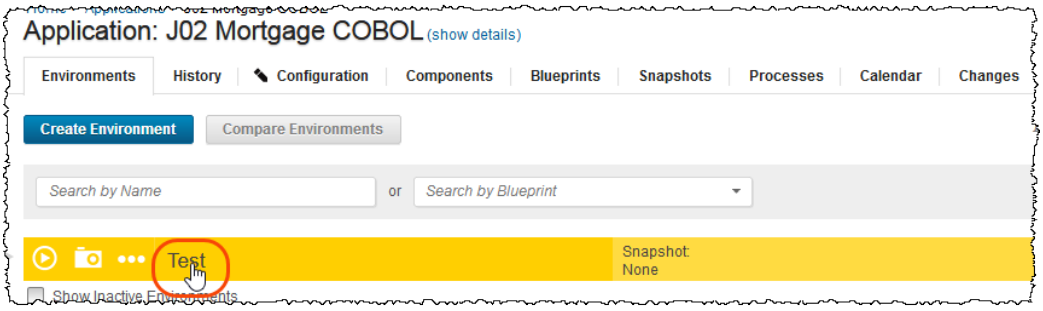
7.1 Click **Create Environment**:



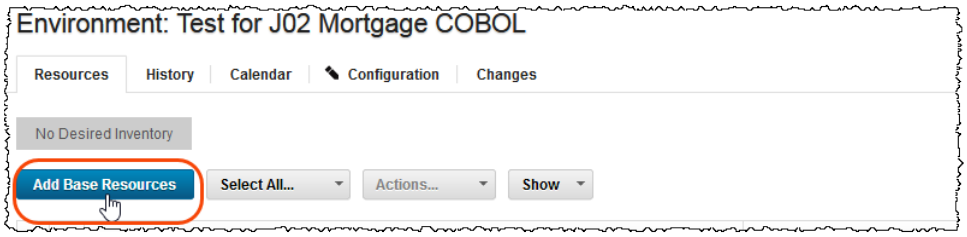
7.2 Provide the value for the Name field in the Create Environment dialog (for example, **Test**), choose a color, and click **Save**:



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

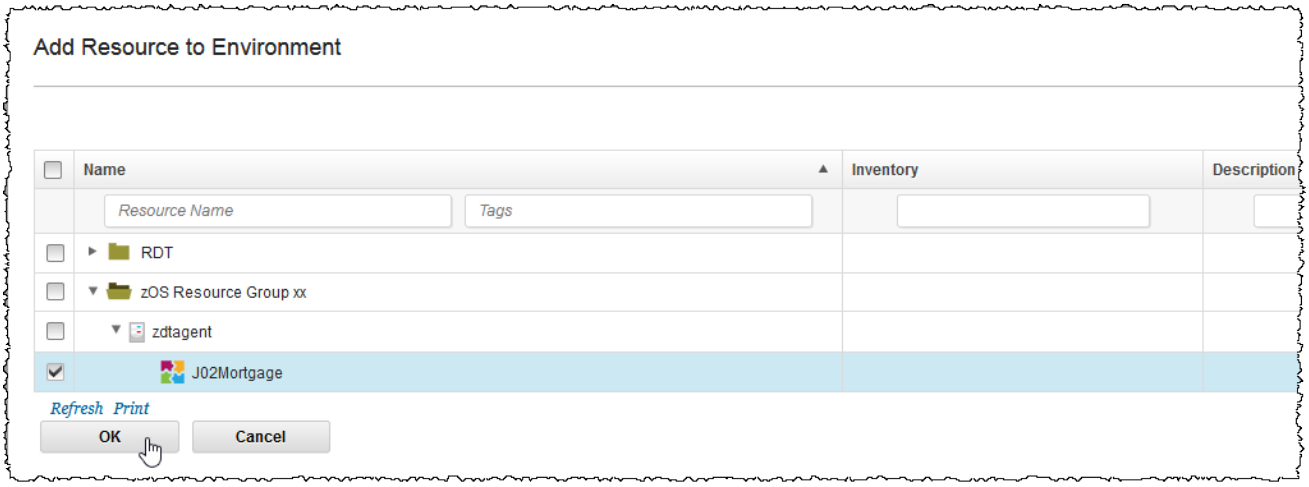


7.4 Click **Add Base Resources**:

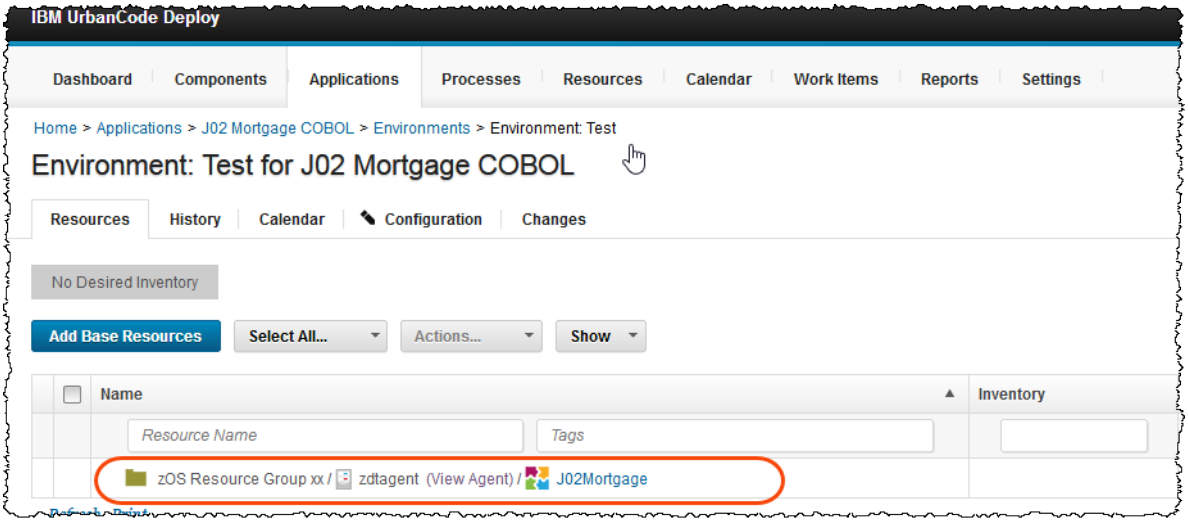


7.5 Select **Component** created earlier ((**J02Mortgage**): and click **OK**:

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.



You now must have:



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.

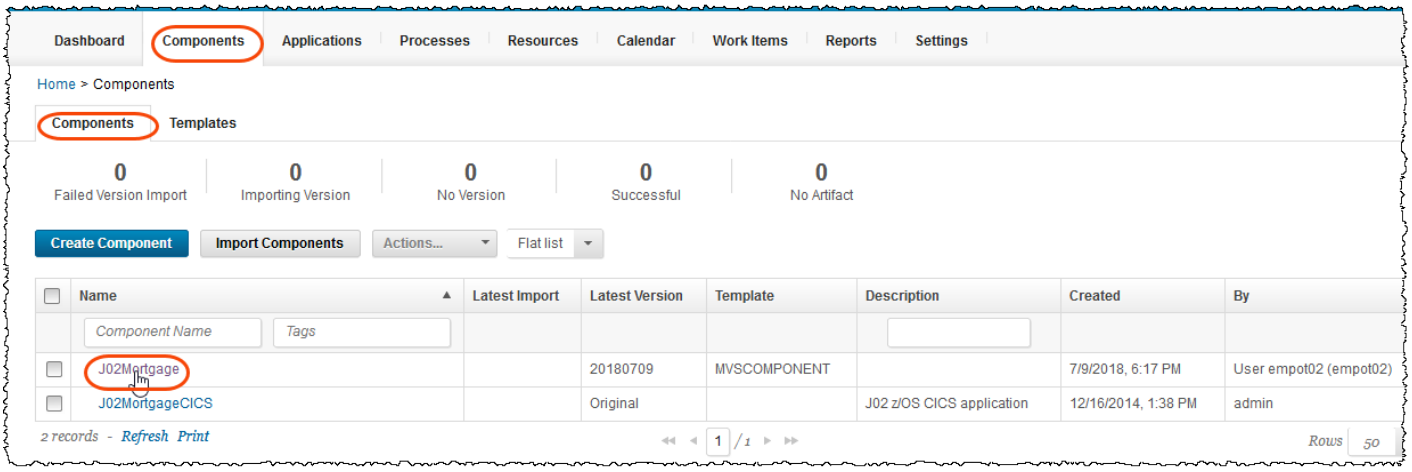
Task 8 - Create a components process

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.

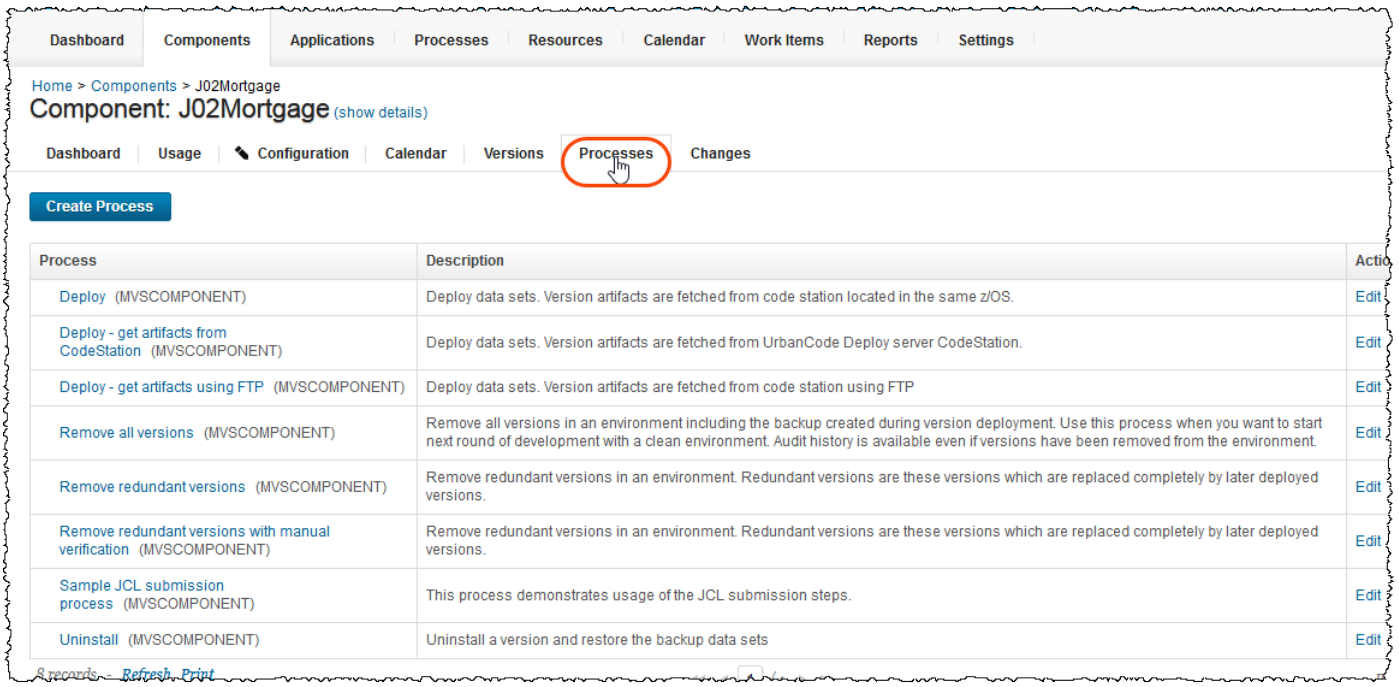
In this task you create a deployment process for your component. Later, you create an application process that uses the component process to deploy the component.

We are going to create a **component process** for this example. This process will automate deployment of the new version of our **J02Mortage COBOL** application.

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



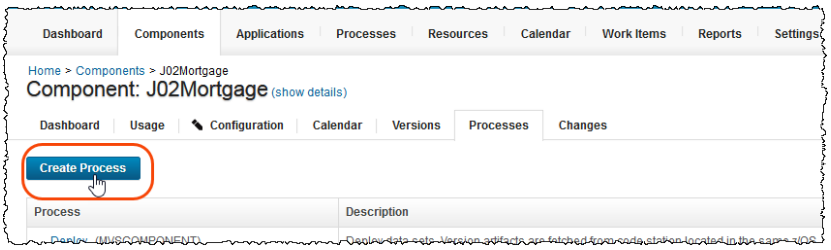
8.2 Click the **Processes** tab



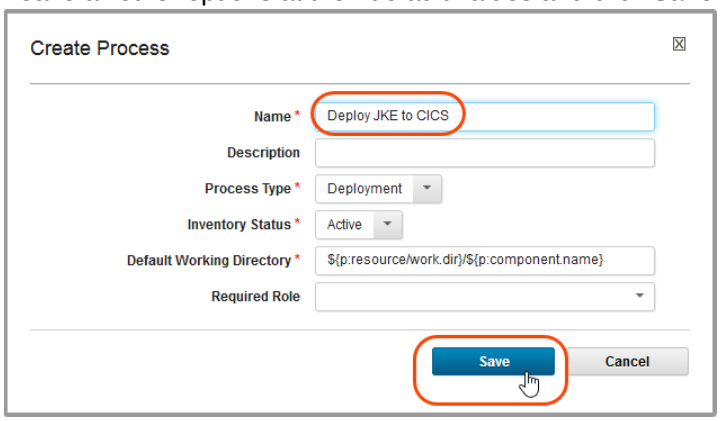
You will notice that there is a small collection of processes that was created automatically for your component (**Deploy, Remove all versions, Uninstall**, and so on). Remember that those processes were created from a template.
These processes can be used by themselves to perform simple tasks or can be used as starting points for more complex processes.

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

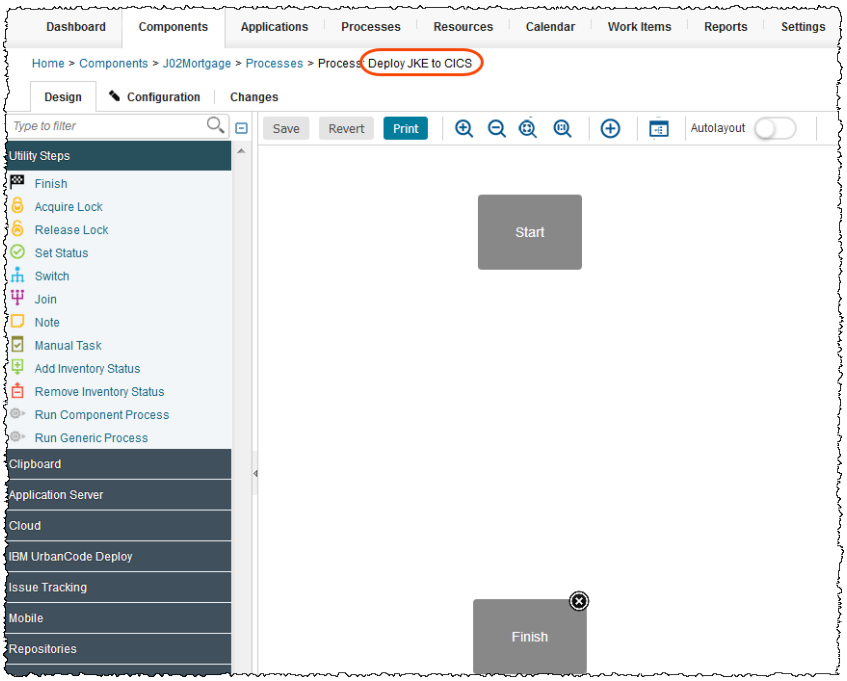
8.3 Click **Create Process** button to get started.



8.4 Enter Process Name (**`Deploy JKE to CICS`**) in the **Create Process** dialog.
Leave all other options at their default values and click **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::
http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.doc/topics/comp_workflow.html

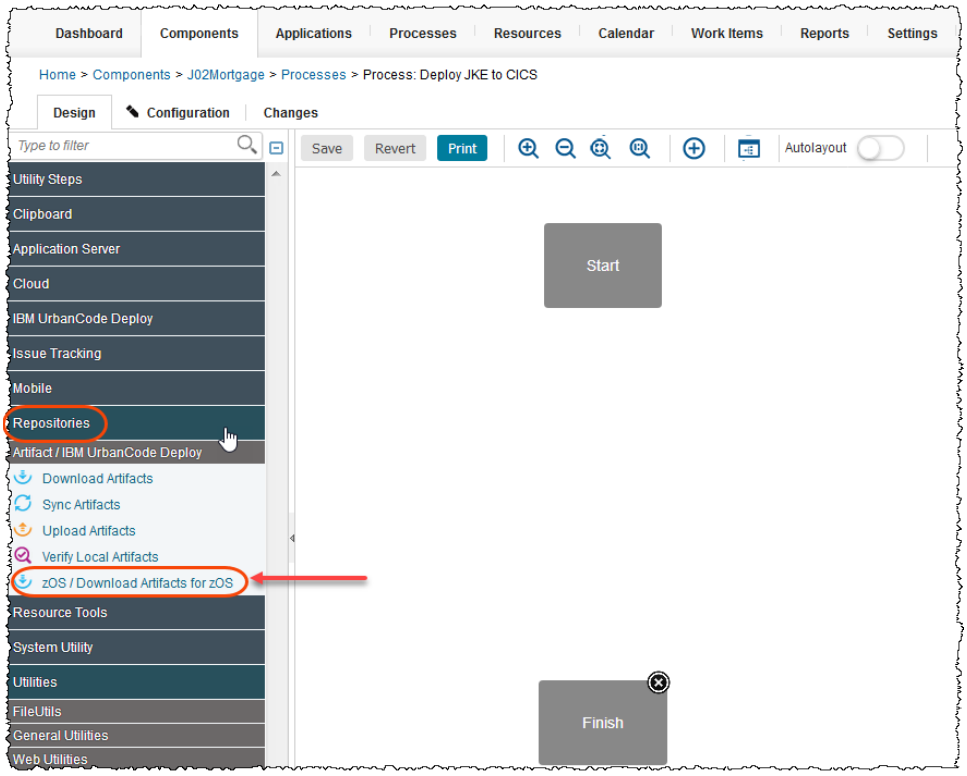


8.6 We are going to create a simple process for updating an existing CICS application.

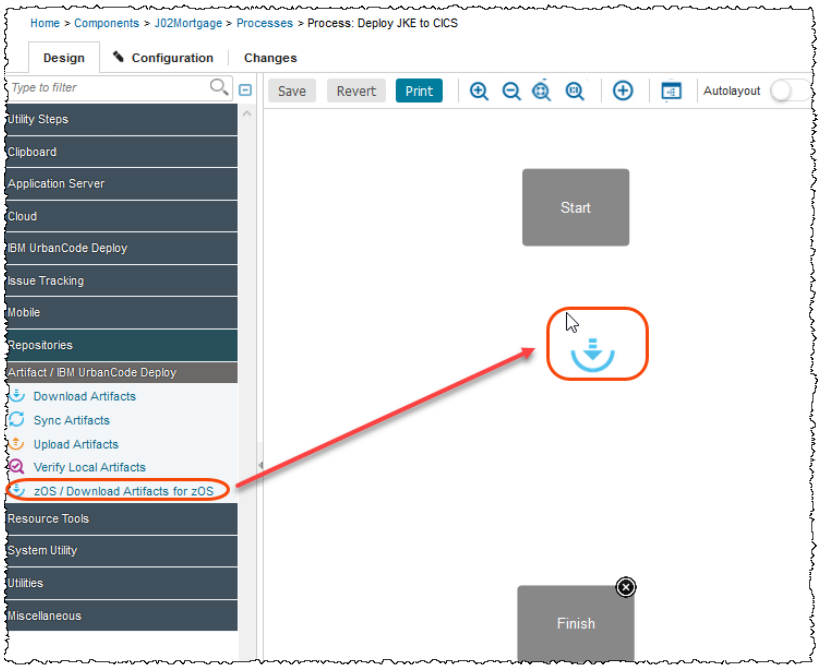
⏭ On the left hand side of the editor you will find an extensive **Palette** of process steps.

⏭ Scroll down and expand **Repositories** .

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

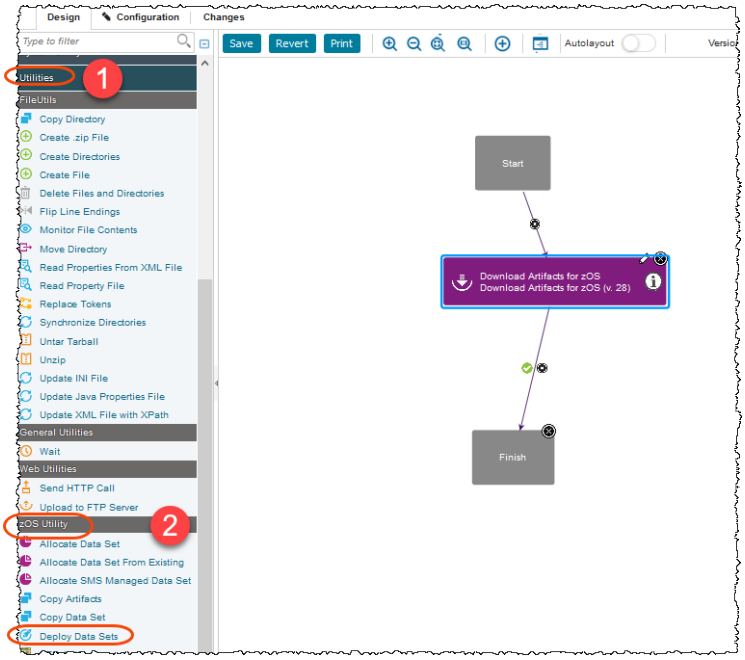


8.7 ⏭ Drag and drop the first step, **zOS /Download Artifacts for zOS**, onto the design space (somewhere under the **Start** block).

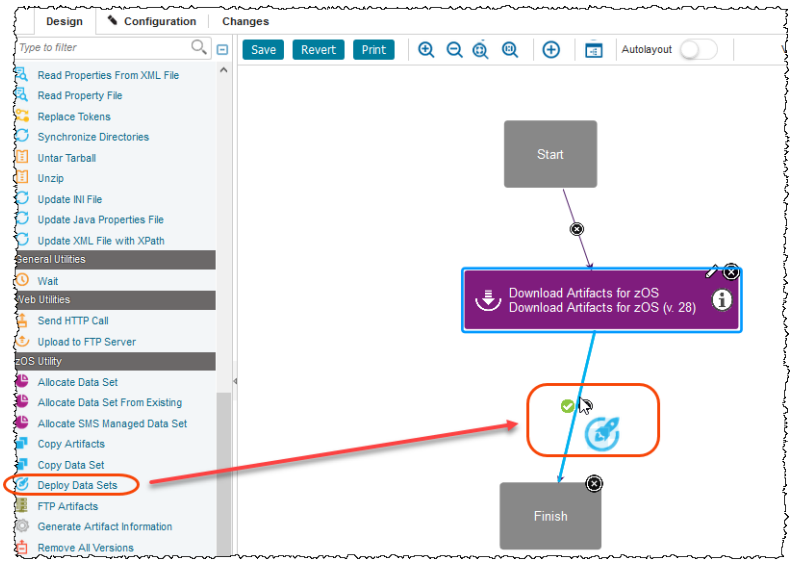


8.8 ⏭ 1 Using the Design palette on left expand **Utilities,**

2 scroll down and 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.

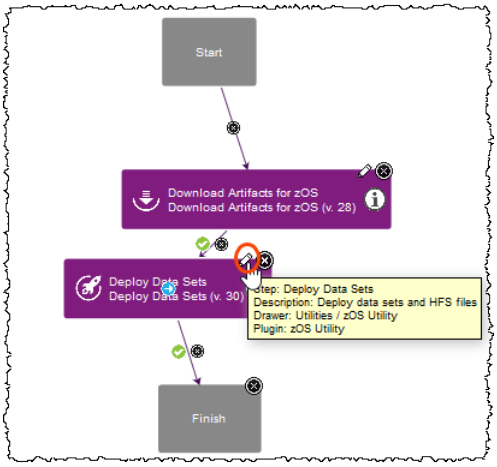


8.9 ⏭ Drag the **Deploy Data Sets** step onto the line that connects to the Finish as below



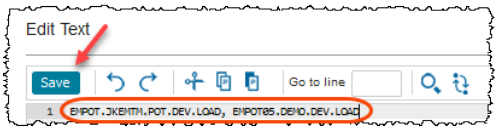
Properties
Properties can be set in UCD for many different things, including components, environments, processes, and applications. You can also set global properties for the system.
You can refer to a property by scope:
${p:scope/propertyName}
or without scope:
${p:propertyName}
For more details, refer to the full documentation on Properties:
http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.reference.doc/topics/ud_properties.html

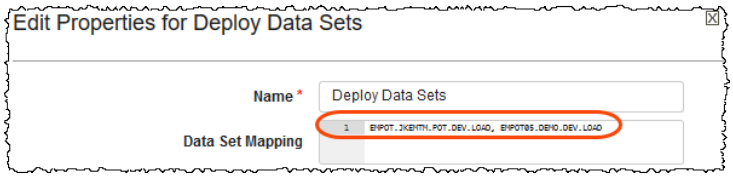
8.10 Click on the **pencil** icon on **Deploy Data Sets** to change the UCD properties for this step**.**



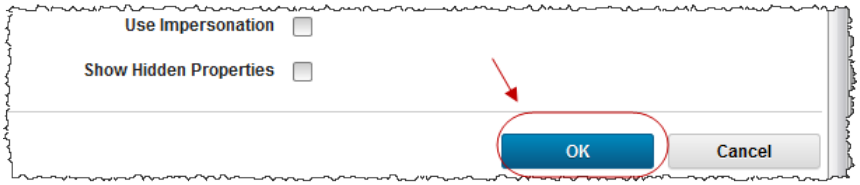
8.11 Enter the value for the *Data Set Mapping* property:
`EMPOT.JKEMTM.POT.DEV.LOAD, EMPOT05.DEMO.DEV.LOAD`

Data Set Mapping specifies which PDS members packaged with the component to deploy, and where to deploy them.
A mapping rule should follow format "From PDS, To PDS":
We could have that as variable but we will fix here to save time.

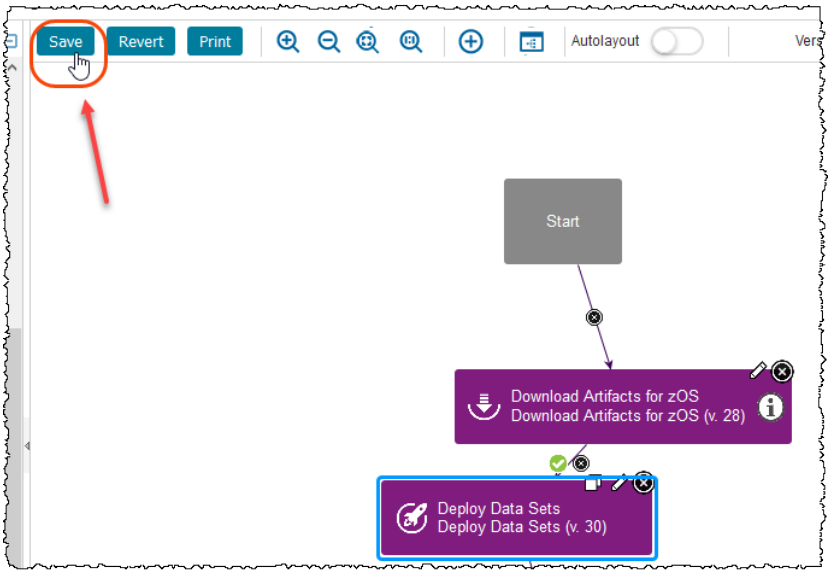




8.12 Click **OK** to save the properties updated

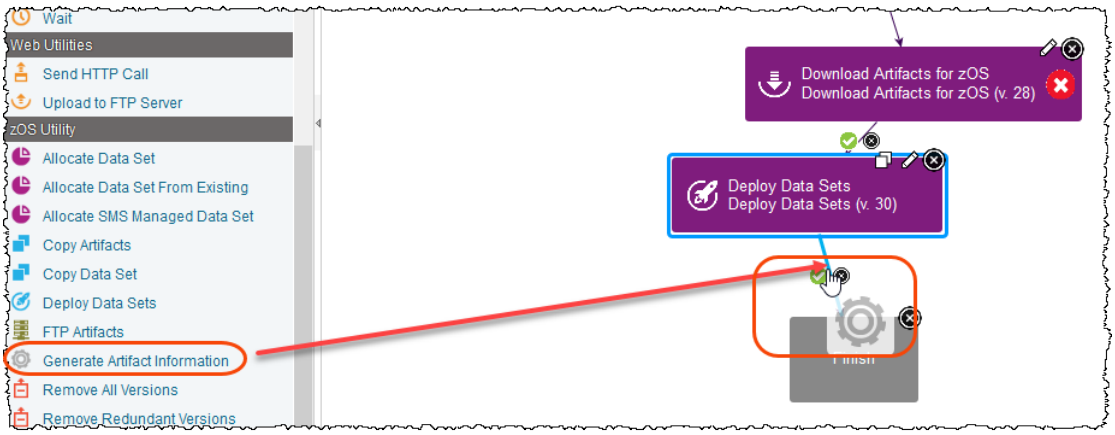


8.13 Click the **Save** button to save what you had done so far**.**

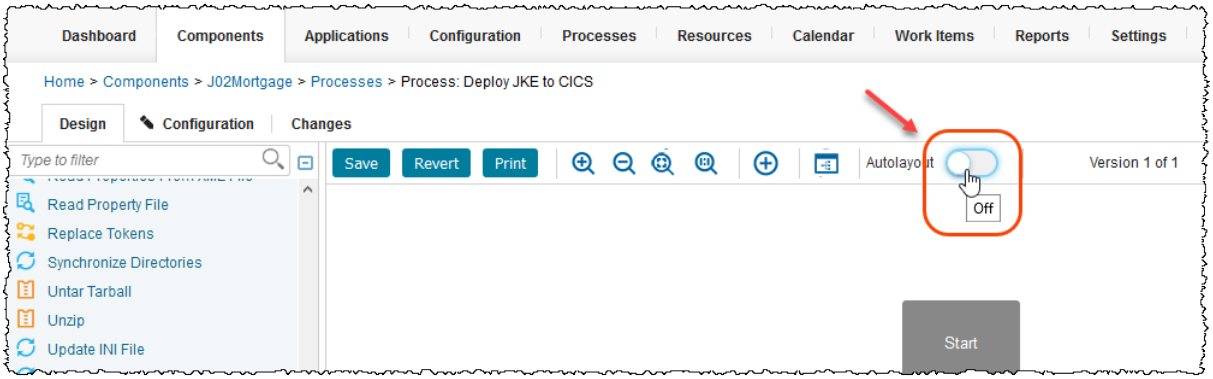


8.14 If there is a warning about missing property ignore it, we will fix that later.

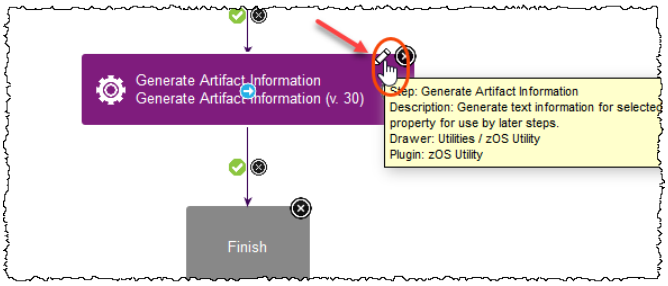
8.15 Scroll down on left pallet and drag and drop **Generate Artifact Information** on the **line** before Finish. You will need to specify a list of the load modules that CICS needs to do the *Newcopy*. This step generate text information for selected version artifacts. Information is put to output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



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



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



8.18 ⏭ Change the step Name from ***Generate Artifact** Information* to **Generate Program List** (must respect upper/lower case and spaces)

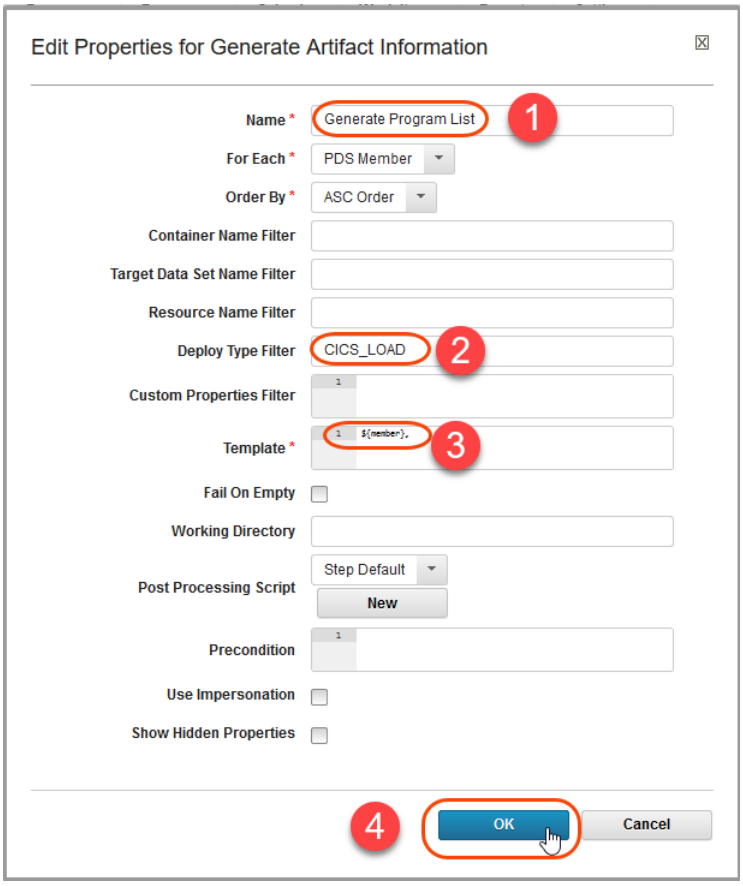
⏭ Update the Properties values with the values:

Use the values **CICS_LOAD** and **${member},**

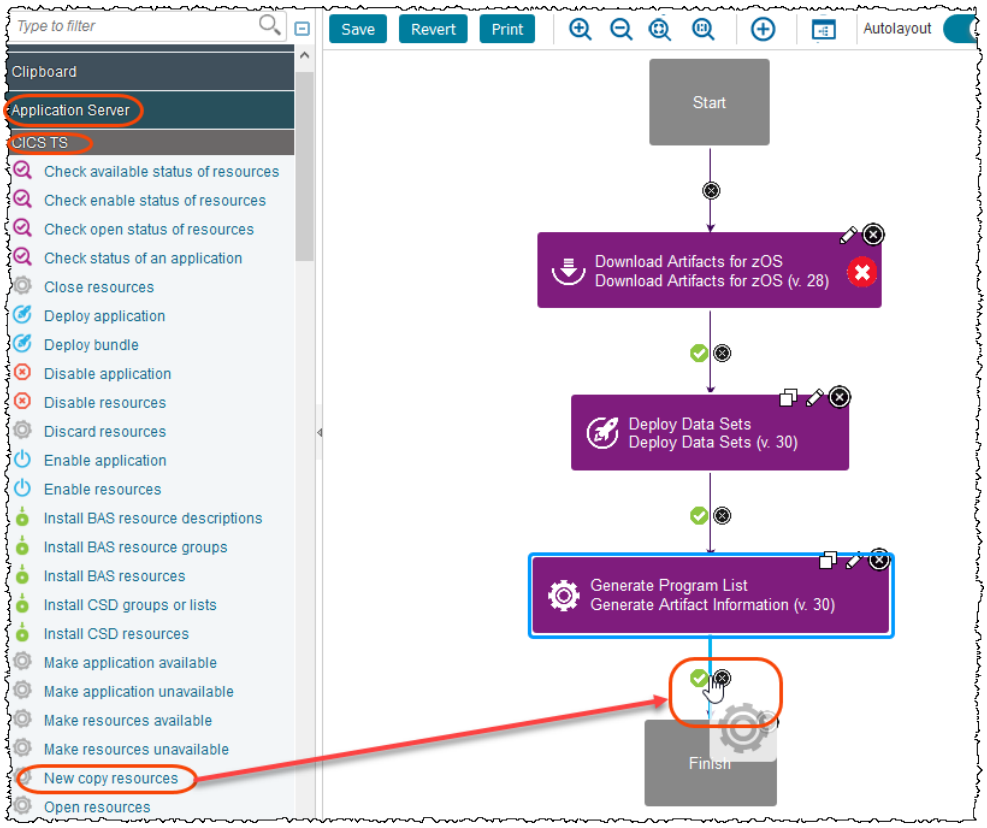
IMPORTANT ➔ Be sure that you add a , (comma) after the ${member},

Notice that the default is "`${dataset}(${member}),`" you need to modify that and click **Save**

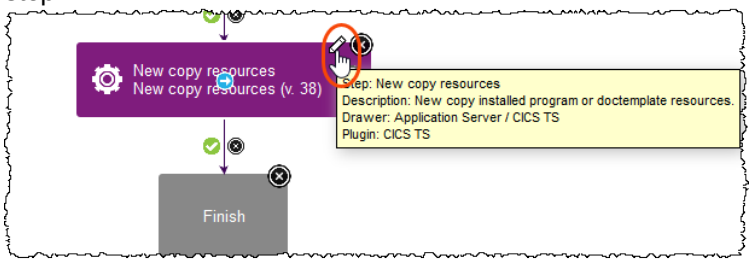
⏭ and click **OK**



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

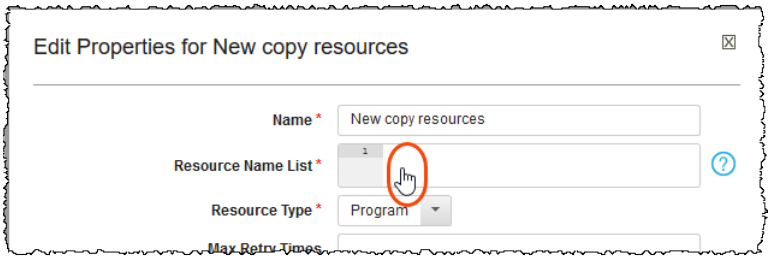


8.20 Click on the **pencil** icon t on **New copy resources** to change the UCD properties for this step.

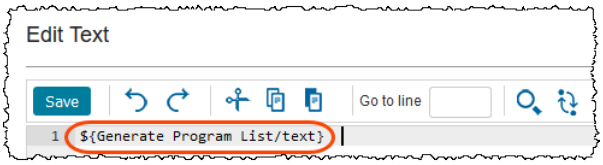


8.21 You need to specify a list of the load modules that CICS needs to do the *Newcopy*.

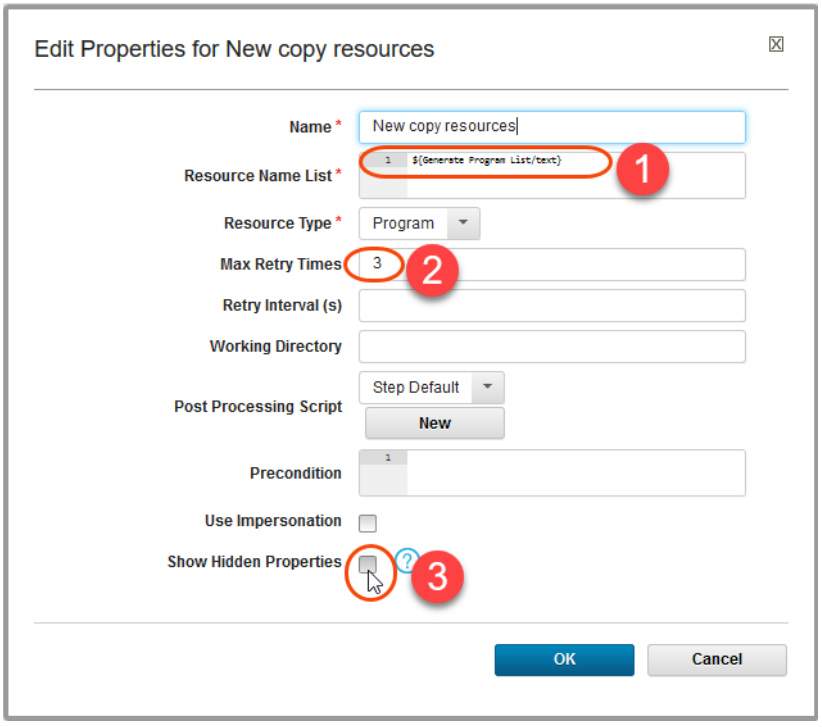
⏭ Change the Program List properties as
Resource Name List: **${Generate Program List/text}** (This is the step name that you created on 8.18)
On *Max Retry Times* specify **3**



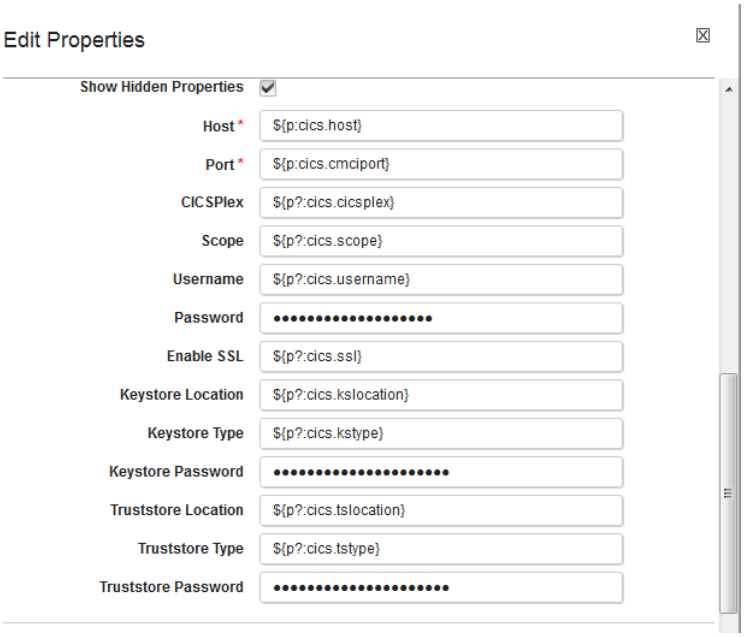
⏭ Type **${Generate Program List/text}** and click **Save**



⏭ Type **3 for Max Retry Times**

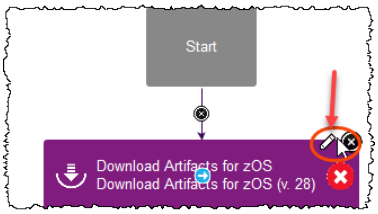


8.22 Click **Show Hidden properties**.
As you see those properties need to be filled. Later we will do that at the **Test** level

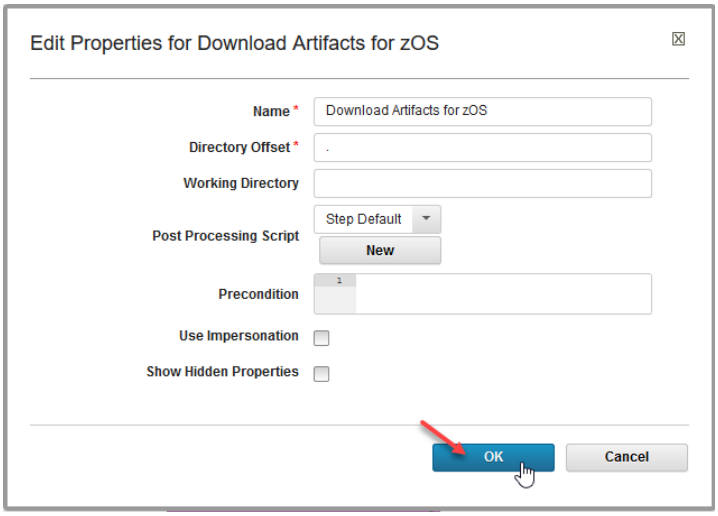


Click **OK** to save it

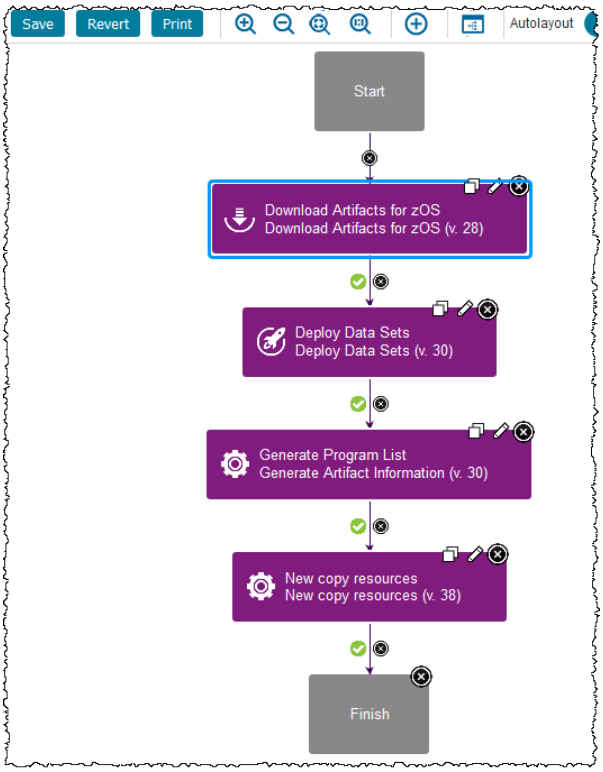
8.23 You still need to fix a warning on first step (see the icon).
Click on the **pencil** icon on **Download Artifacts for zOS** to possibly change the properties.



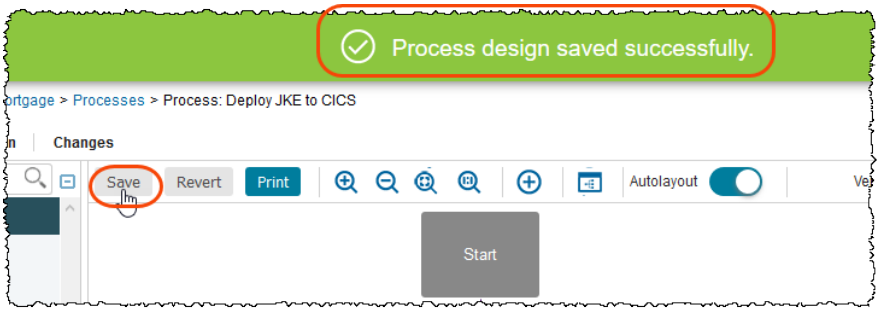
8.24 No changes are required here and click **OK**



8.26 The final result should look similar to this:



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



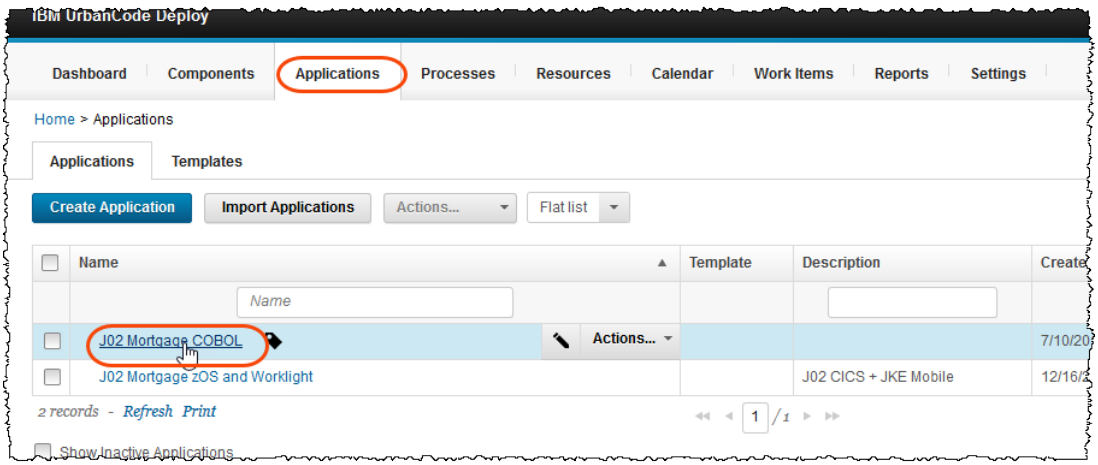
A message will indicate that the process is saved.

Task 9 - Create an application process

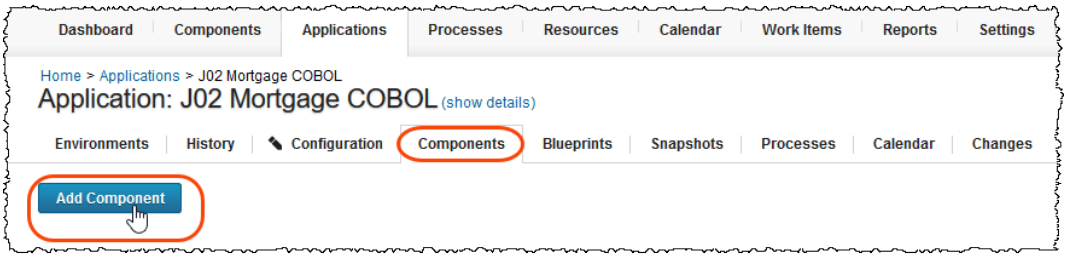
Application processes direct underlying component processes and orchestrate multi-component deployments. An application process, like a component process, consists of steps that are configured with the process editor.

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

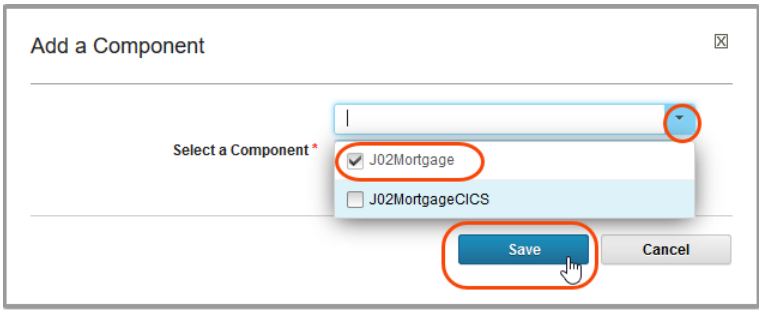
9.1 ⏭ Go to **Applications** page, select your application (**J02 Mortgage - COBOL**),



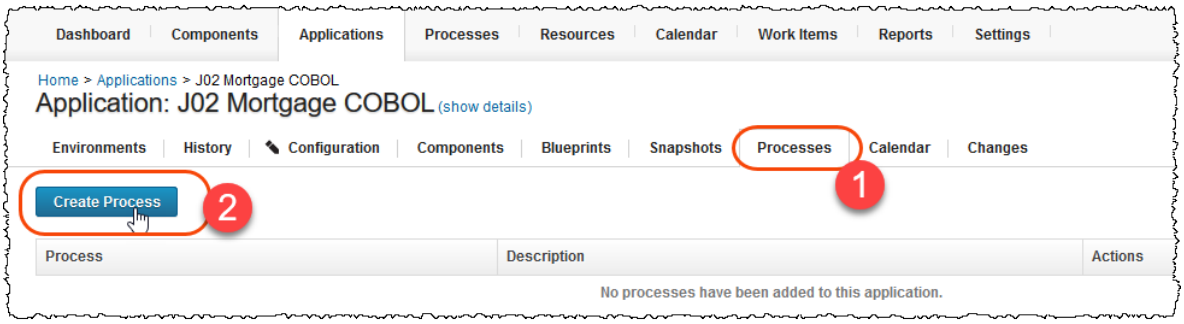
9.2 ⏭ Click **Components** tab and click **Add Component**:



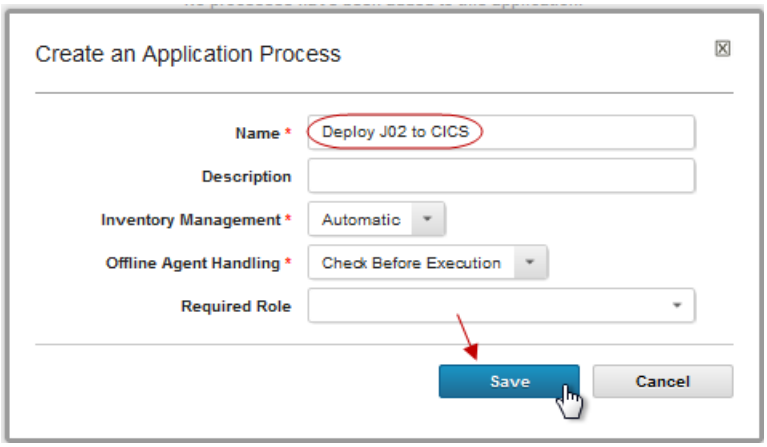
9.3 ⏭ Select your UCD component created previously (**J02Mortgage**) and click **Save**:



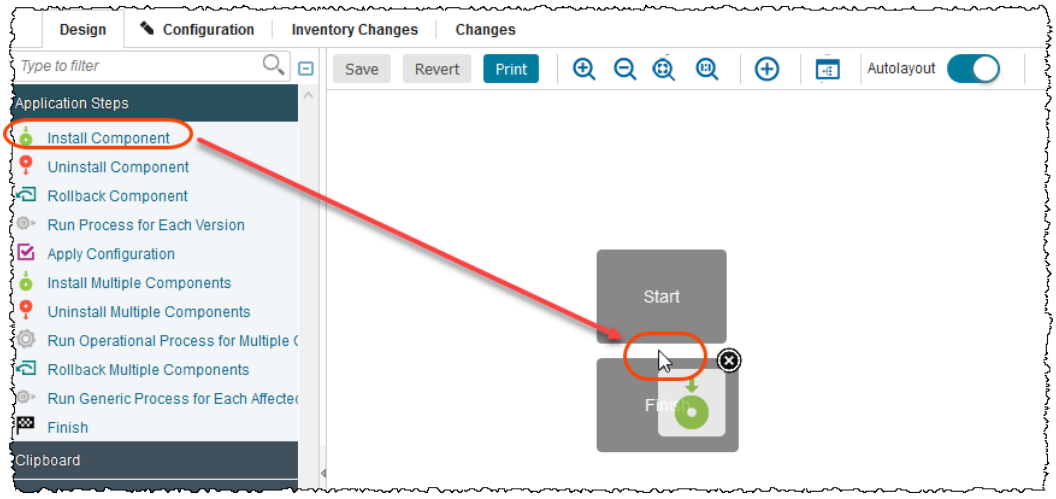
9.4 Click the **Processes** tab and click **Create Process**:



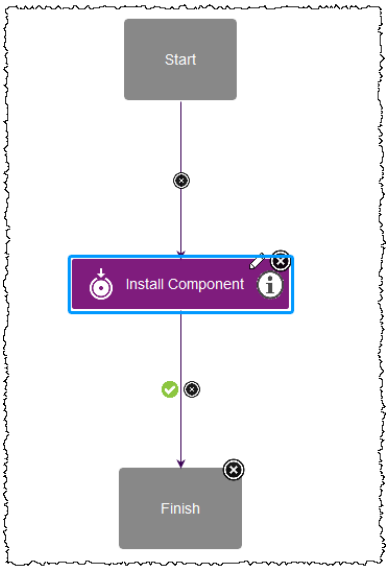
9.5 Give a name to your process, Like **Deploy J02 to CICS** and click **Save**:



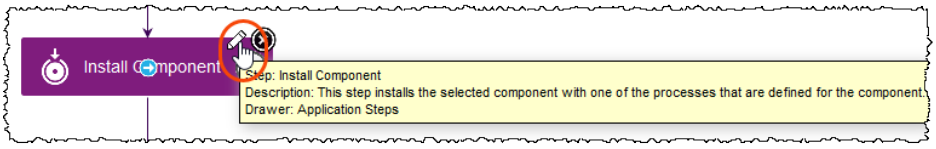
9.6 Find **Install Component…** step in the Design Palette, and drag it onto the space between Start and Finish boxes.



9.7 The result will be as below:

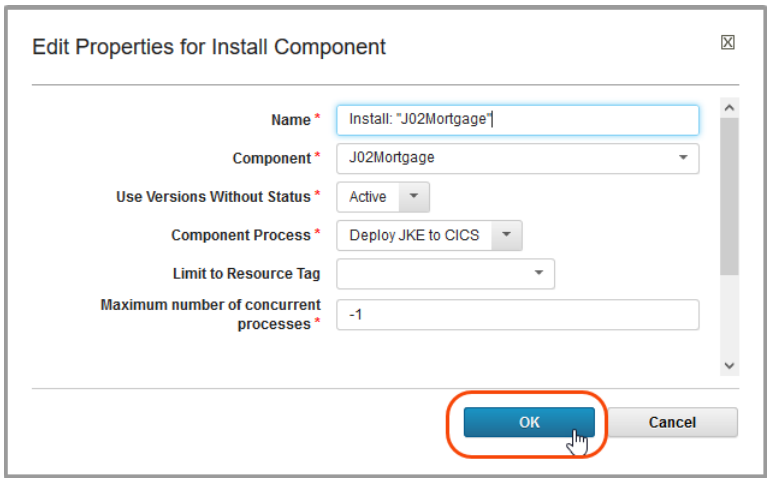


9.8 Click on the **pencil** icon on **Install Component** to possibly change the properties



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

Click **OK**:

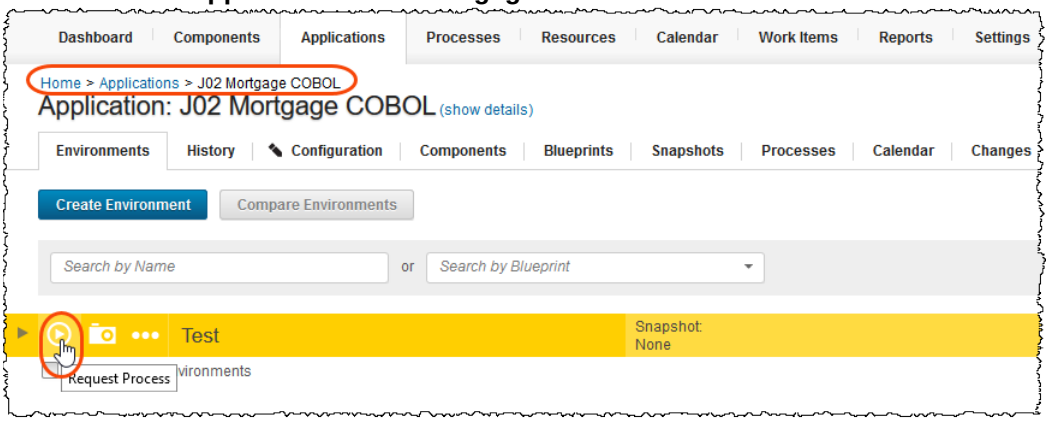


9.10 Click **Save**

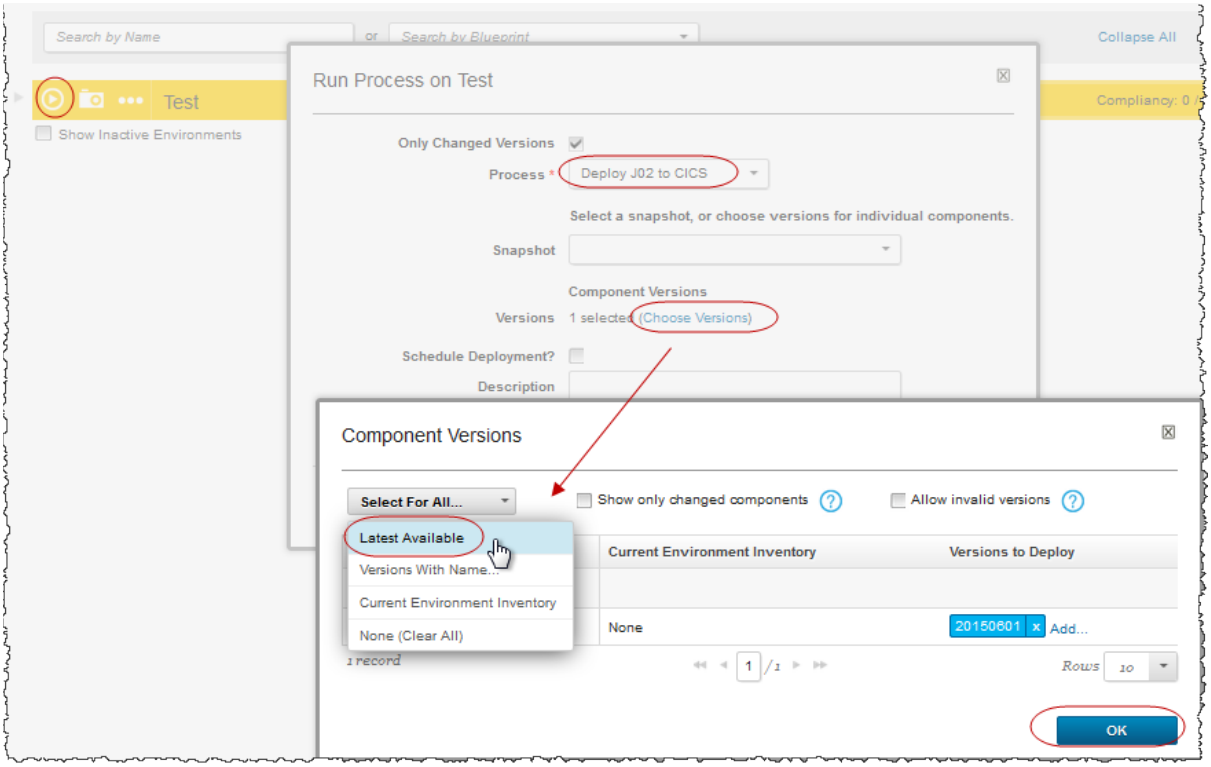
Task 10 – Environment properties configuration

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..

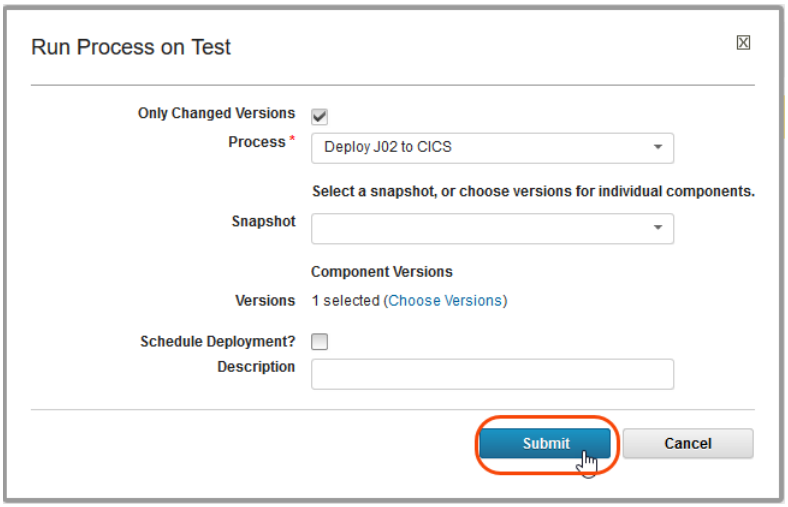
10.1 Go to **Application > J02 Mortgage COBOL** and click the icon on **Test** environmnet



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

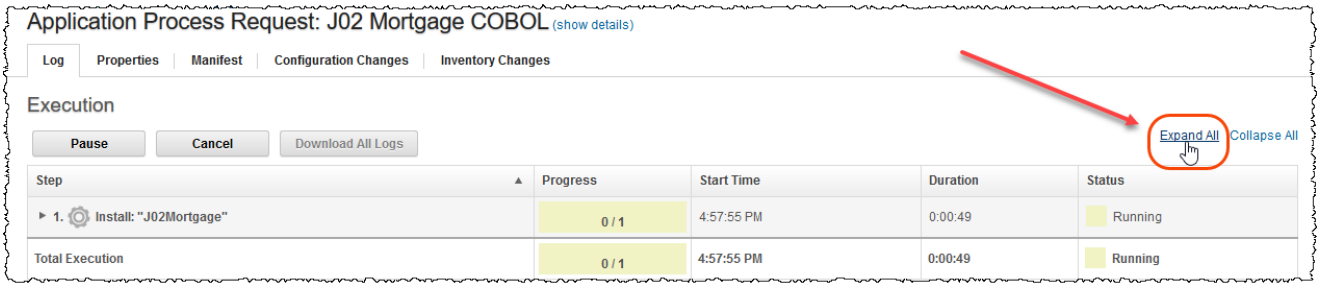


10.3 When the dialog below is shown click **Submit**

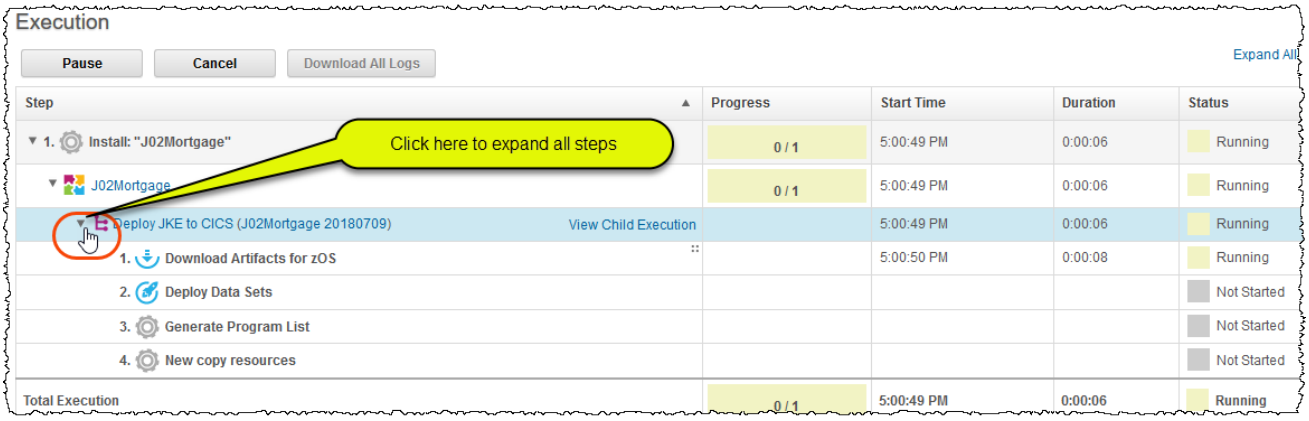


The Deploy process start, but it will fail as you see soon..

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

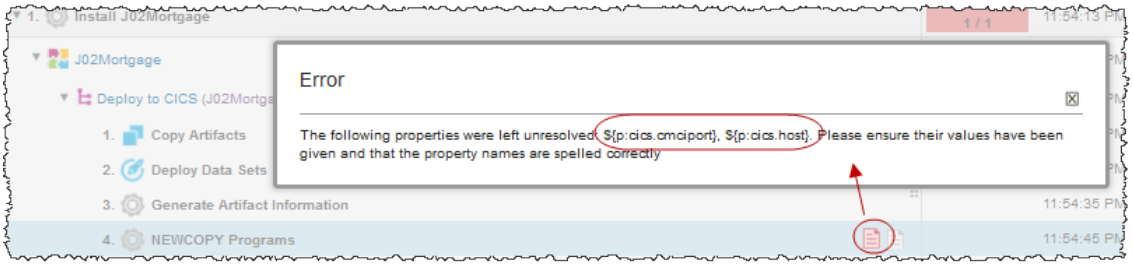


10.5 Click the icon ▸ to expand all steps being executed



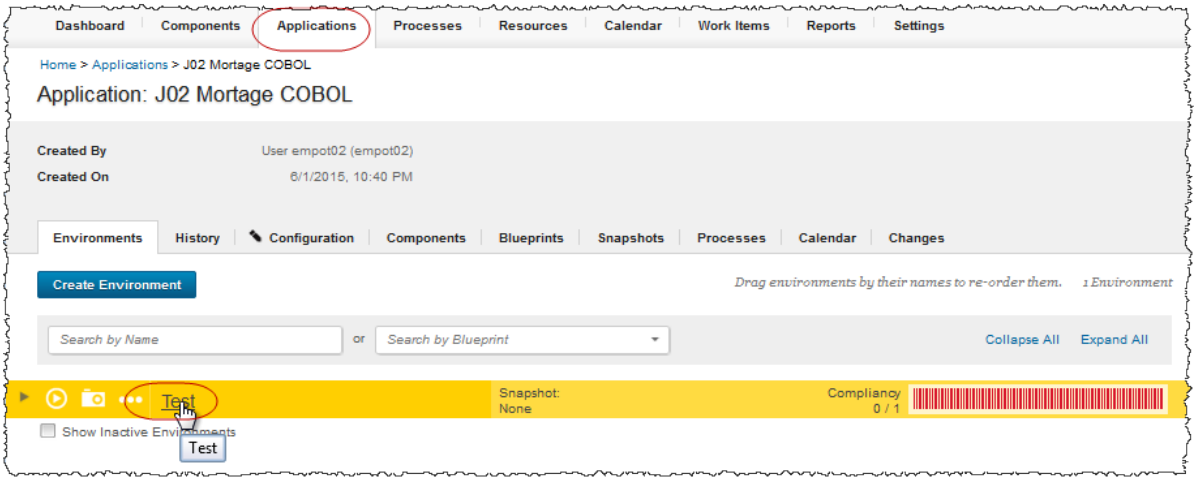
10.6 ⏭ Click on **Error Log** icon

10.7 ⏭ See the properties missing and **close the dialog**

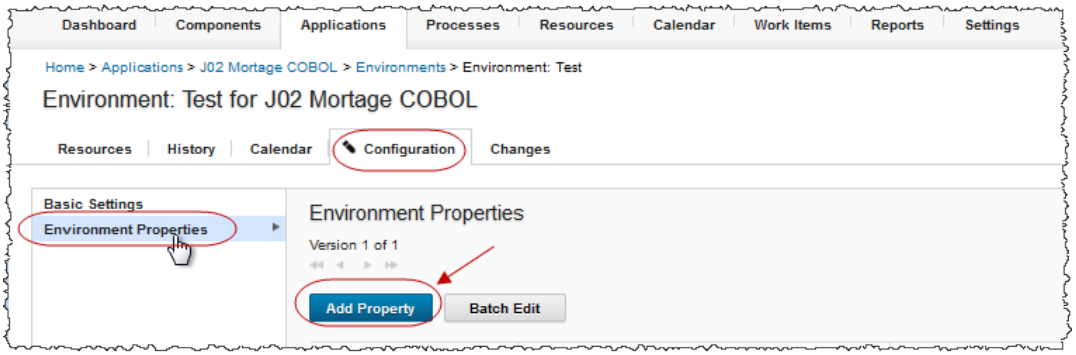


10.8 We could add properties at various levels.. In our example let's make the variables at the Environmnet level. The missing properties are:
cics.cmciport and **cics.host.**

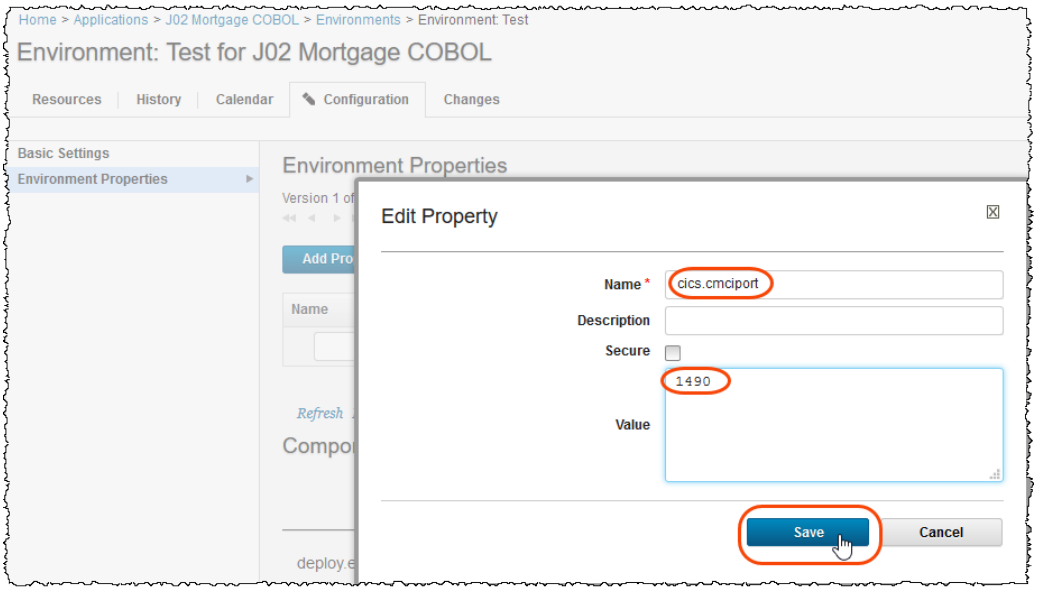
10.9 To add the properties..
⏭ Click **Applications > J02 Mortgage COBOL > Test**



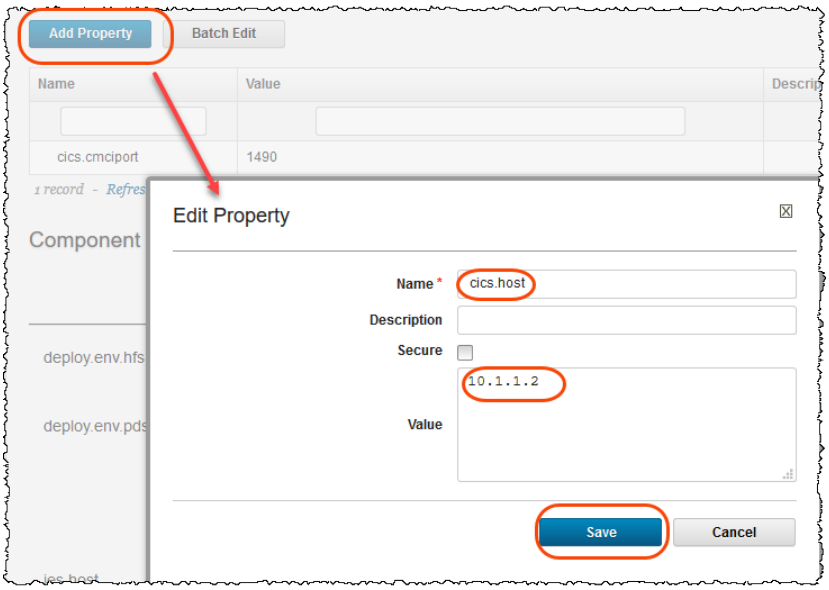
10.10 To add properties: select **Configuration** > **Environment properties** and click **Add Property**



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



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

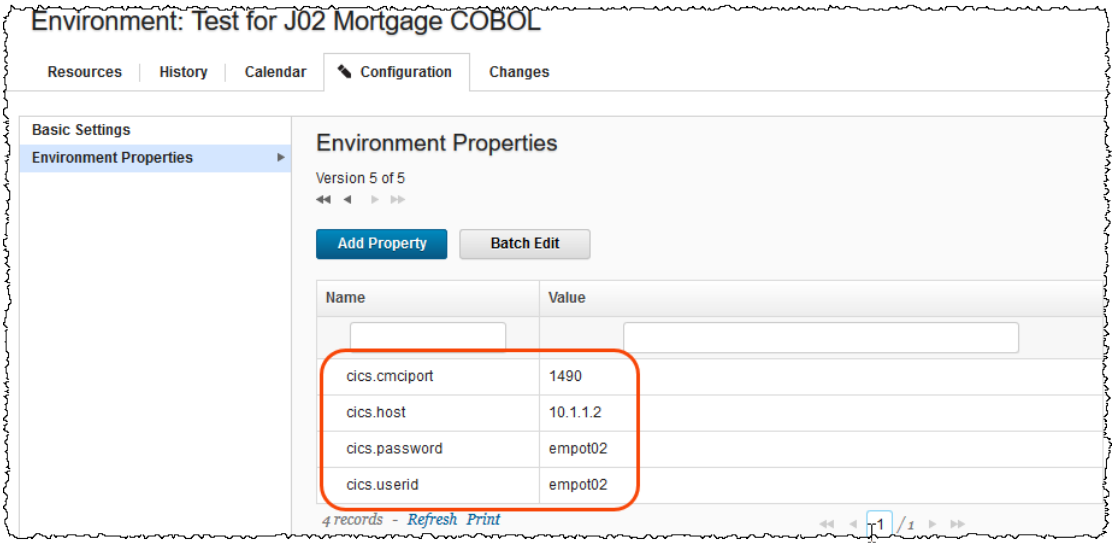


10.13 Add the property **cics.userid** with the value **empot02** (your userid)

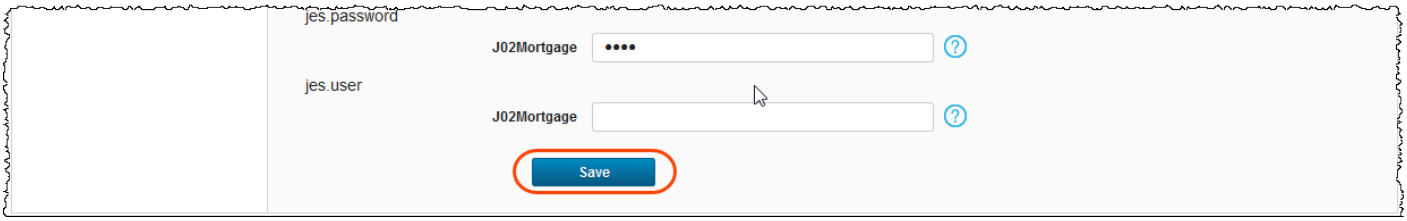
10.14 Add the property **cics.password** with the value **empot02**.

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:



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



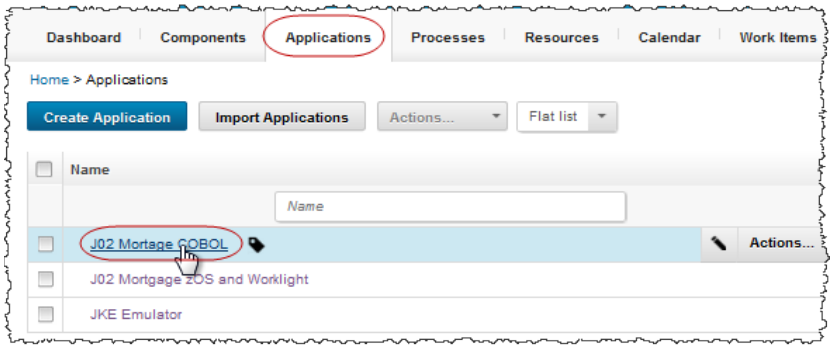
PART 3 – Deploy the application to z/OS CICS

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.

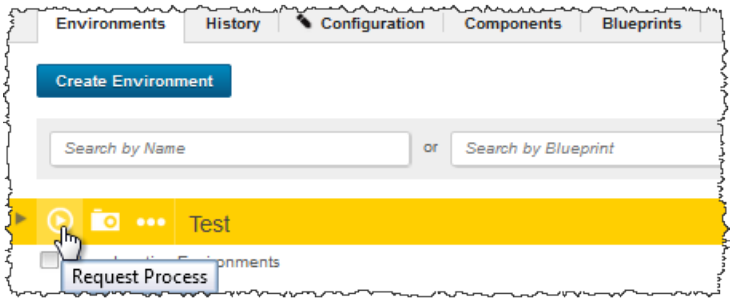
Task 11 – Run an application process

You are now ready to test our deployment process.

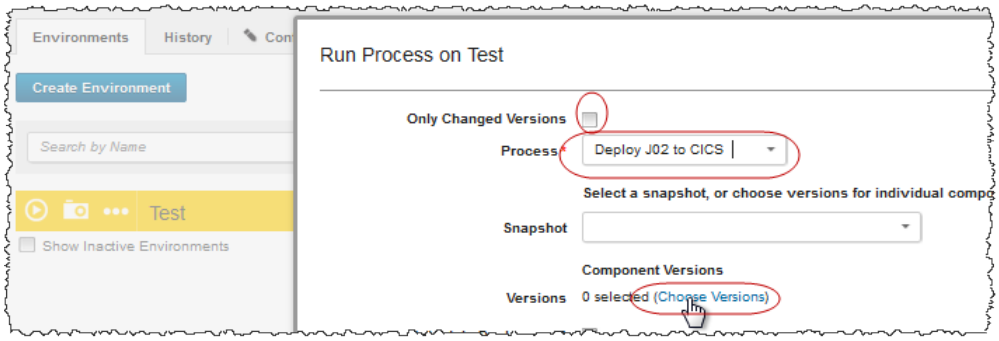
11.1 Click the **Applications** tab and then click **J02 Mortgage COBOL**



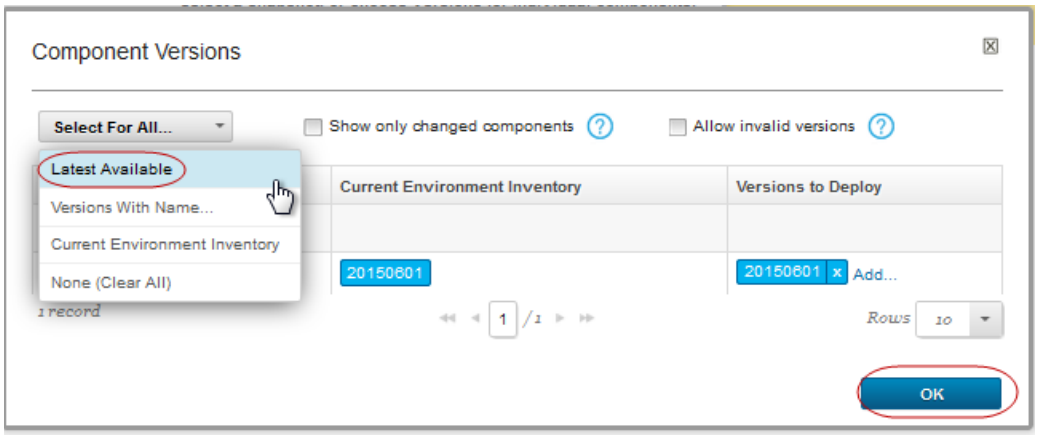
11.2 Under **Test** environment click the **Request Process** icon



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**

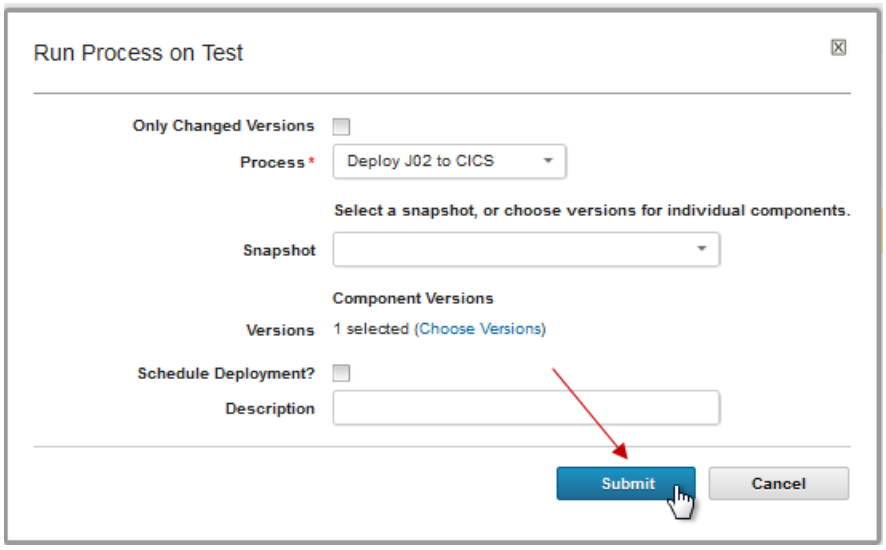


11.4 At the top of the window, click **Select For All** > **Latest available** click **OK**



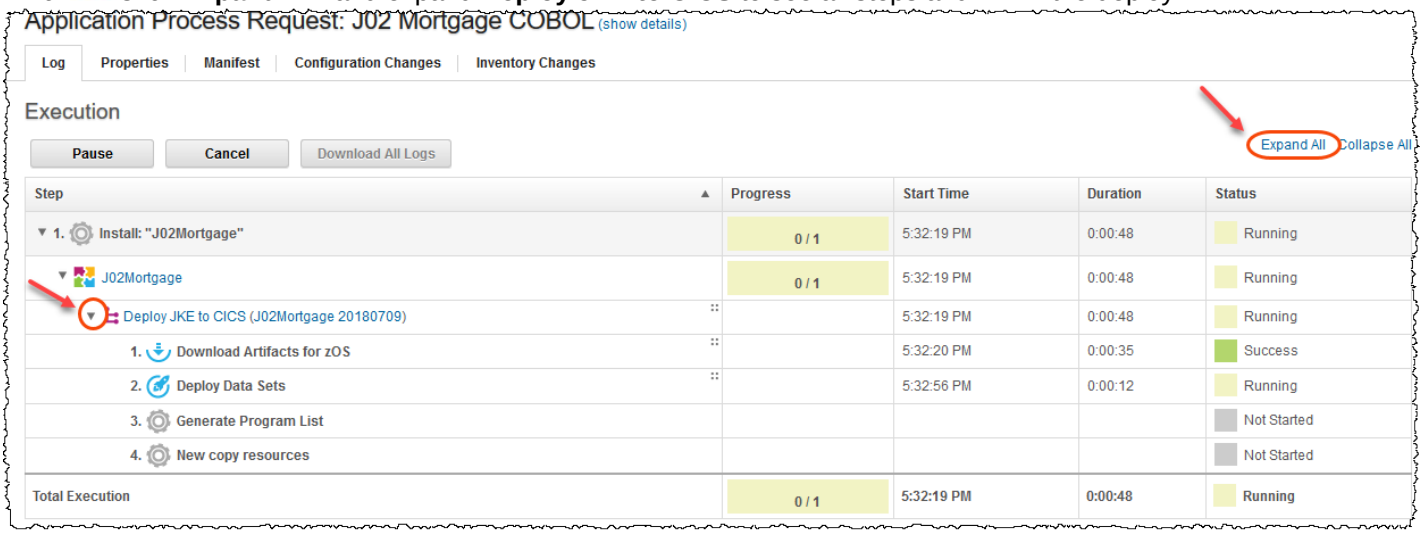
Make sure that the version that you created on PART 1 of the lab is selected for **J02Mortgage**
If you do not select a version, that component is not deployed.

11.5 Click **Submit**

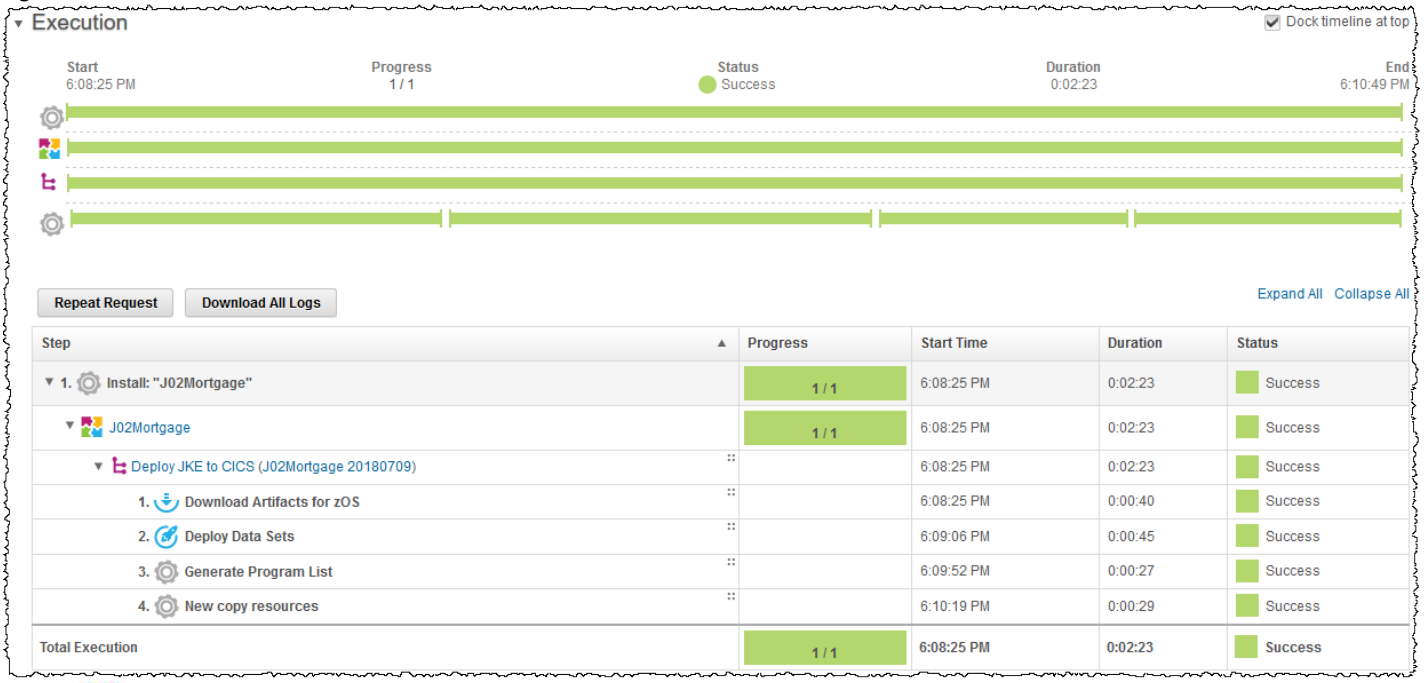


The web page shows you the progress of the application process request. From this page, you can watch as the processes run. The following figure shows the application process partially completed. The application component process is finished and the other two component processes are running.

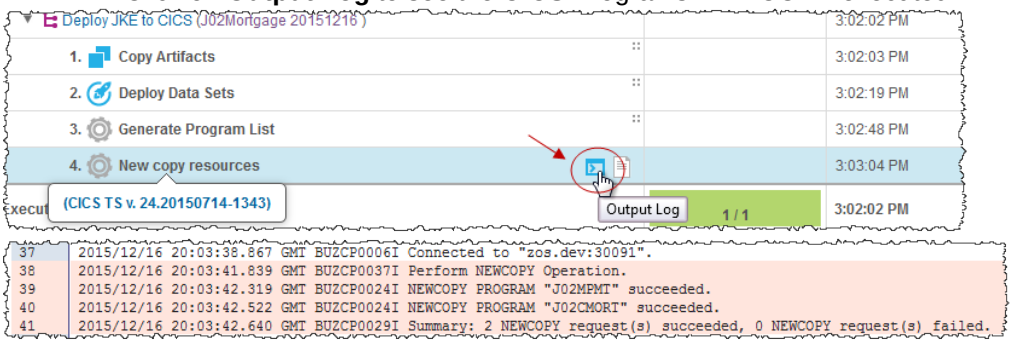
11.6 Click **Expand All** and expand **Deploy JKE to CICS** to see all steps and WAIT the deploy...



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



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




```
37  2015/12/16 20:03:38.867 GMT BUZCP0006I Connected to "zos.dev:30091".
38  2015/12/16 20:03:41.839 GMT BUZCP0037I Perform NEWCOPY Operation.
39  2015/12/16 20:03:42.319 GMT BUZCP0024I NEWCOPY PROGRAM "J02MPMT" succeeded.
40  2015/12/16 20:03:42.522 GMT BUZCP0024I NEWCOPY PROGRAM "J02CMORT" succeeded.
41  2015/12/16 20:03:42.640 GMT BUZCP0029I Summary: 2 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.
```

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.

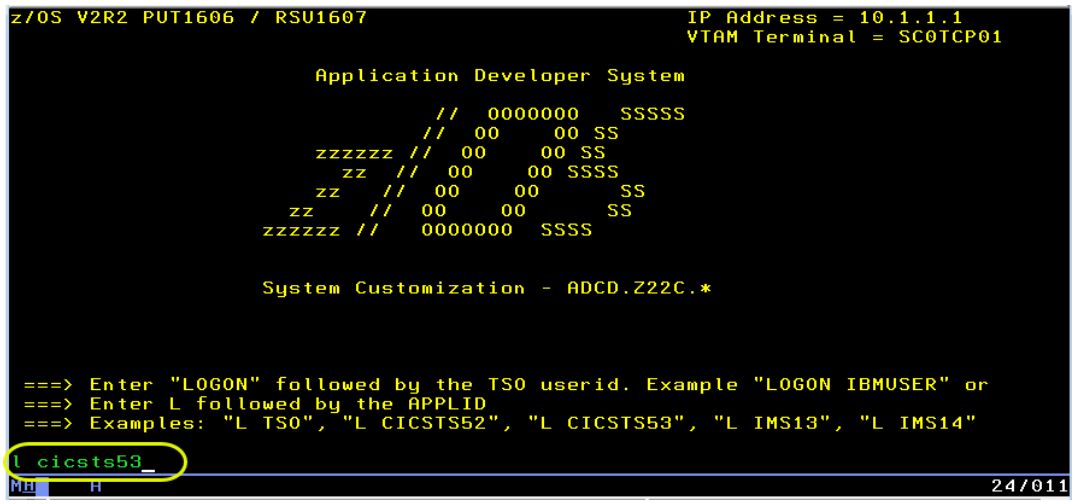
12.1 Use the 3270 emulator

Double click the icon  below on the desktop

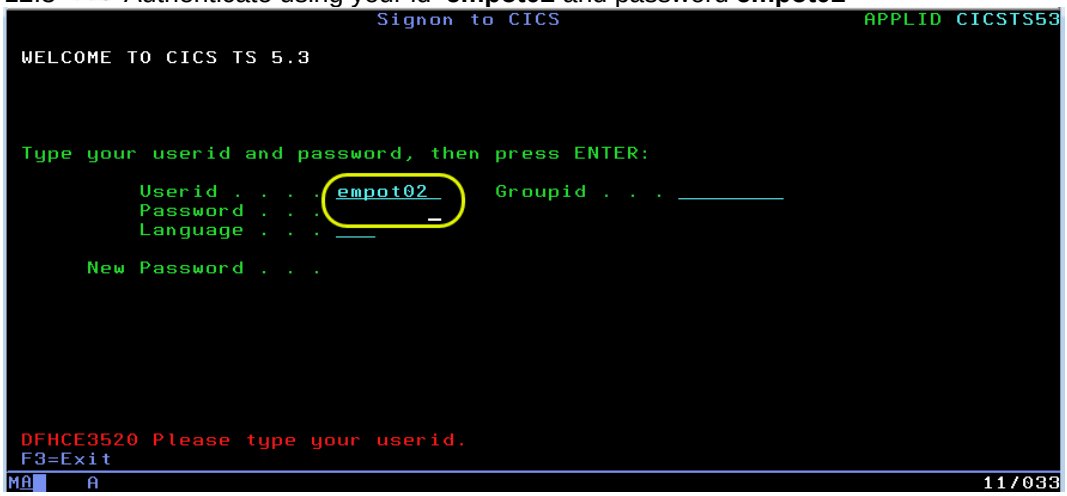
12.2 Verify the CICS application.

Type **l cicsts53**

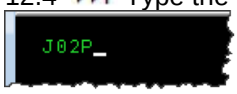
:



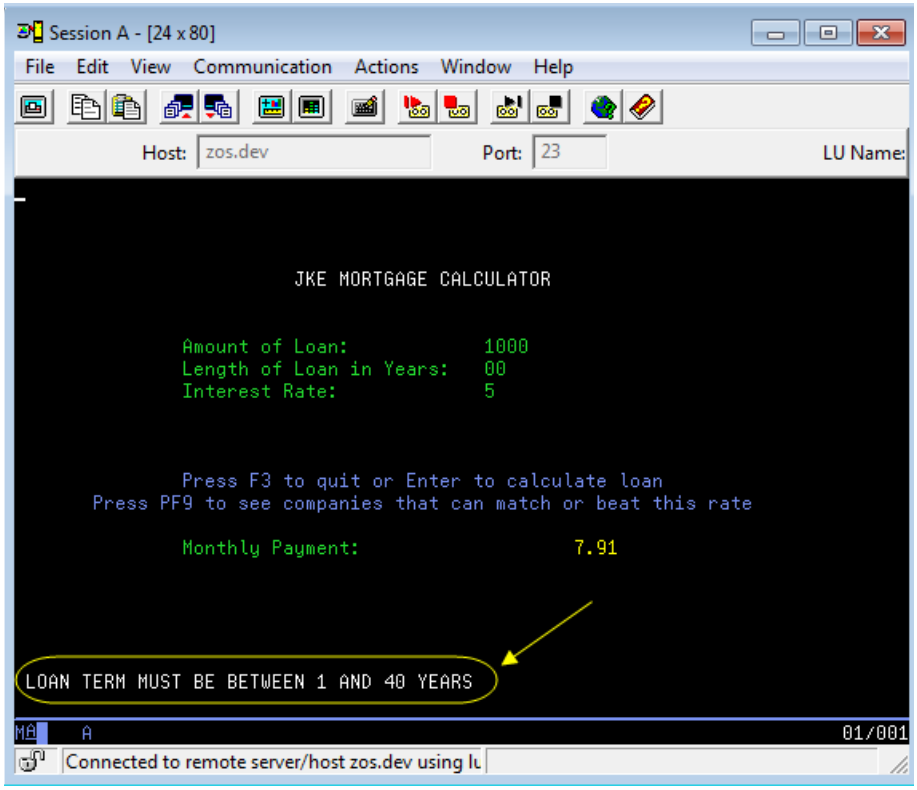
12.3 Authenticate using your id **empot02** and password **empot02**



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 7 – Using IBM Dependency Based Build with Git, Jenkins and UCD on z/OS (60 minutes)

Updated August 26 2019 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.

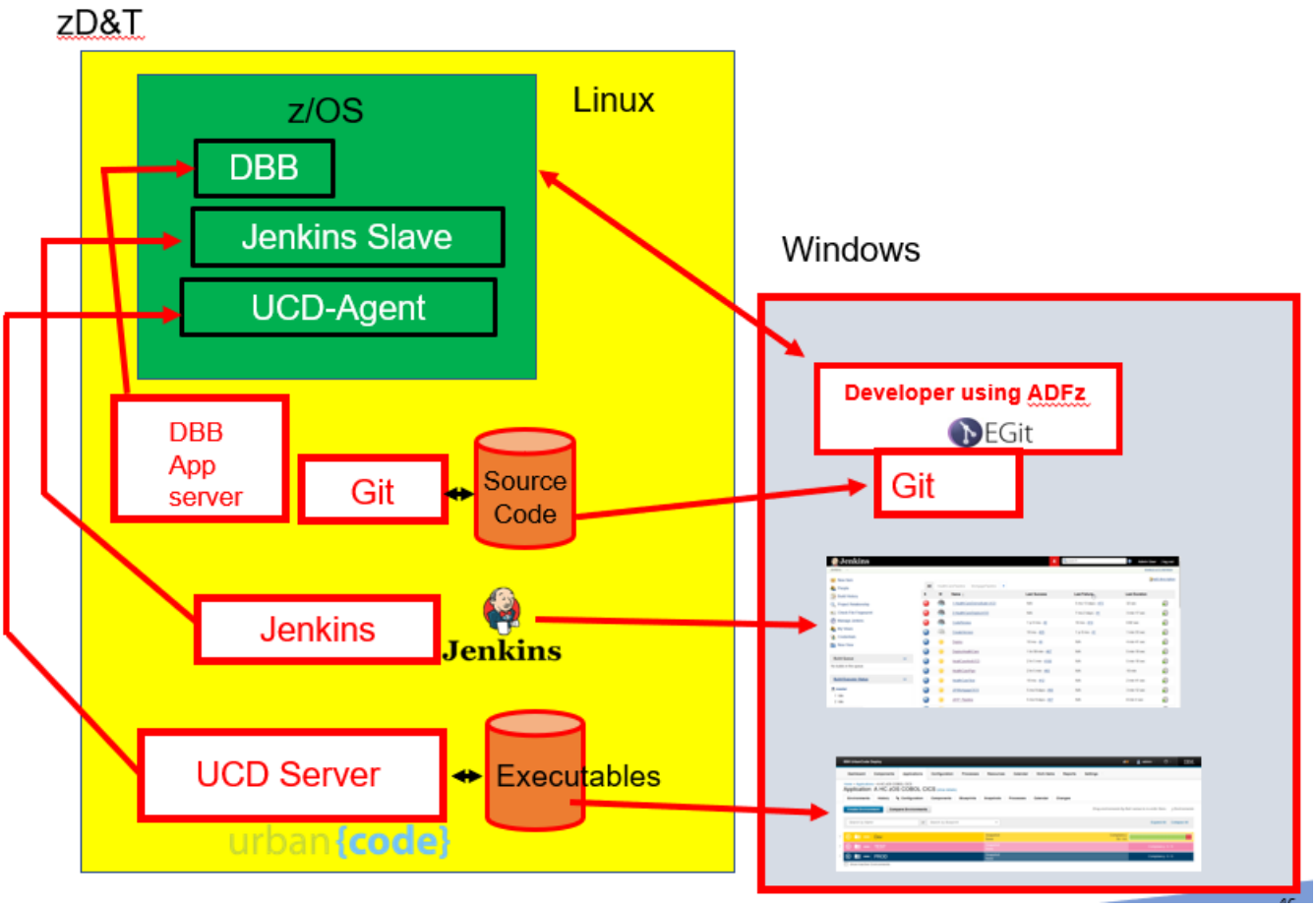
The updated code will be deployed to CICS using *UrbanCode Deploy* (UCD)

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_1.0.1/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.



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 *CICS* 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 *CICS* 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 *CICS* 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 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.

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_POT\LAB7** to copy/paste values if you don't want to type).

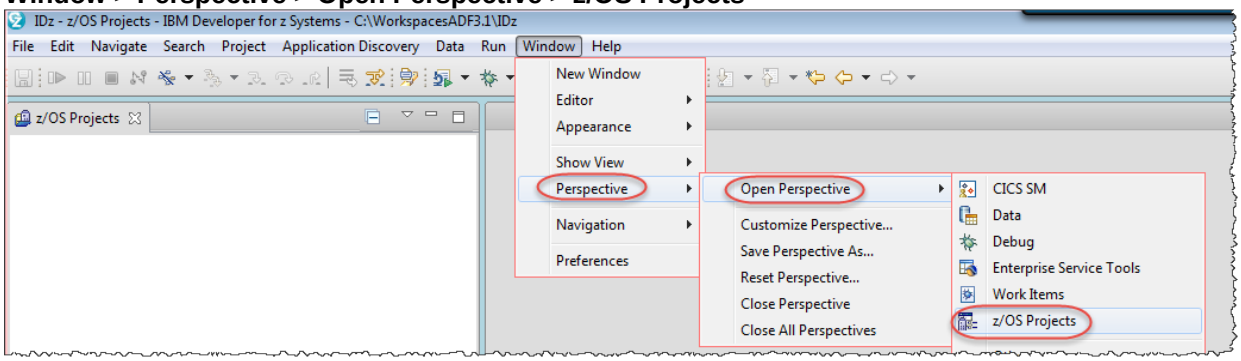
___ 1.1 Connect to z/OS and emulating a CICS 3270 terminal

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

1.1.1 Start *IBM Developer for z Systems* If not already started..

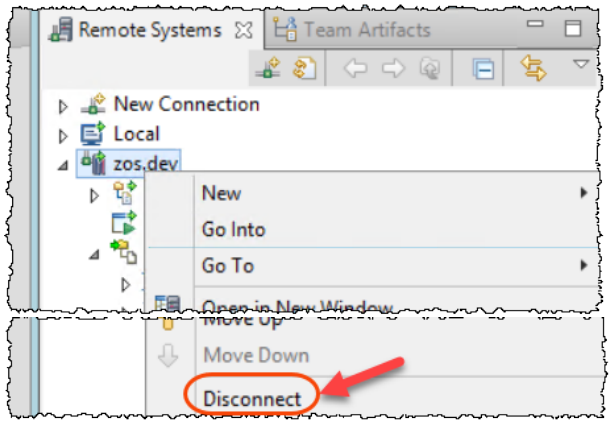
⏭ Double click on **IDz 14.1** icon Be sure that you clicked the icon on the Windows desktop.

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

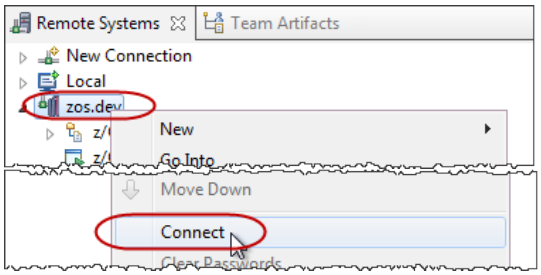


1.1.3 If you used userid **empot01** before, you need to disconnect now. Your new userid now will be **empot05**.

⏭ If you are connected, right click on **zos.dev** and select **Disconnect.** Otherwise go to step 1.1.4.



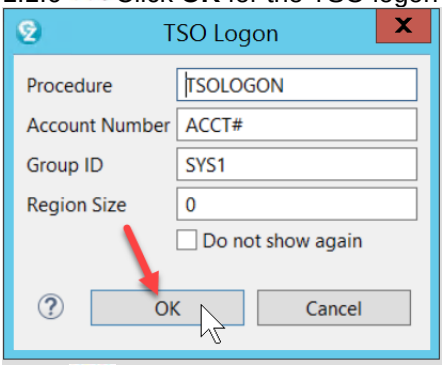
1.1.4 Right click on **zos.dev** and select **Connect**



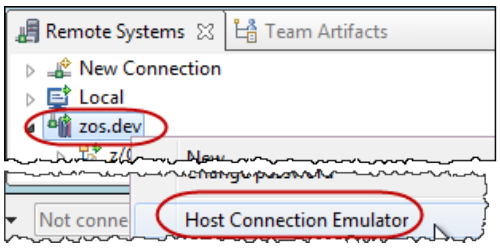
1.1.5 Type **empot05** as userid and **empot05** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.

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

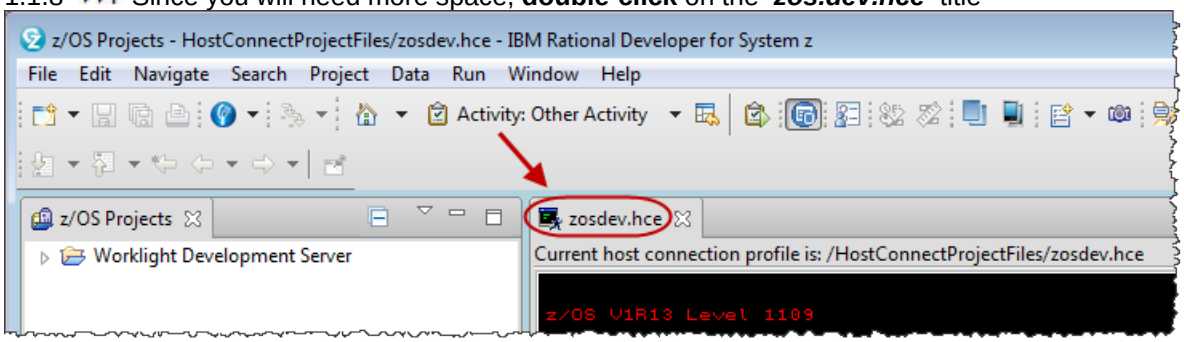
1.1.6 Click **OK** for the TSO logon dialog



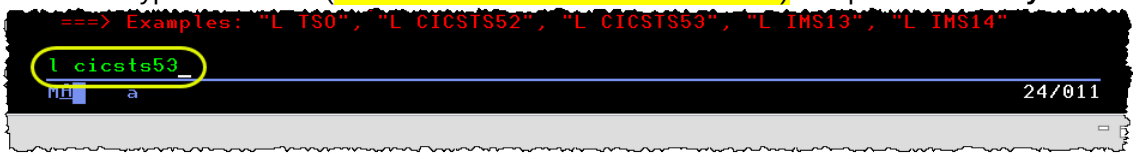
1.1.7 Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



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

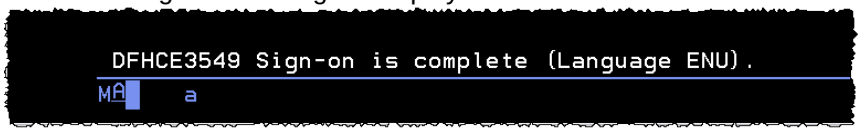


1.1.9 Type **l cicsts53**. (where "l" is the lower case of letter "L") and press **Enter key**.



1.1.10 Logon using your z/OS user id and password (**empot05)** and press **Enter**.

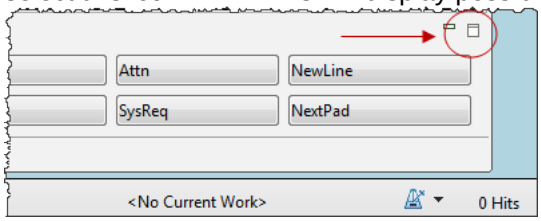
1.1.11 The sign-on message is displayed



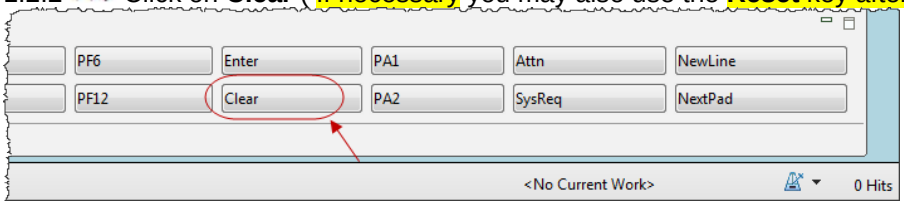
___ 1.2 Run CICS transaction J05P

You should now be in the z/OS CICS region named *CICSTS53*. This is the CICS instance where you will make the program changes.

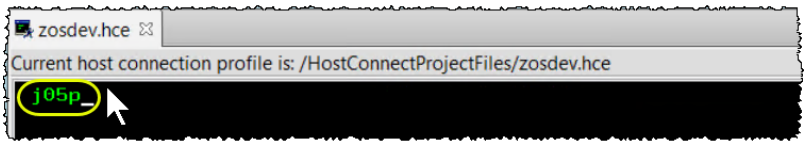
1.2.1 You may need to use the **clear** key. If the clear button is not displayed, look in the right lower corner, select this icon . This will display possible keys, including the clear button.



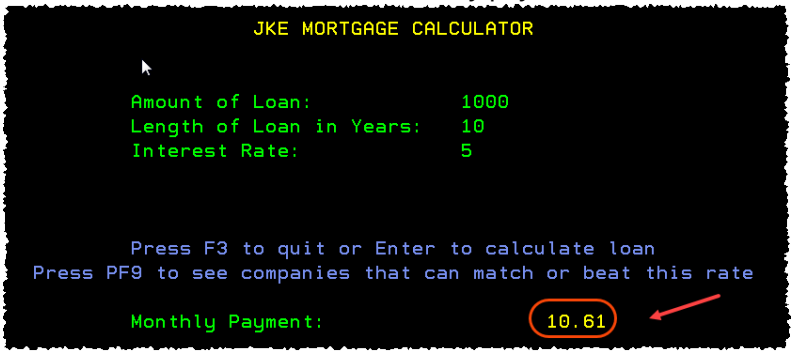
1.2.2 Click on **Clear** (If necessary you may also use the **Reset** key after clicking **NextPad**)



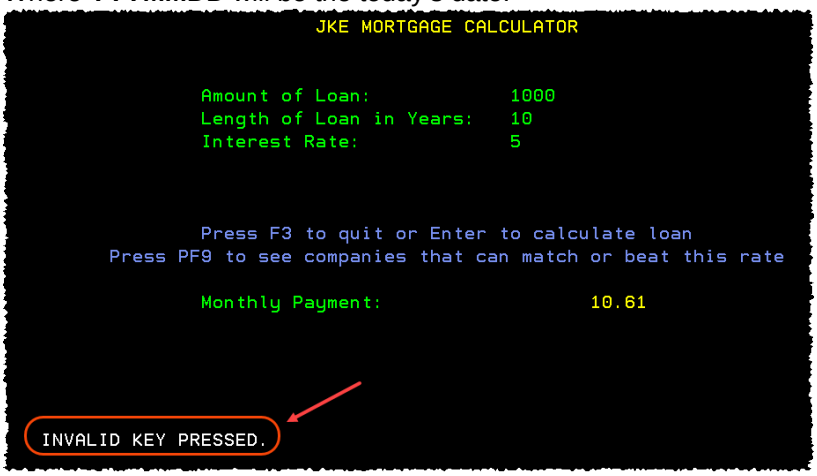
1.2.3 Type the CICS transaction **j05p** and press the **Enter** key.



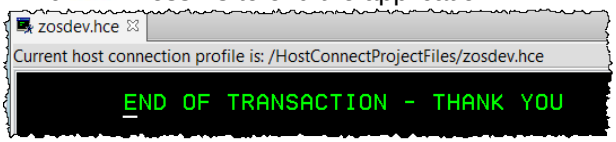
1.2.4 Press **enter** to see the monthly payment for the default data .



1.2.5 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 **YYYMMDD** will be the today's date.



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



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

1.2.8 You may need to restore the **z/OS Projects** perspective 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 **J05P** and verified a simple interaction with the Mortgage application. The objective here was to show the code that you will update.

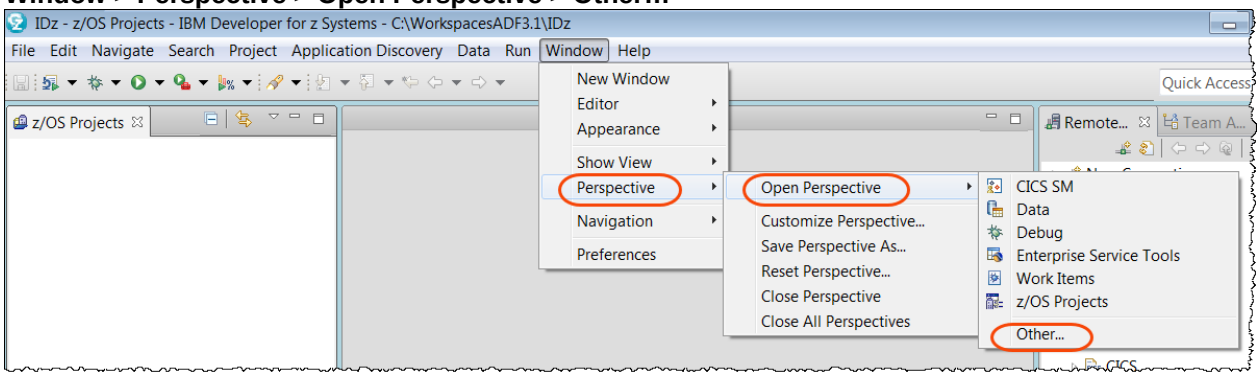
Section 2 – Load the source code from Git to the local IDz workspace

You will load the COBOL code that is stored on Linux to your Windows client to be modified.

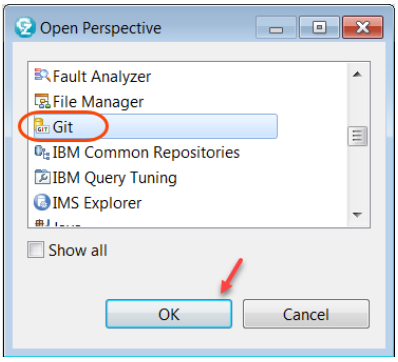
___ 2.1 Cloning the Git Repository

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.

2.1.1 Open the **Git** perspective by selecting
Window > Perspective > Open Perspective > Other…



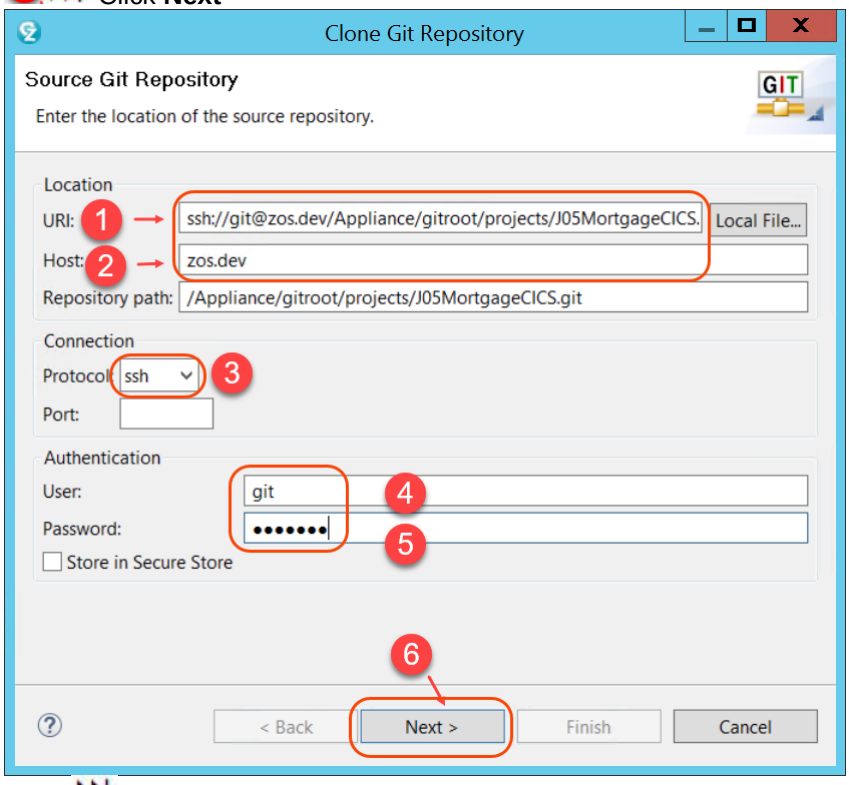
2.1.2 Select **Git** and click **OK**



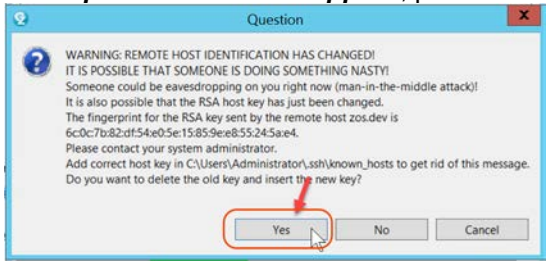
2.1.3 In the *Git Repositories* tab, click on the hyperlink to '**Clone a Git repository'**.

2.1.4 In the dialog that opens, enter the following values
(Tip → Use the notepad and edit **C:\ADF_POT\LAB7\LAB7_Copy_Paste.txt** to copy/paste).

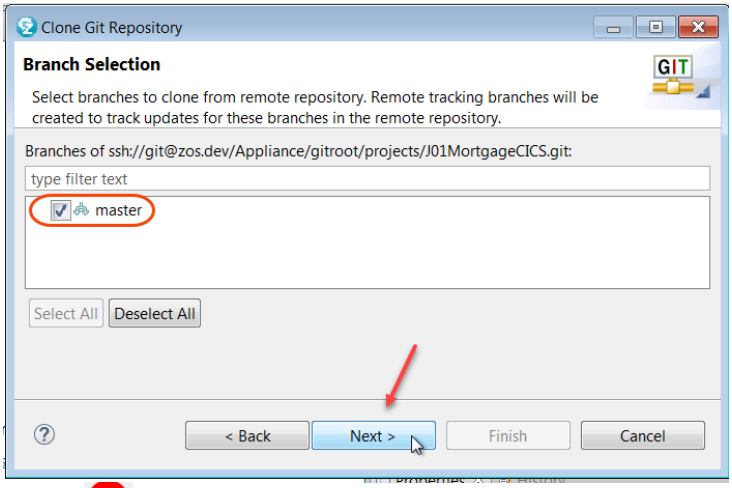
1. Use **Ctrl + Space** to select the URI below (or type).
   URI: **`ssh://git@zos.dev/Appliance/gitroot/projects/J05MortgageCICS.git`**
2. Host: **`zos.dev`**
3. Protocol: **`ssh`** (from the dropdown)
4. User: **`git`** (lower case)
5. Password: **`G1tnimd`** (Note the **1** is number one. Respect upper/lower case)
6. Click **Next**



2.1.5 Select **Yes** for the following pop-up Question.
If the question does not appear, proceed to next step



2.1.6 ⏭ Select **master** and click on **Next**

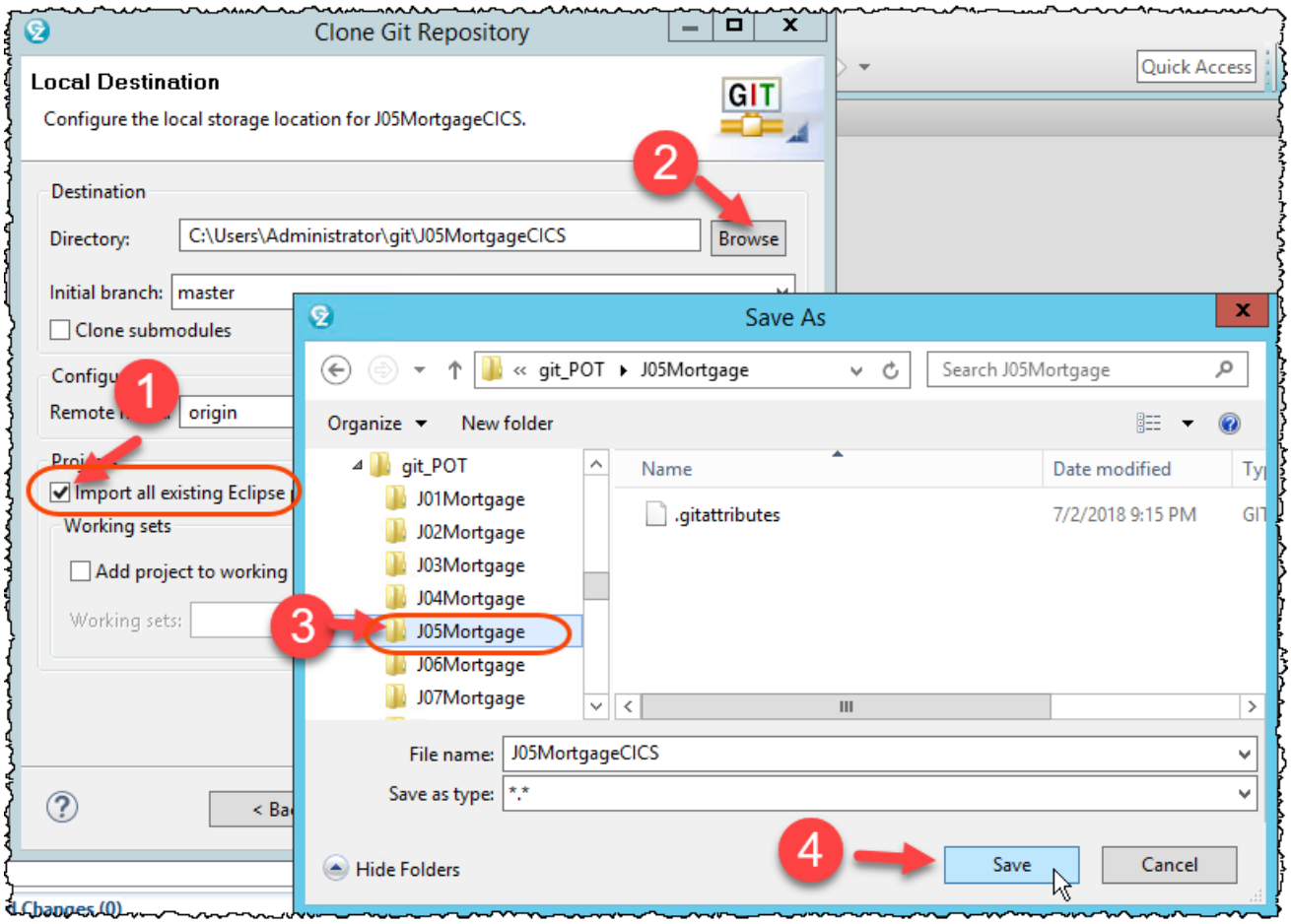


2.1.7 1 ⏭ Click the checkbox to **Import all existing Eclipse project after clone finishes**

2 ⏭ Use **Browse** to select the directory **C:\git_POT\J05Mortgage**.

3 ⏭ Click folder **J05Mortgage** **on left**

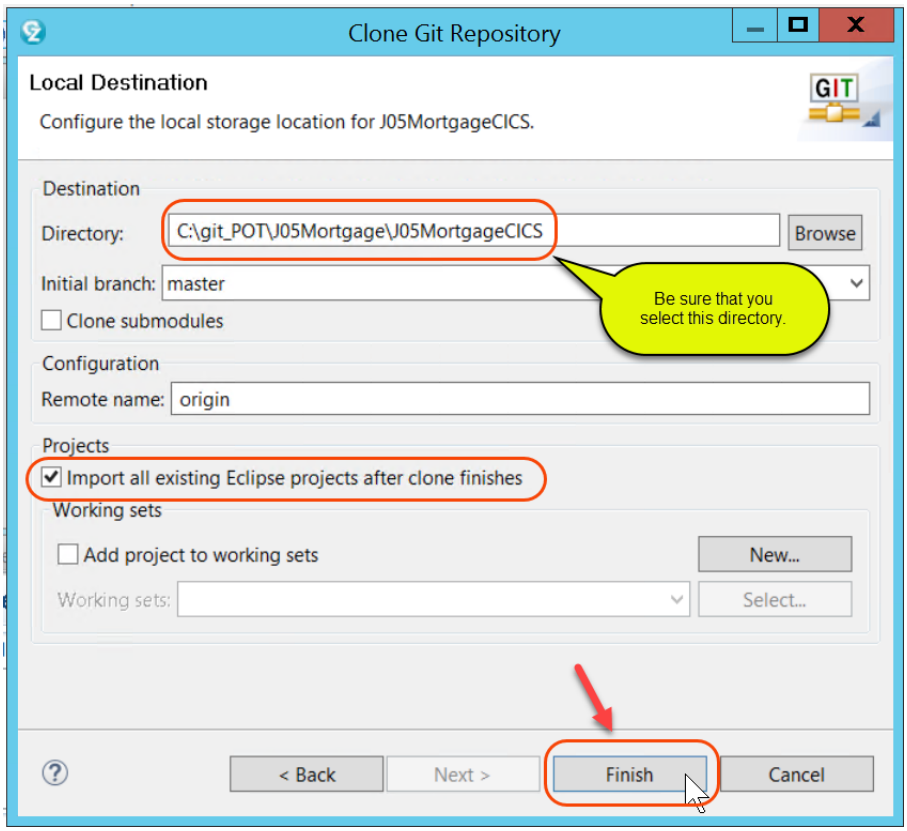
4 ⏭ Click **Save**



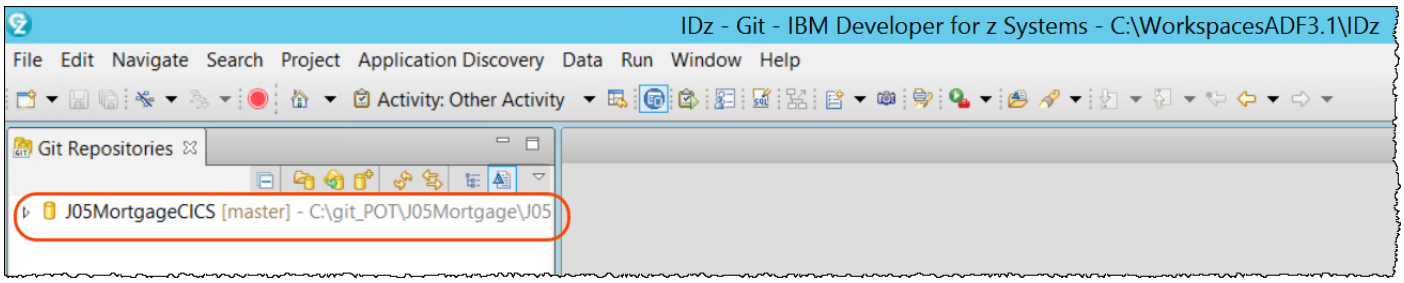
2.1.8 The *Directory* now should point to **`C:\git_POT\J05Mortgage\J05MortgageCICS`**

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

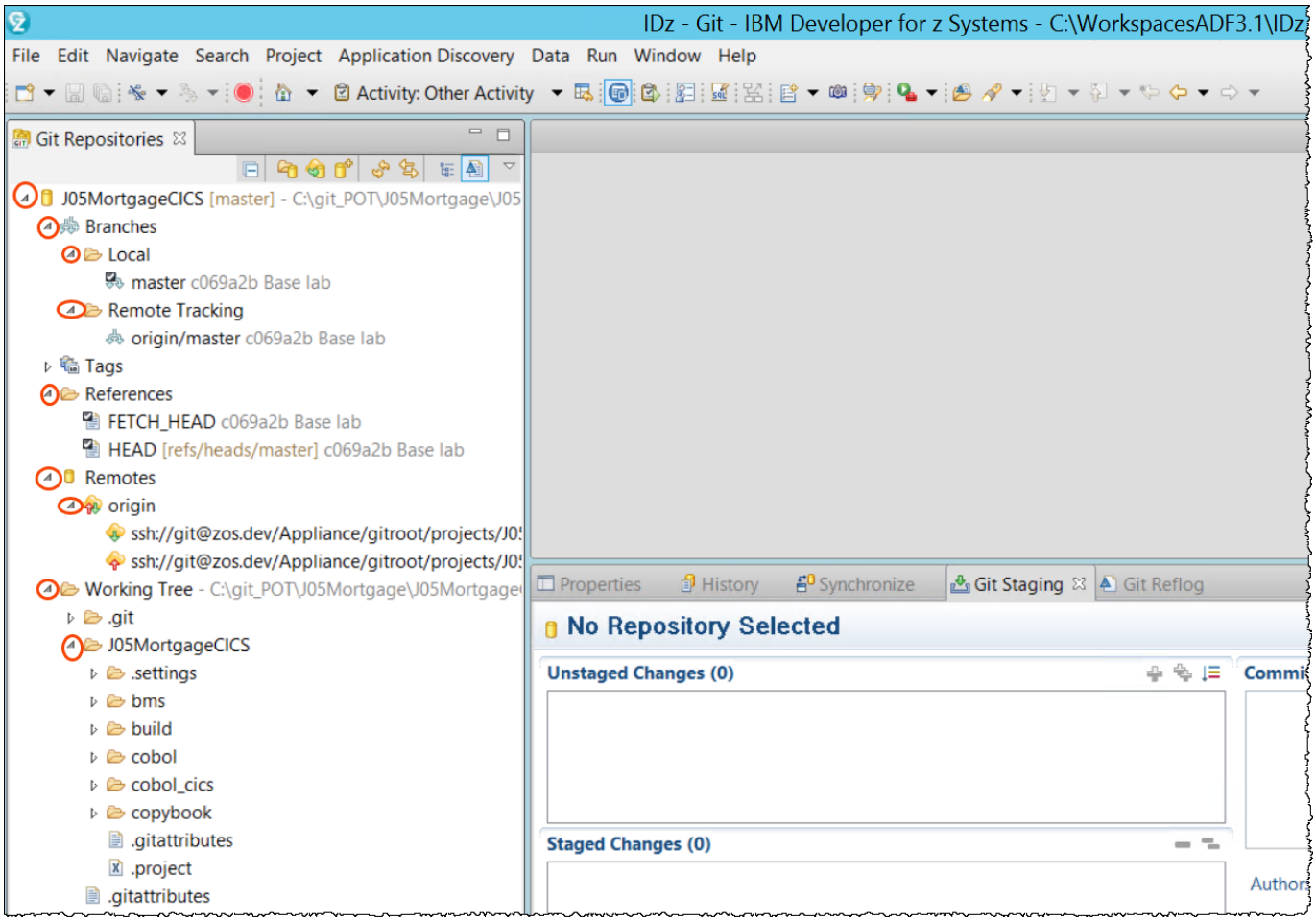
⏭ Click **Finish**



2.1.9 The Mortgage Application will be cloned from the Remote master repository and will appear in the *Git Repositories* view.



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

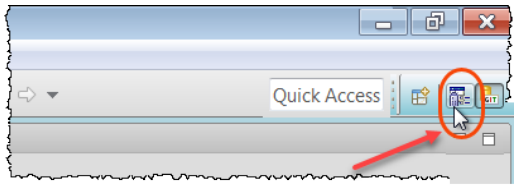


___ 2.2 Verify the code cloned using z/OS projects perspective

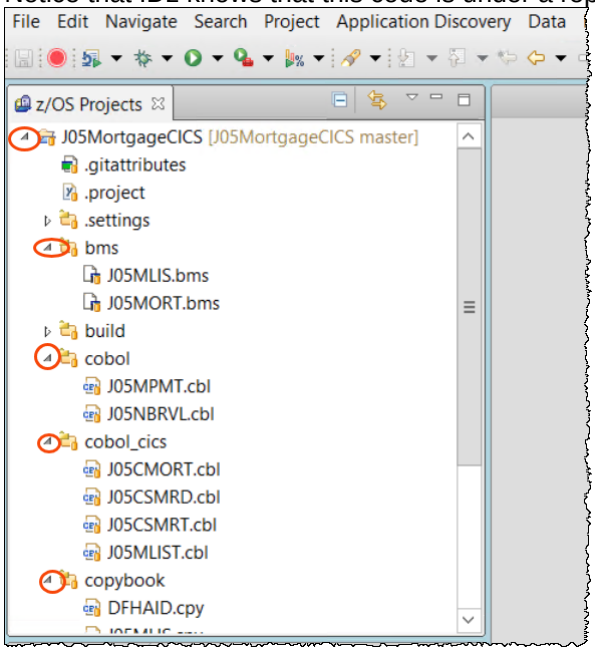
The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

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.



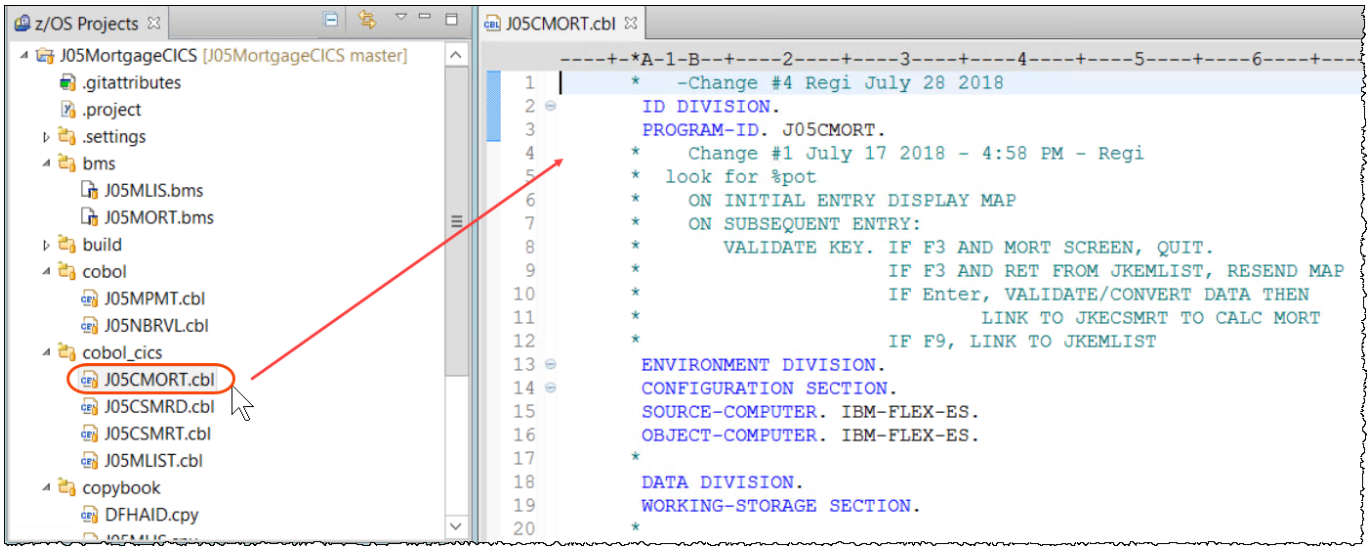
Section 3 –Modify the COBOL code using IDz.

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*".

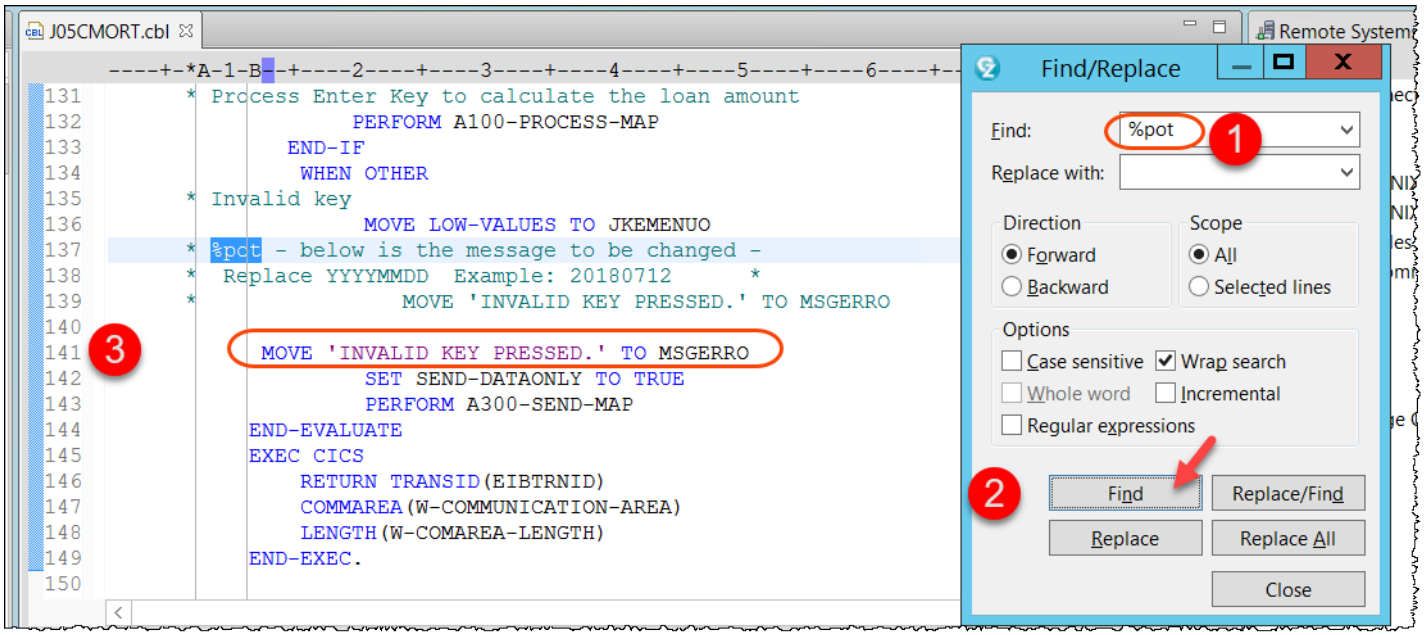
___ 3.1 Edit and modify the code that send the message

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 hit)**. You will modify the line **141**..



3.1.3 **Close** the find dialog and modify the line 141

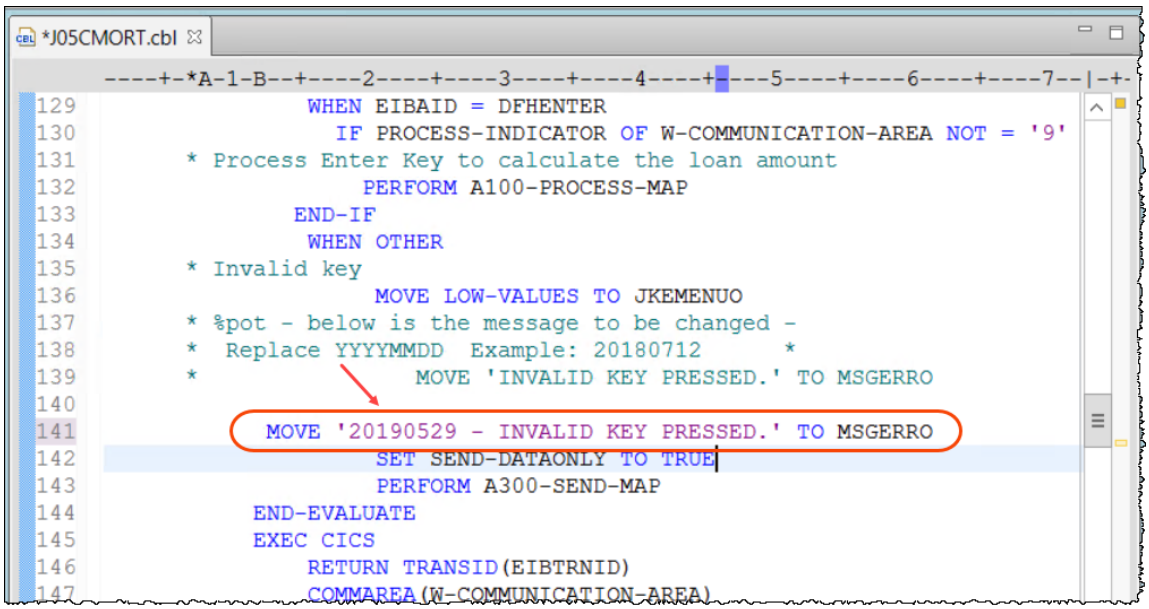
From

```
MOVE 'INVALID KEY PRESSED.' TO MSGERRO
```

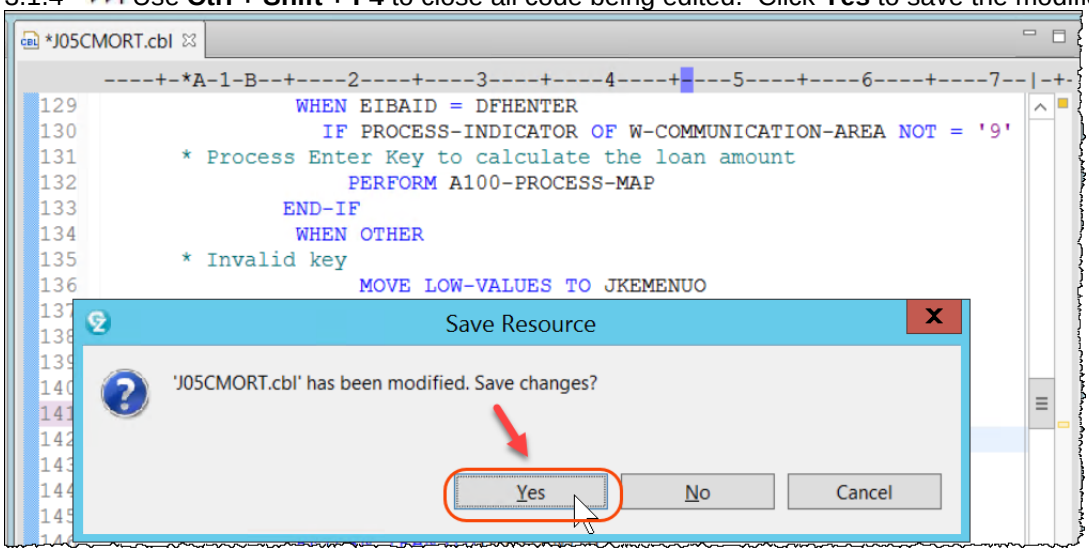
To

```
MOVE 'YYYYMMDD - INVALID KEY PRESSED.' TO MSGERRO
```

Where **YYYYYMMDD** is the today's date.. Example, I changed to **20190529..**



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

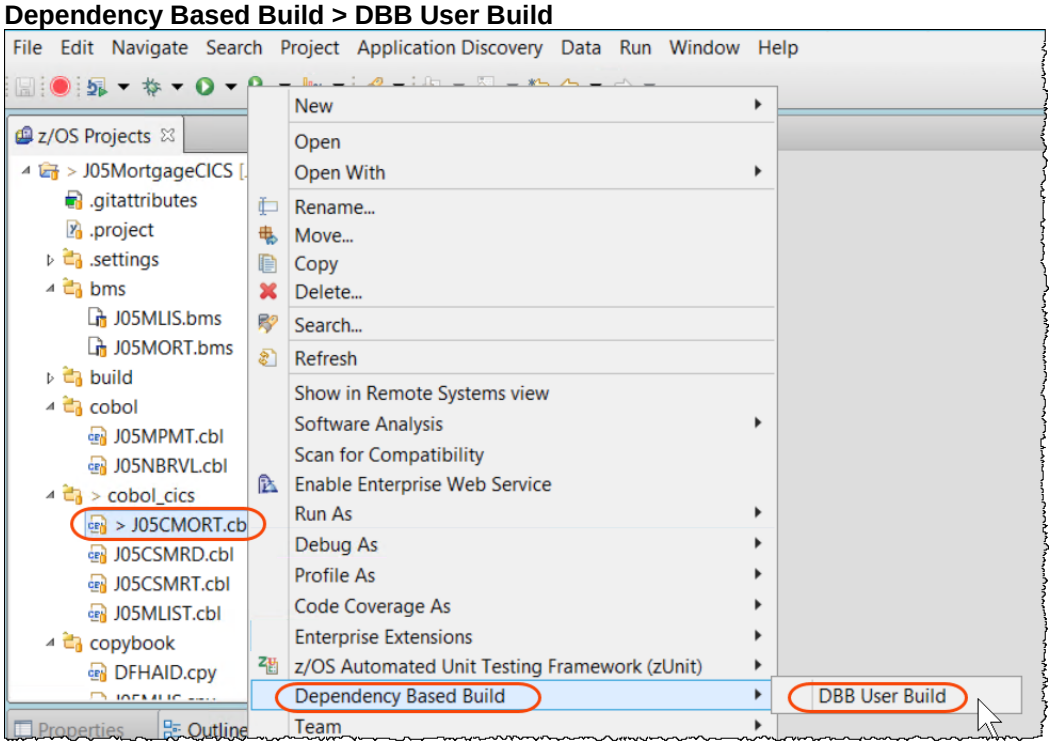


Section 4. Use IDz *DBB User Build* to compile/bind and perform personal tests.

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.

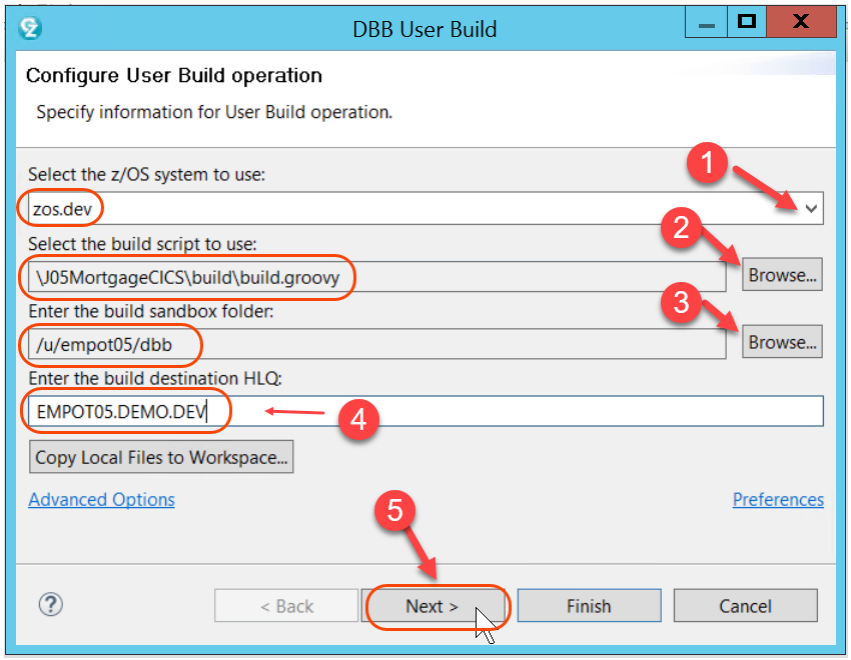
___ 4.1 Using Dependency Based Building option

4.1.1 Under *z/OS Projects* view right click **J05CMORT** and select **Dependency Based Build > DBB 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** to copy/paste).

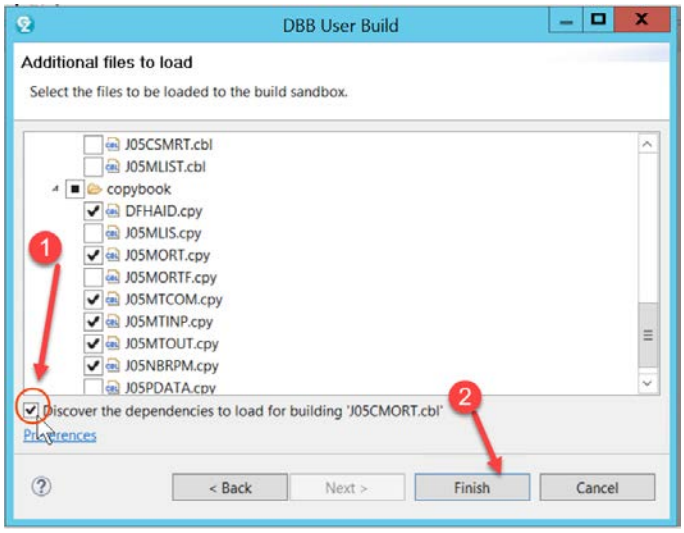
1. *z/OS system:* **`zos.dev`**
2. *Build script* **:** **`\J05MortgageCICS\build\build.groovy`** (use **Browse** the **local workspace**)
3. *Build sandbox* **:** **`/u/empot05/dbb`** (Use **Browse** and click **MyHome** and **dbb**)
4. *Build destination HLQ*: **`EMPOT05.DEMO.DEV`** (must type)
5. ⏭ Click **Next**



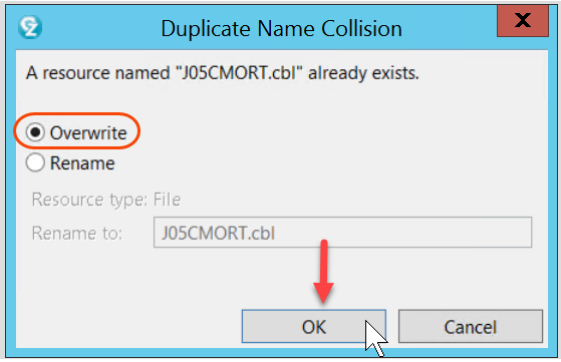
4.1.3 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

⏭ Since you will also need the related copybooks for a clean compile **click "Discover the dependencies to load for building J05CMORT.cbl "**.

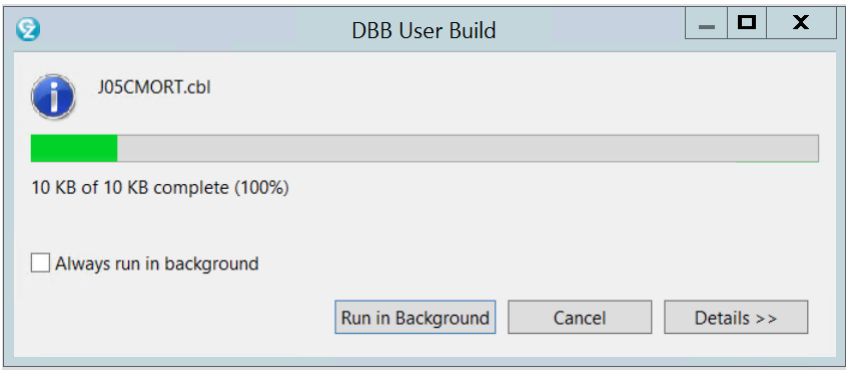
⏭ Click **Finish**



4.1.4 If the dialog below pops up select **Overwrite** and click **OK**



4.1.5 The DBB User build will be running..



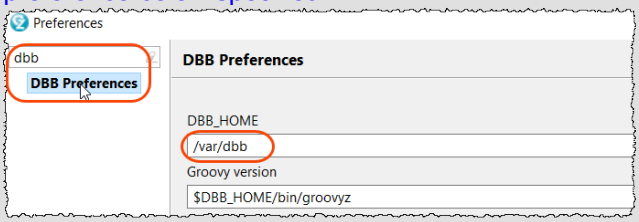
The DBB User build is failing with the message below?.

Error:

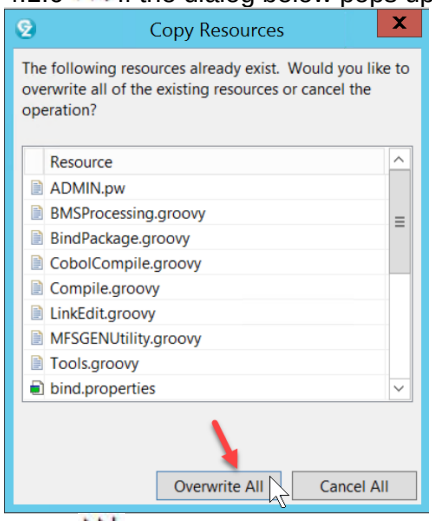
```
sh

cd /u/empot01/dbb
DBB_HOME=$DBB_HOME
sh -c "$DBB_HOME/bin/groovyz /u/empot01/dbb/J05MortgageCICS/build/build.groovy --
sourceDir /u/empot01/dbb  --workDir /u/empot01/dbb/work --hlq EMPOT05.DEMO.DEV --
userBuild J05MortgageCICS/cobol_cics/J05CMORT.cbl"
/u/empot05/dbb>
/bin/groovyz: FSUM7351 not found
/u/empot05/dbb>
```

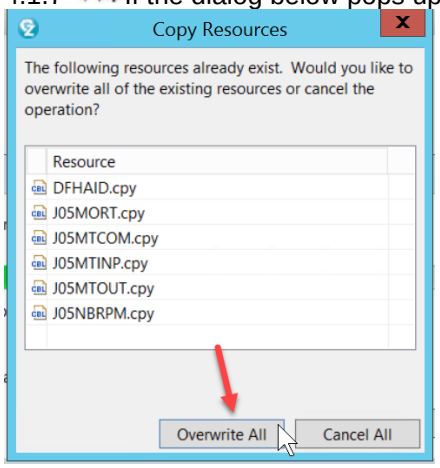
Be sure you are using the correct IDz workspace and also verify that your workspace has the preference below specified.



4.1.6 ⏭ If the dialog below pops up click **Overwrite All**

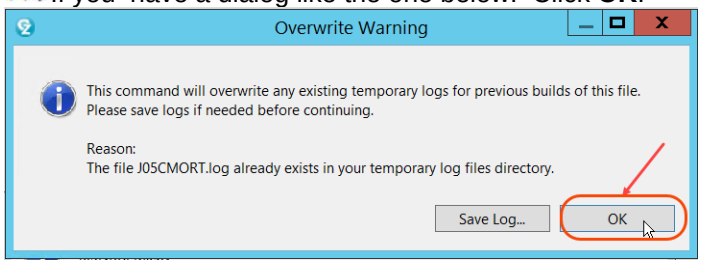


4.1.7 ⏭ If the dialog below pops up click **Overwrite All**

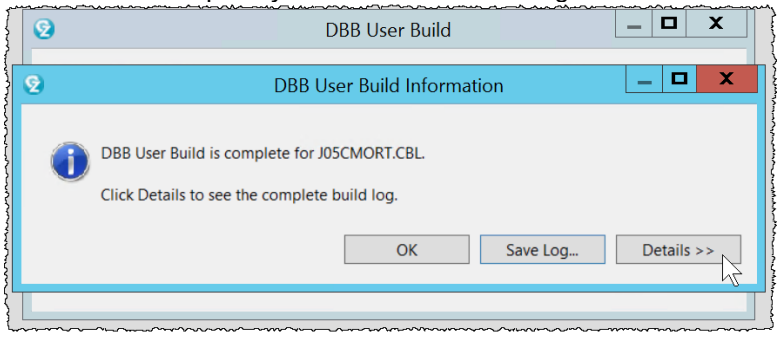


4.1.8 The execution will start and this will take a while (2 to 3 Minutes) . Time for a quick break.

⏭ If you have a dialog like the one below. Click **OK**.

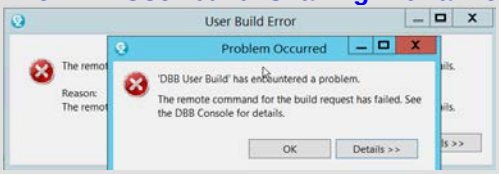


4.1.9 When complete you will have the message below:



If the build fails with an error message see the explanation on next page. It is a known issue.
You may also get a dialog asking if I wanted to save previous logs, just click **OK**.

The DBB User build is failing with a message below?

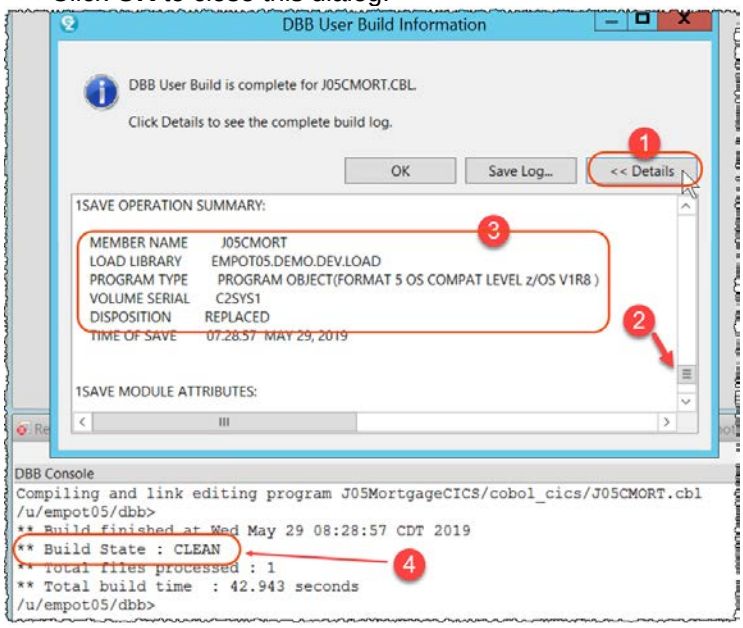


Its related to the low z/OS memory being used on cloud that may cause eventual errors.
Sometimes just trying it again may solve the issue,
Look at the Console view. If it shows **`Build State : CLEAN`** you are good.

___ 4.2 Verify the DBB User Build results

4.2.1 ⏭ Click on **Details >>** and scroll down and verify that a new load module name **J05CMORT** was replaced on the dataset ***EMPOT05.DEMO.DEV.LOAD***
Notice also the console shows a *CLEAN* build.

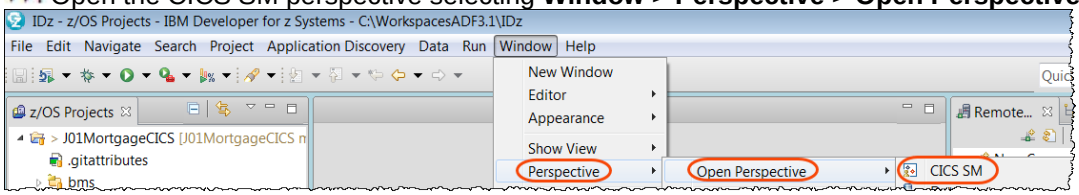
⏭ Click **OK** to close this dialog.



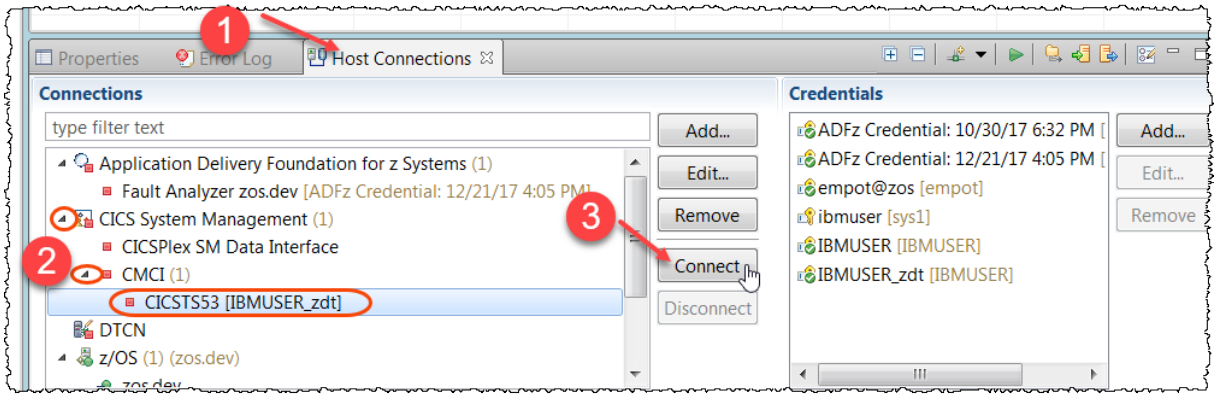
___ 4.3 Issuing a CICS New copy

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 ⏭ Open the tab **Host Connections on the bottom**, expand **CICS System Management, CMCI**, select **CICSTS53** and click on **Connect**

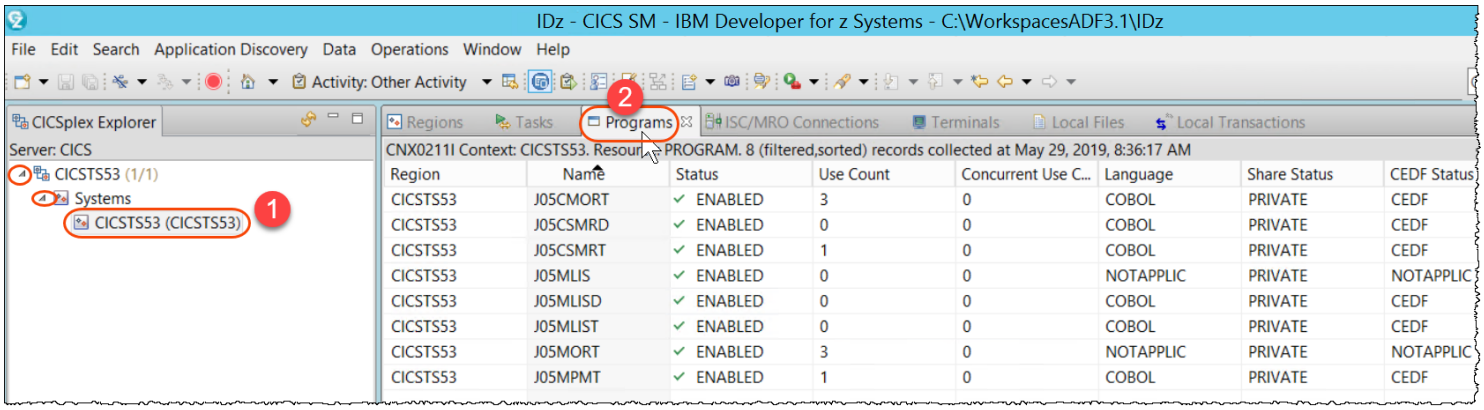


4.3.3 The red dot will turn green once is connected to CICS

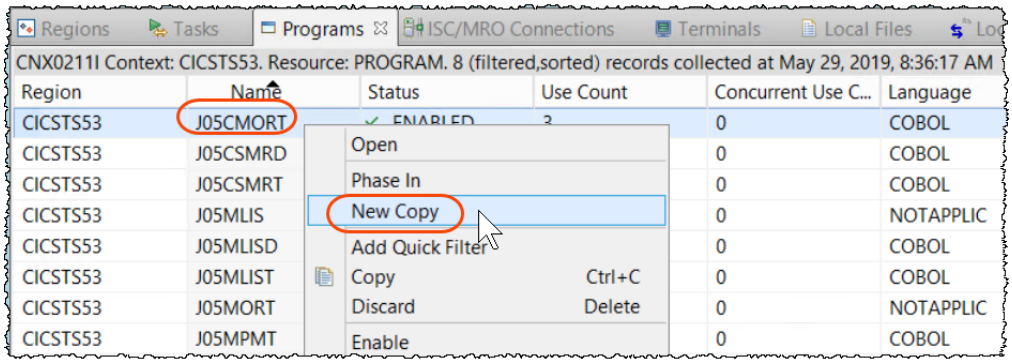
4.3.4 ⏭ On top left expand **CICSTS53** and l cicsts53 click on **CICSTS53 (CICSTS53).**

⏭ Click on the **Programs** tab.

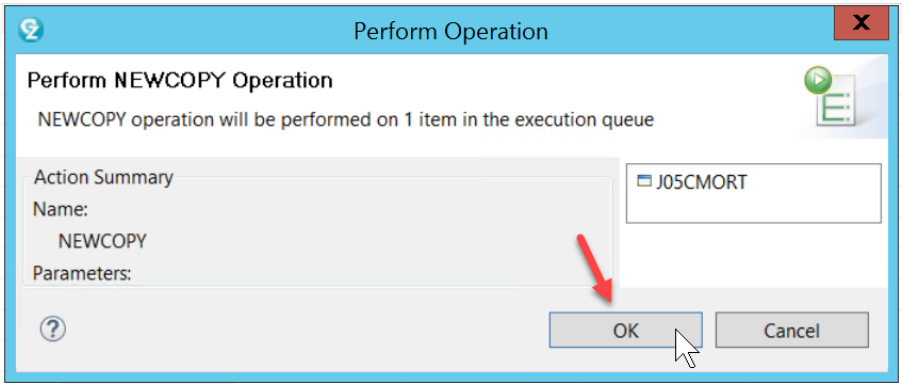
Since the filter is already made for " **Name = J05*** " you should see the list below:



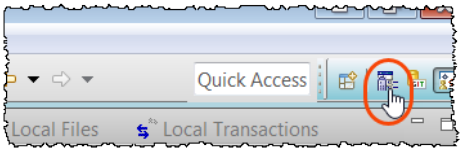
4.3.5 ⏭ Right click on **J05CMORT** and select **New Copy** as the example below



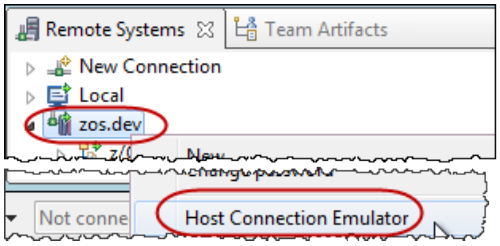
4.3.6 ⏭ Select **OK**



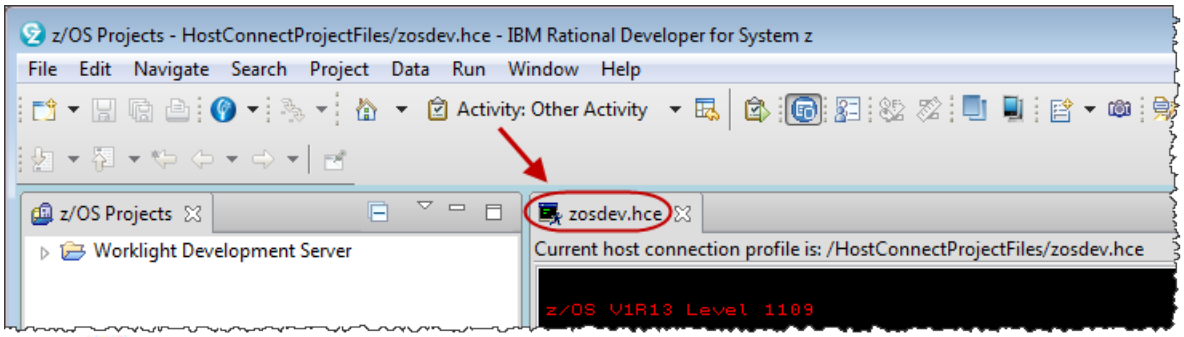
4.3.7Let's start the 3270 emulation again ..
⏭ Go to the **z/OS Projects** perspective



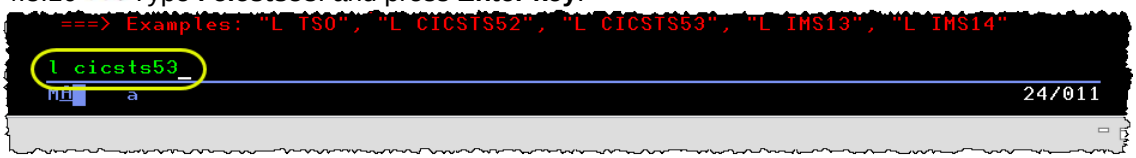
4.3.8 ⏭ Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



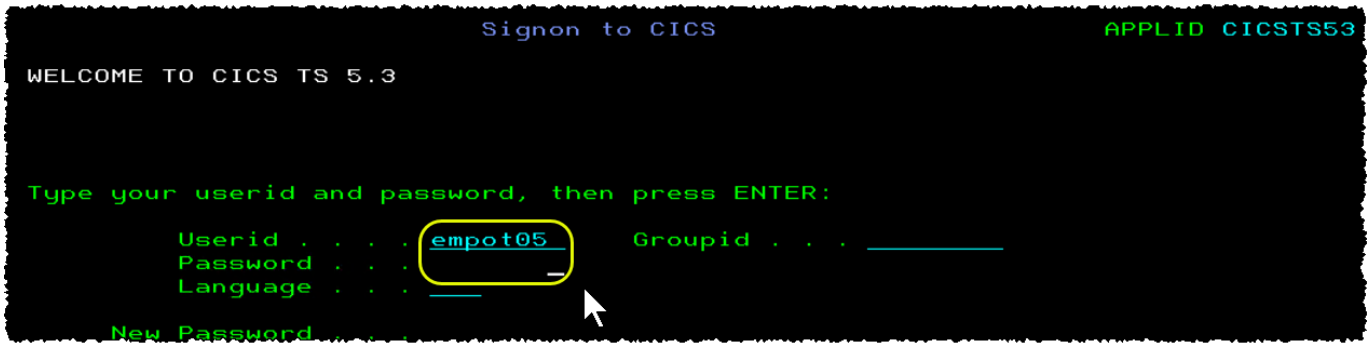
4.3.9 ⏭ Since you will need more space, **double-click** on the **zos.dev.hce** title



4.3.10 ⏭ Type **l cicsts53**. and press **Enter key**.



4.3.11 Logon using **empot05** and password **empot05** and press **Enter**.

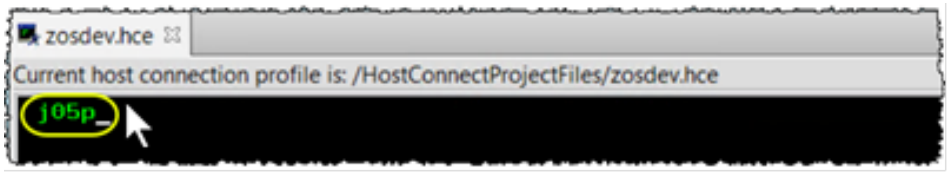


4.3.12 The sign-on message is displayed

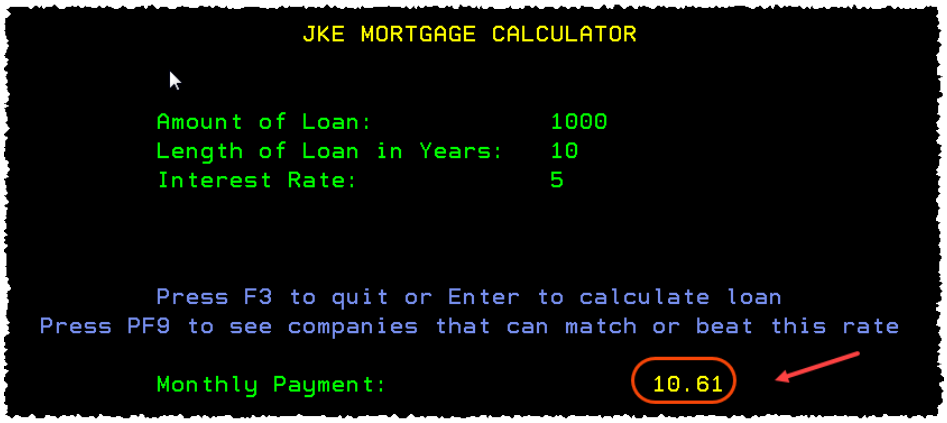
___ 4.4 Running *j05p* CICS transaction

You should now be on the z/OS CICS region named *CICSTS53*. This is the CICS instance where you made the program changes.

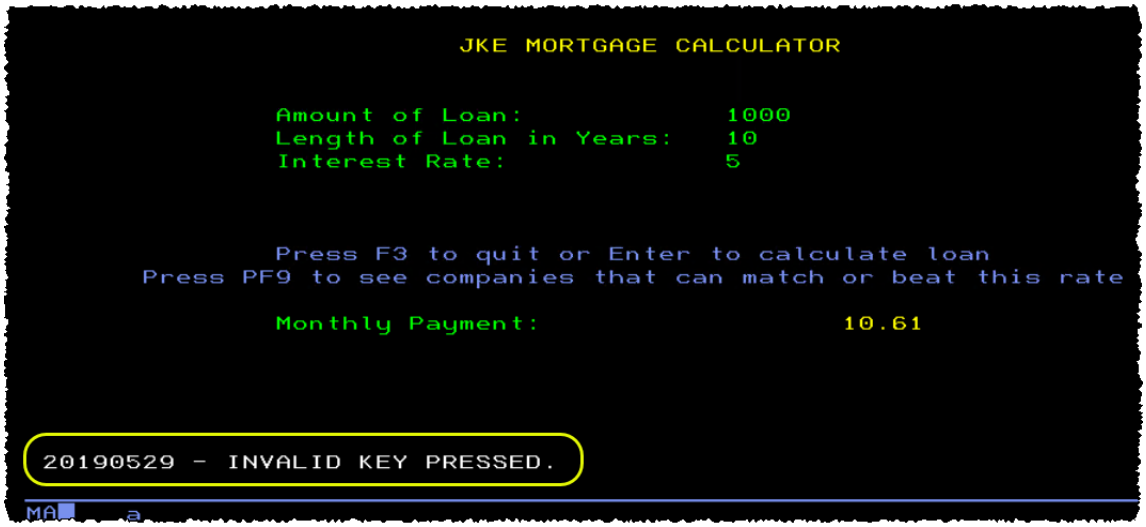
4.4.1 Type the CICS transaction **j05p** and press the **Enter** key.



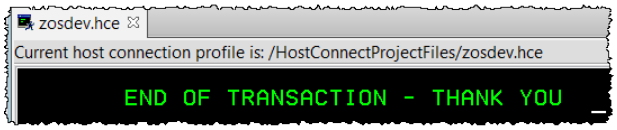
4.4.2 Press **enter** to see the monthly payment for the default data .



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



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



4.4.5 Use **Ctrl + Shift + F4** to close the terminal emulation.

Section 5. Push and Commit the changed code to Git.

Using IDz you will commit the changes you made to Git and later perform the final building and deploy using Jenkins and UCD.

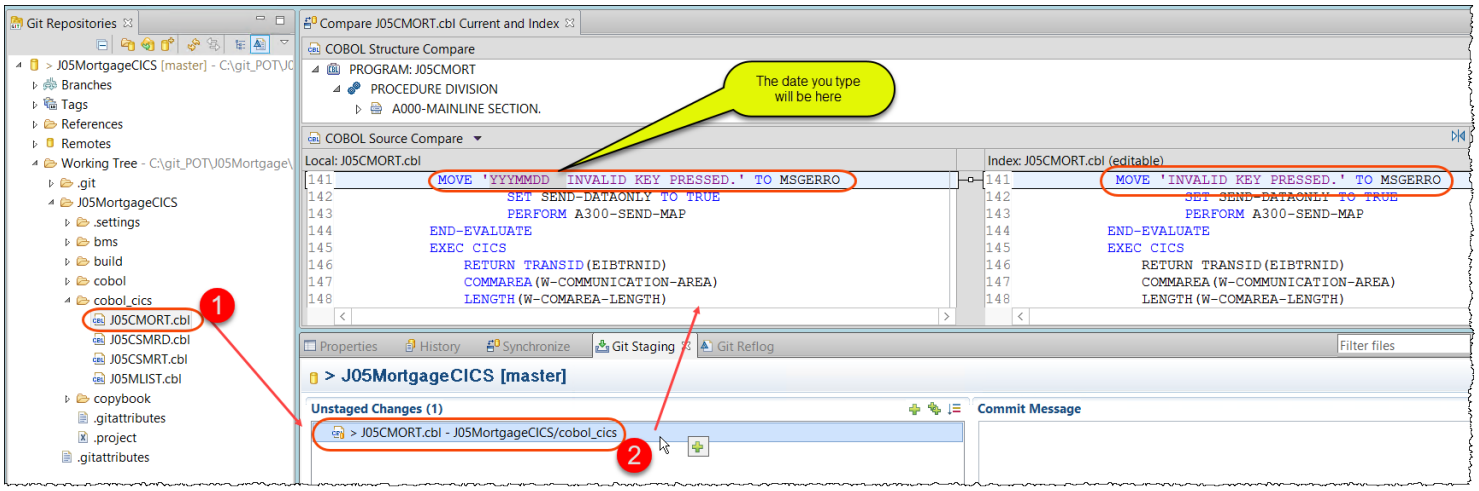
___ 5.1 Using IDz with the Git plugin to compare the change versus original

.

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

5.1.2 ⏭ 1 Expand the nodes and click on the program **J05CMORT.cbl** that you have modified.
Notice that the changed program is listed in the *Git Staging* view..
⏭ 2 Double click on the **J05CMORT.cbl** that is listed under ***Unstaged Changes*** and noticed the comparison among the program that you updated versus the one that is in the Git repository.

You will need to commit the changes to the Git repository to make a final build later.



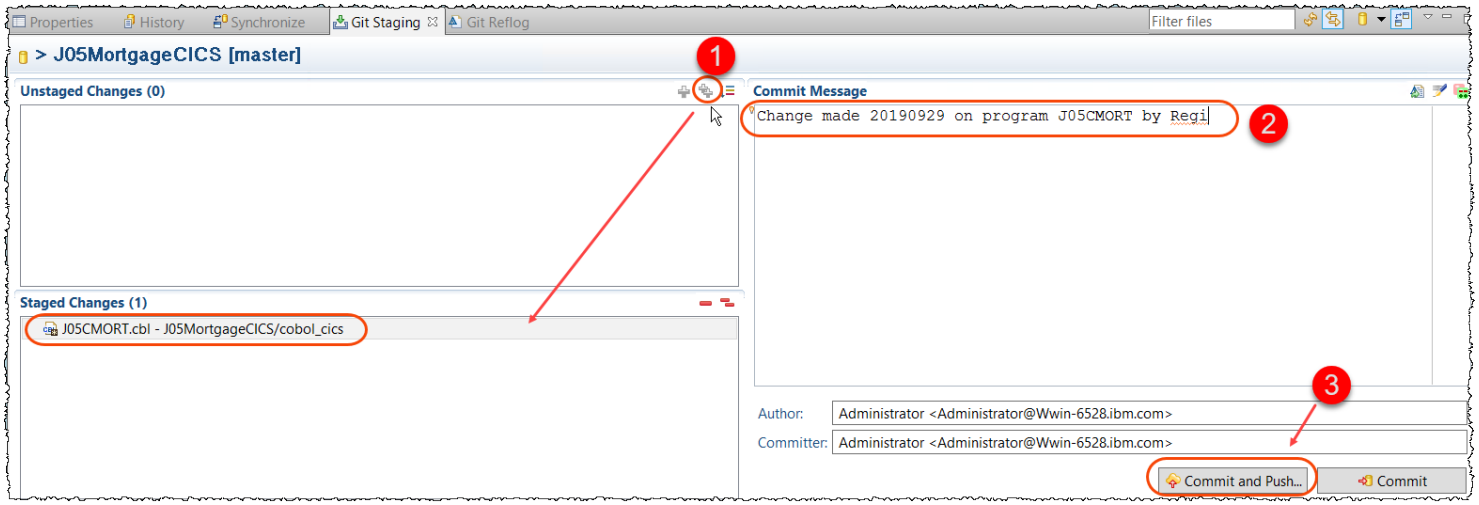
5.1.3 ⏭ Use **Ctrl + Shift + F4** to close the editors.

___ 5.2 Push and Commit the changes to Git

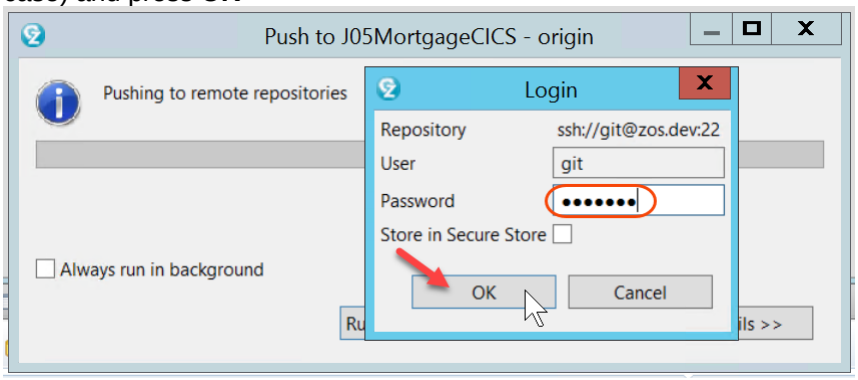
.

5.2.1 ⏭ Using **Git Staging** view, ,
1 Click on the icon ,
2 Write a commit message like: **Change made yyyymmdd on program J05CMORT by *your name***

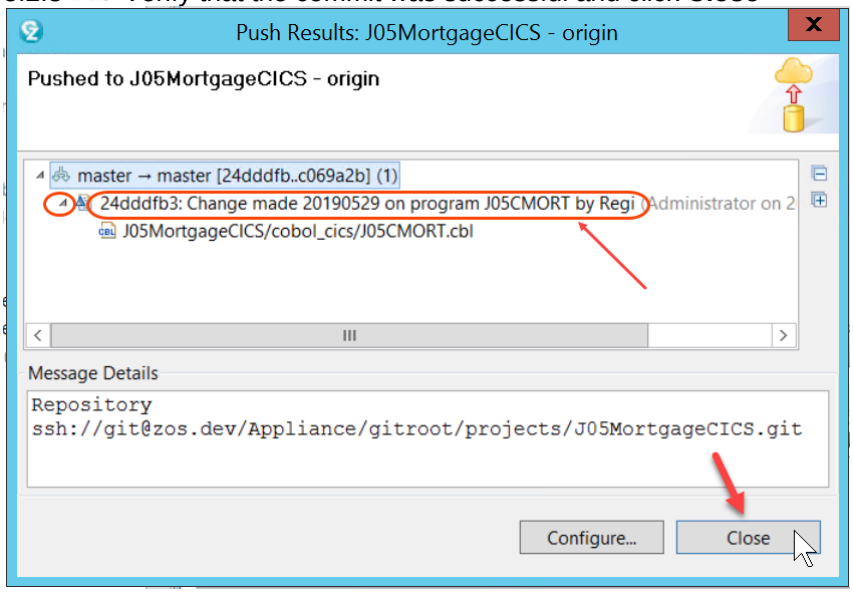
3 Click **Commit and Push …**



5.2.2 If the password prompt pops up, use **`G1tnimd`** (where “1” is number one, respect the upper/lower case) and press **OK**



5.2.3 Verify that the commit was successful and click **Close**

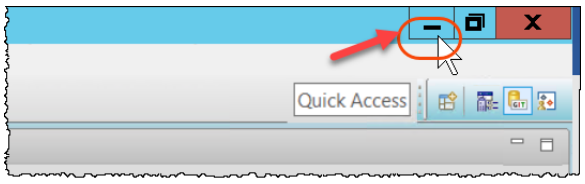


Section 6. Use Jenkins with Git plugin to build all the modified code committed to Git.

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.

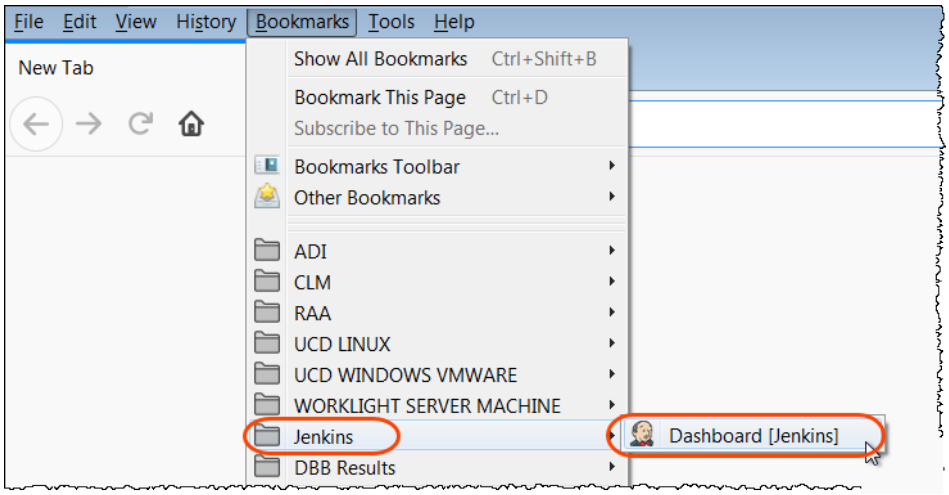
___ 6.1 Logon to Jenkins using a Web Browser

6.1.1 **Minimize IDz** clicking on top right:

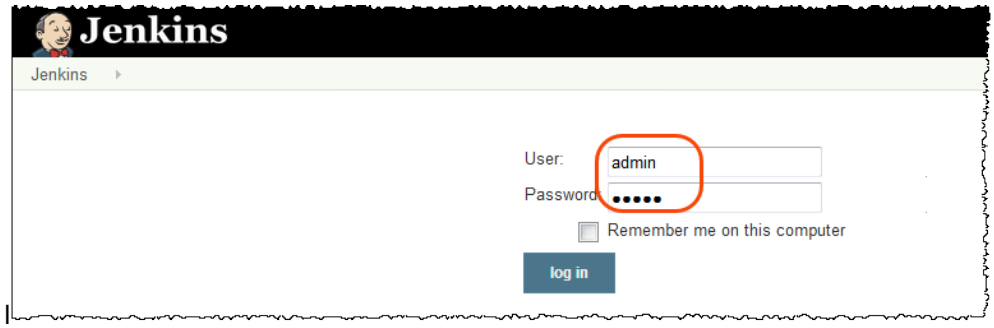


6.1.2 Start the **Firefox** browser

6.1.3 Using the **Bookmarks** click on **Jenkins > Dashboard**



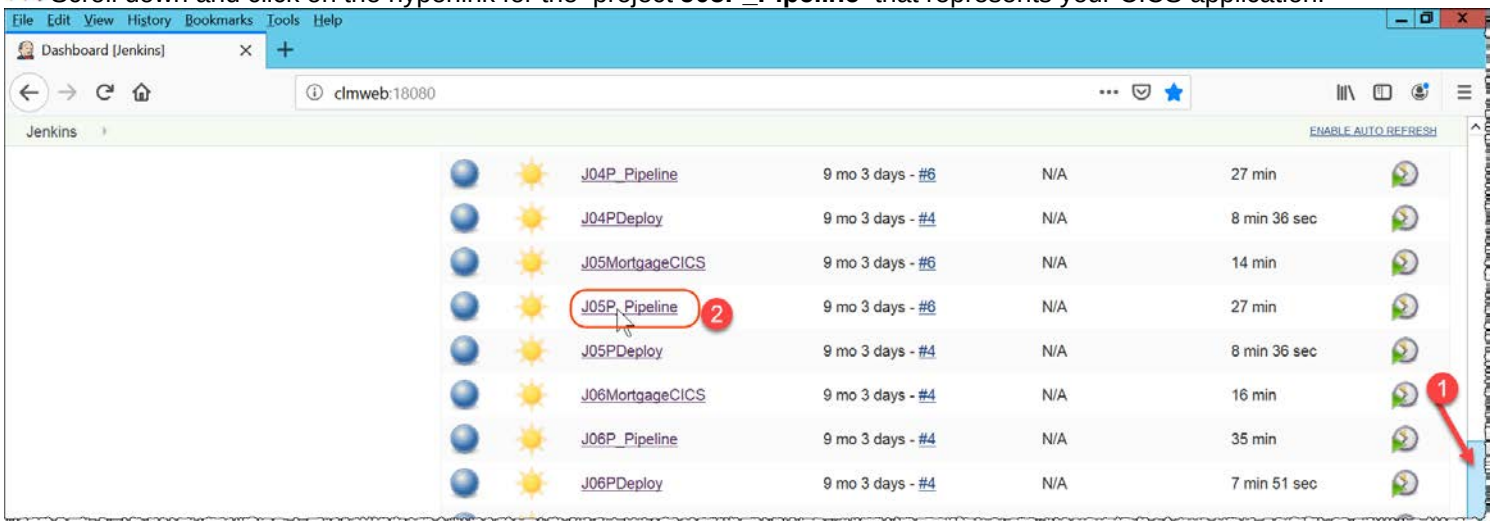
6.1.4 If required type the user **admin** and password **admin** (lowercase) and click **log in**.



___ 6.2 Starting the Jenkins Pipeline

6.2.1 The homepage lists all of the Jenkins jobs and pipelines.

⏭ Scroll down and click on the hyperlink for the project **J05P_Pipeline** that represents your CICS application.

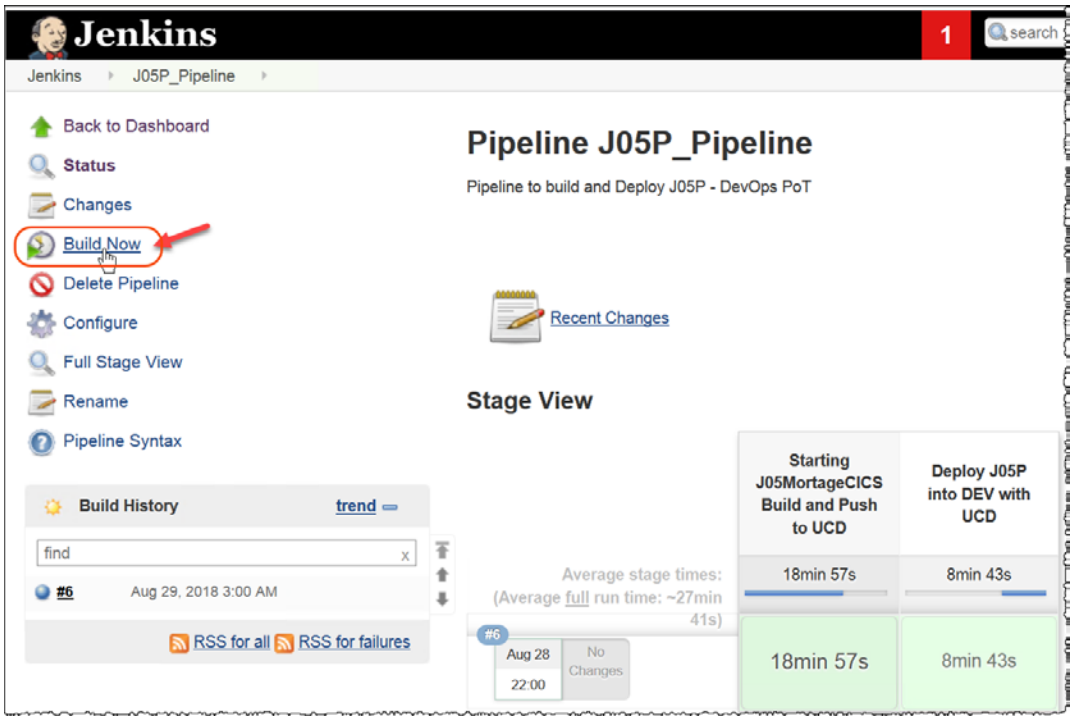


6.2.2 This opens the **J05P_Pipeline** view. This pipeline has been designed with two stages:

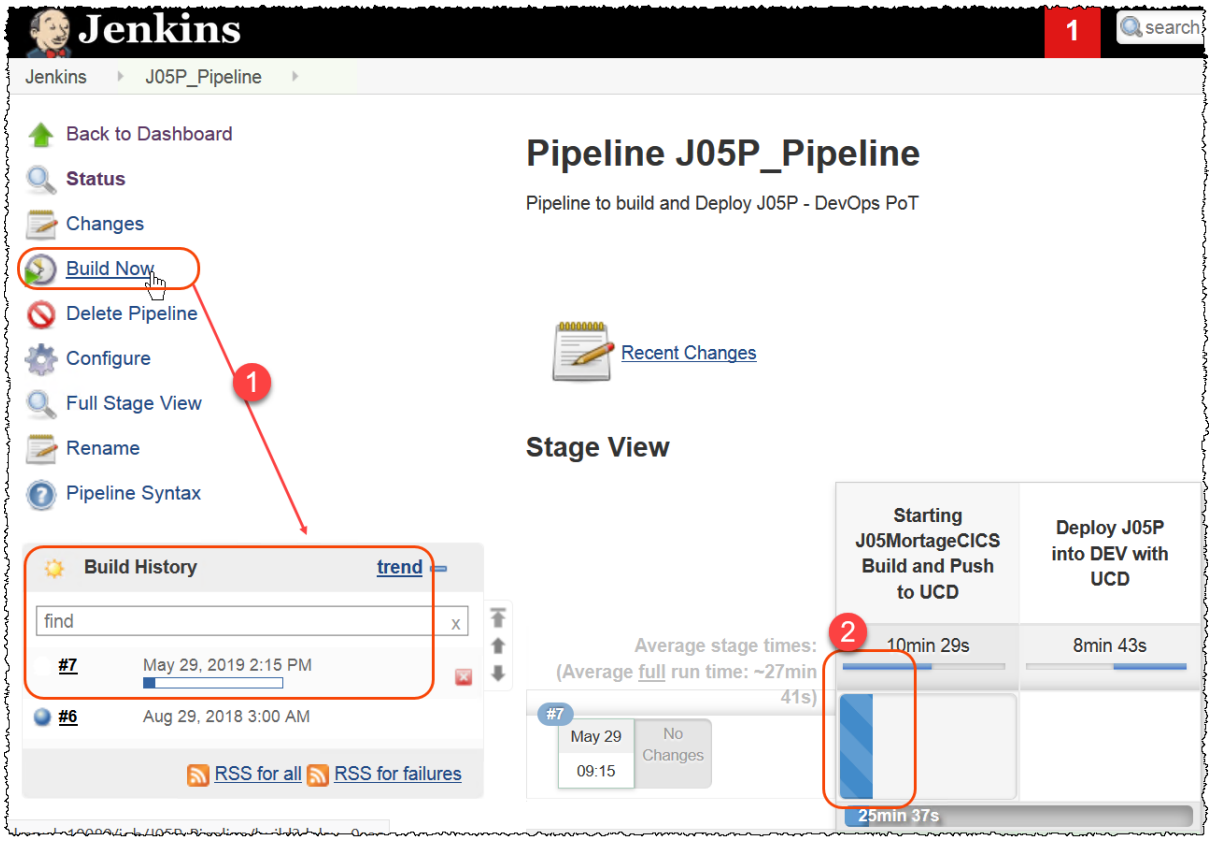
1 – Stage 1 builds ONLY the changed code from Git. This is an example of a Build pipeline.
2 – Stage 2 deploys the changes into CICS Development Region (Dev) using UrbanCode Deploy (UCD).
This is an example of a Delivery pipeline.

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.



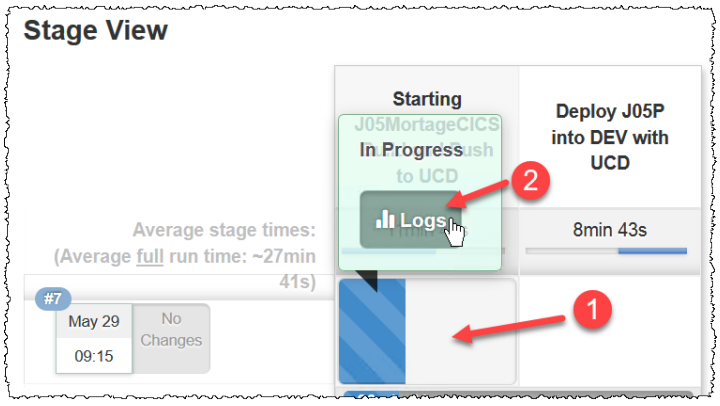
6.2.3 (1) Notice that the new build has appeared under **Build History** indicating the currently started Pipeline.
(2) The **Stage View** also shows a new line with an in-progress bar.



___ 6.3 Checking the results

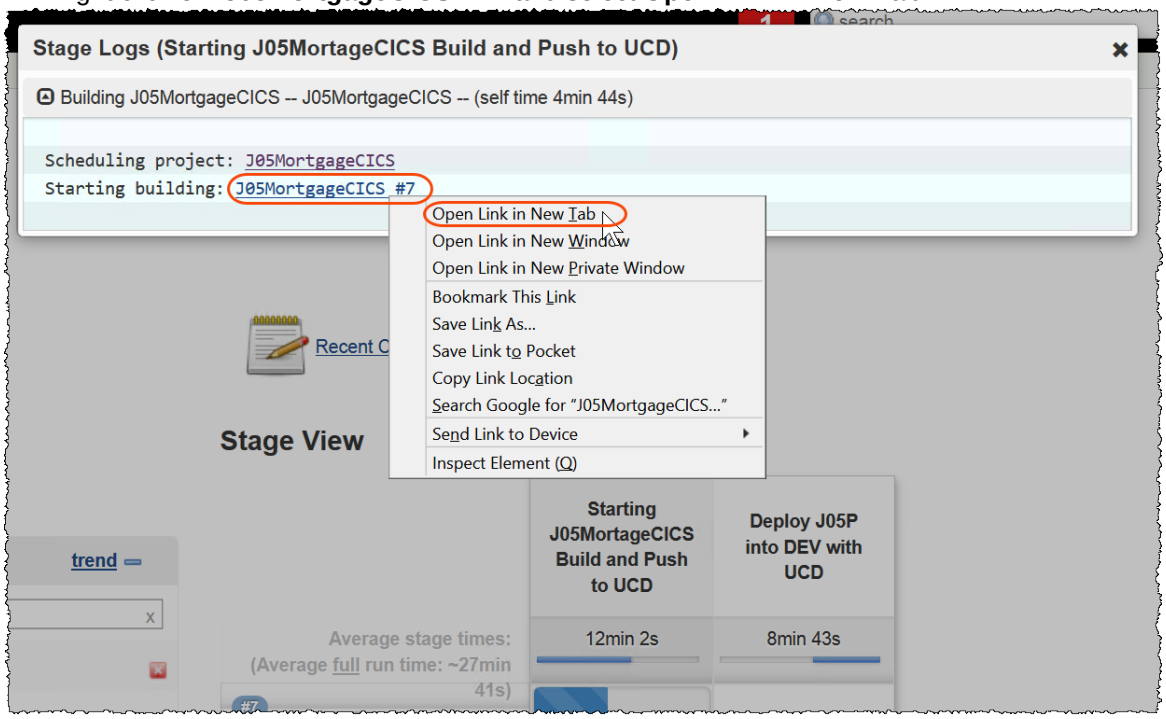
6.3.1 Since this z/OS is running in a small processor in the cloud, this activity may take 10 or more minutes, but you don't need to wait to be completed to check the results.
You can follow what is going on as described below ..

⏭ Click as show in the **blue area** (1) and then click on **Logs** (2).



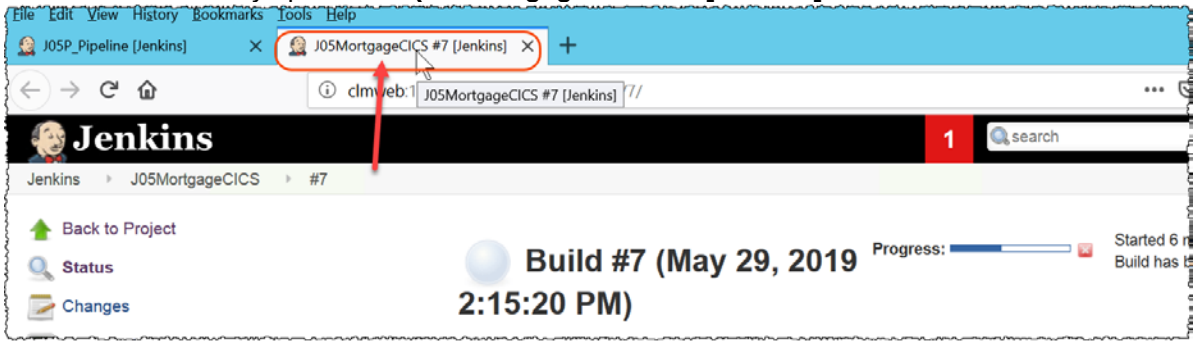
6.3.2 The dialog Stage Logs will open.

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



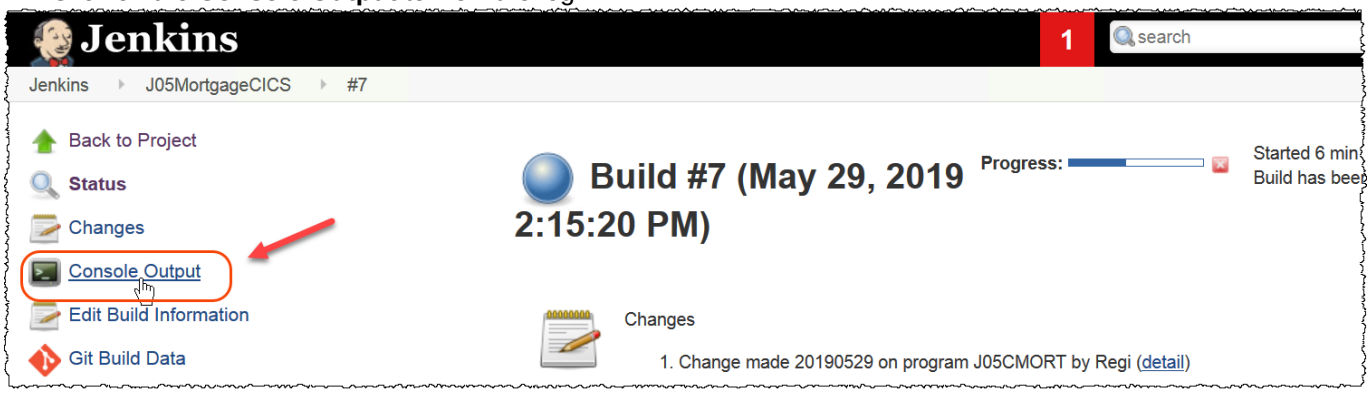
6.3.3 This opens a new Jenkins tab as shown below.

⏭ Click on the newly opened tab **(J05MortgageCICS#nn [Jenkins]**



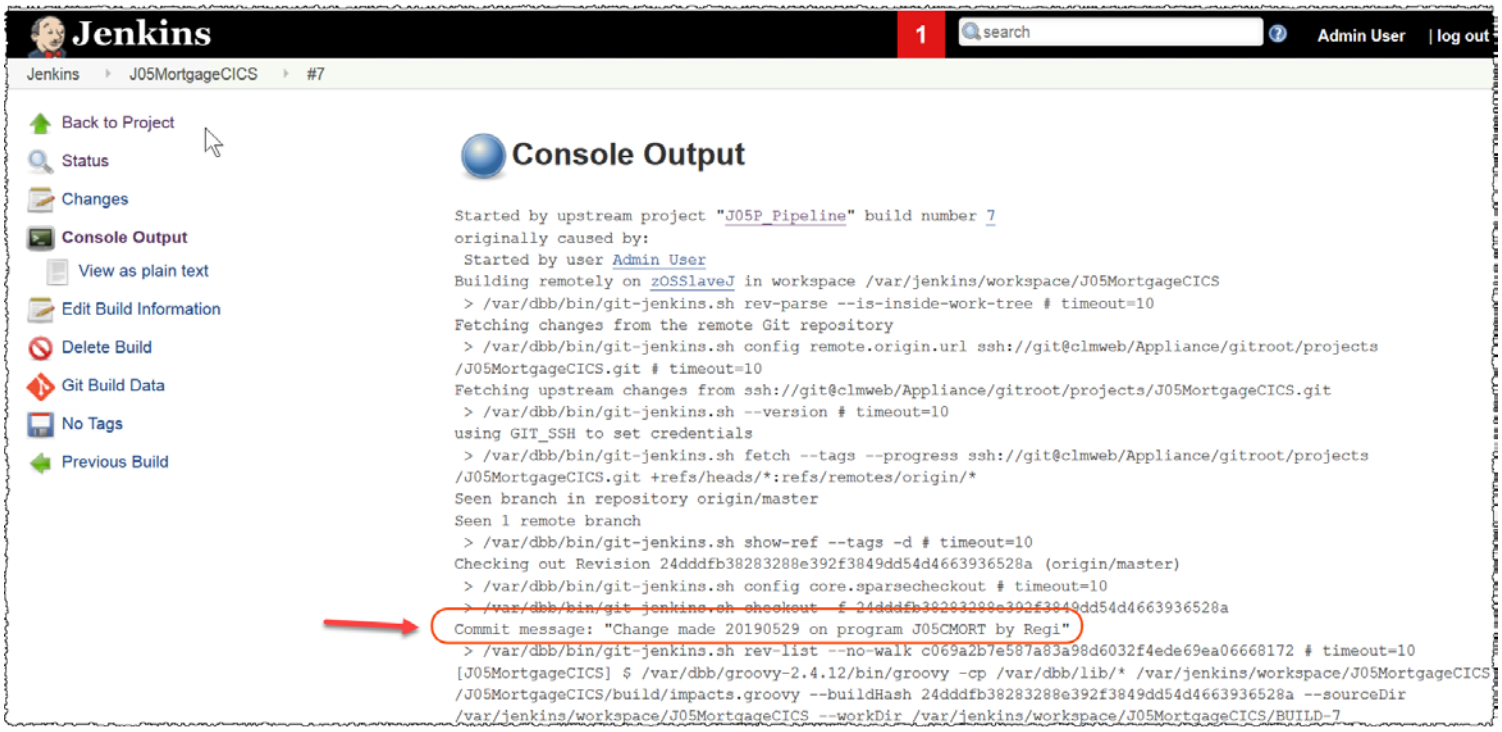
6.3.4 This opens the current running task for the first step of the J05Mortgage Application Build.

⏭ Click on the **Console Output** to view the log.



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.

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.



____ 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

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 20190529.022017.020
** Searching for last successful build commit hash for build group J05MortgageCICS
Last successful build commit hash located. label : build.20180829.031408.014 , buildHash :
c069a2b7e587a83a98d6032f4ede69ea06668172
** Executing Git command: git diff --name-only c069a2b7e587a83a98d6032f4ede69ea06668172
24dddfb38283288e392f3849dd54d4663936528a
Number of changed files detected since build build.20180829.031408.014 : 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-7/buildlist.txt
** Impact analysis finished at Wed May 29 14:20:55 GMT 2019
** Total # build files calculated = 1
** Total analysis time : 37.993 seconds
[J05MortgageCICS] $ /var/dbb/groovy-2.4.12/bin/groovy -cp /var/dbb/lib/* /var/jenkins/workspace/J05MortgageCICS
/J05MortgageCICS/build/build.groovy --buildHash 24dddfb38283288e392f3849dd54d4663936528a --sourceDir
```

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 build list.
** Writing buildlist to /var/jenkins/workspace/J05MortgageCICS/BUILD-7/buildlist.txt
** Impact analysis finished at Wed May 29 14:20:55 GMT 2019
** Total # build files calculated = 1
** Total analysis time : 37.993 seconds
```

This collection is on the DBB App Server

6.4.4 **A compile/link-edit** of the changed programs

```
HTTP/1.1 200 OK
** 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 May 29 14:22:58 GMT 2019
** Build State : CLEAN
** Total files processed : 1
** Total build time  : 1 minutes, 13.561 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.

```
[J05MortgageCICS] $ /var/dbb/groovy-2.4.12/bin/groovy -cp /var/dbb/lib/* /var/jenkins/workspace/J05MortgageC
/J05MortgageCICS/build/deploy.groovy --buztool /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh --workDir
/var/jenkins/workspace/J05MortgageCICS/BUILD-7 --component J05MortgagePOT
** Create version start at 20190529.022335.023
** Properties at startup:
   component -> J05MortgagePOT
   startTime -> 20190529.022335.023
   workDir -> /var/jenkins/workspace/J05MortgageCICS/BUILD-7
   buztoolPath -> /etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh
** Read build report data from /var/jenkins/workspace/J05MortgageCICS/BUILD-7/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-7/shiplist.xml
** Create version by running UCD buztool
/etc/ibm-ucd/v6.2.6/dtsc-agent/bin/buztool.sh createzosversion -c J05MortgagePOT -s /var/jenkins/workspace
/J05MortgageCICS/BUILD-7/shiplist.xml -o /var/jenkins/workspace/J05MortgageCICS/BUILD-7/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 : 20190529-142402
....Shiplist file : /var/jenkins/workspace/J05MortgageCICS/BUILD-7/shiplist.xml
....Output File:/var/jenkins/workspace/J05MortgageCICS/BUILD-7/buztool.output
Verifying version
....Repository location : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20190529-142402
Pre-processing shiplist:
....Shiplist after processing :/etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20190529-142402
/shiplist.xml
```

```
   Packaging data sets:
   ....Location to store zip : /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J05MortgagePOT/20190529-142402
   ....Zip name : package.zip
   ....EMPOT05.DEMO.DEV.LOAD.bin
   ....Elapsed time for data set package or deploy operation : 4.594007
   Post-processing package:
   PackageManifest file post-processing completed.
   Create version and store package:
   ....Version artifacts stored to UCD server CodeStation
   ....Version:20190529-142402 created
   Elapsed time 42.0 seconds.

   ** buztool output properties
      version.url -> https://ucdserver:18443//#version/8244c2d4-85b1-4d6c-9df5-e184298d8583
      version.repository.type -> CODESTATION
      version.name -> 20190529-142402
      version.id -> 8244c2d4-85b1-4d6c-9df5-e184298d8583
      component.name -> J05MortgagePOT
      version.shiplist -> /var/jenkins/workspace/J05MortgageCICS/BUILD-7/shiplist.xml
   Finished: SUCCESS
```

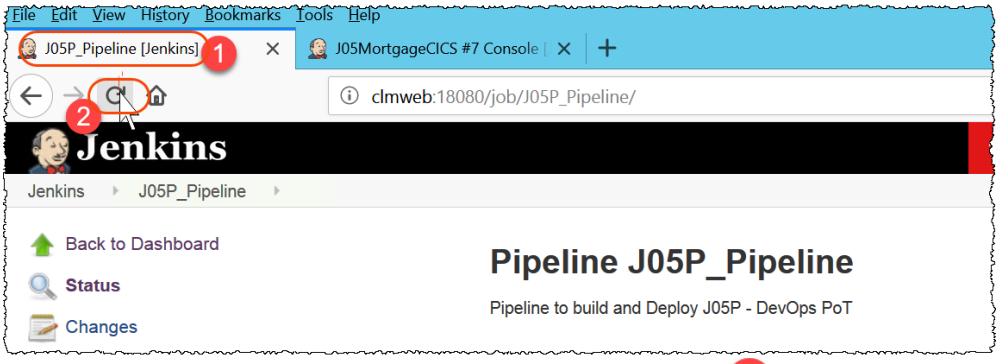
Section 7. Use Jenkins and UCD plugin to deploy results and test the CICS transaction again

You will verify the results of the second stage (deploy using UCD) .

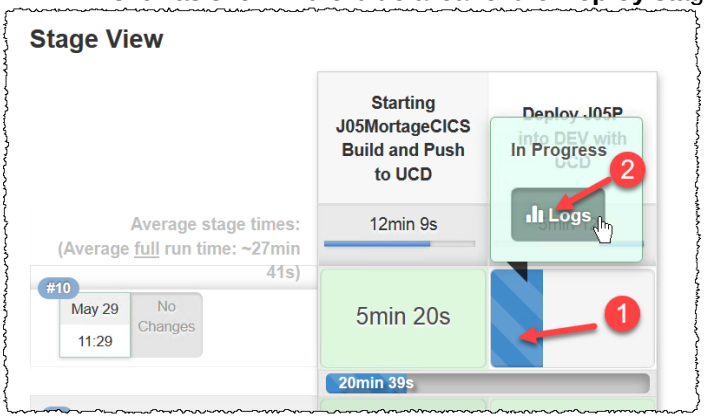
___ 7.1 Checking the deploy results (second stage)

7.1.1 ⏭ 1 Click on the previous tab **J05P_Pipeline[Jenkins]**

2 Click the Refresh icon

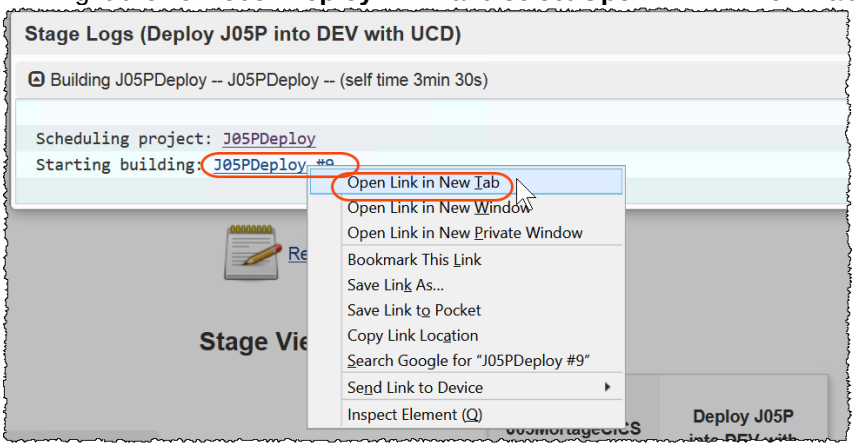


7.1.2 ⏭ Click as show in the **blue area** of the **Deploy** stage 1 and then click on **Logs** 2.

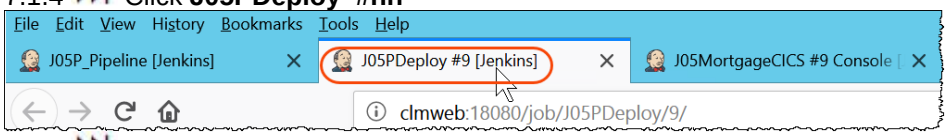


7.1.3 The dialog Stage Logs will open.

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



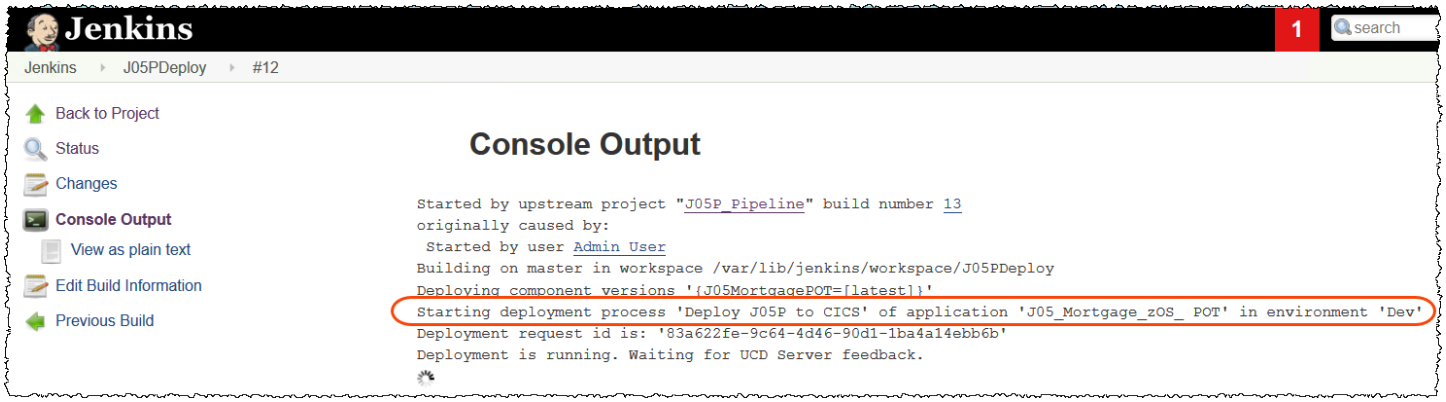
7.1.4 ⏭ Click **J05PDeploy #nn**



7.1.5 ⏭ Click on the **Console Output** to view the logs.

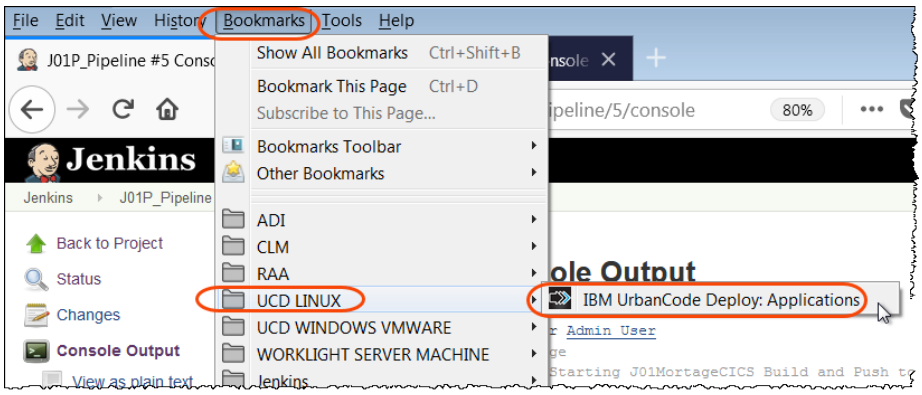
7.1.6 The Console Output log of second pipeline shows that the program that you delivered will be deployed to CICS Dev environment using UCD..

⏭ You don't need to wait for the deploy finished. Go ahead and check UCD in action (step 7.2.1 below).



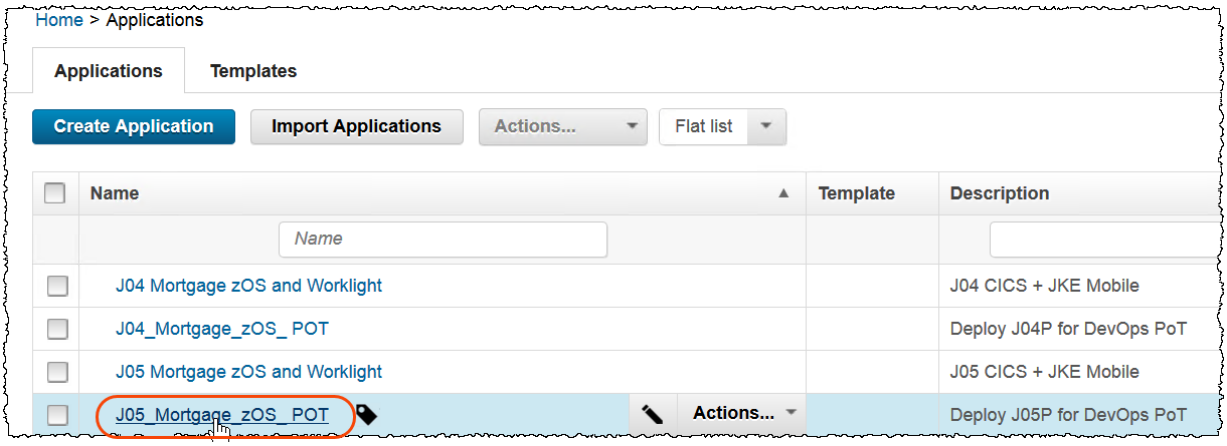
___ 7.2 Checking UrbanCode Deploy logs

7.2.1 ⏭ Start UCD console using the Bookmarks below

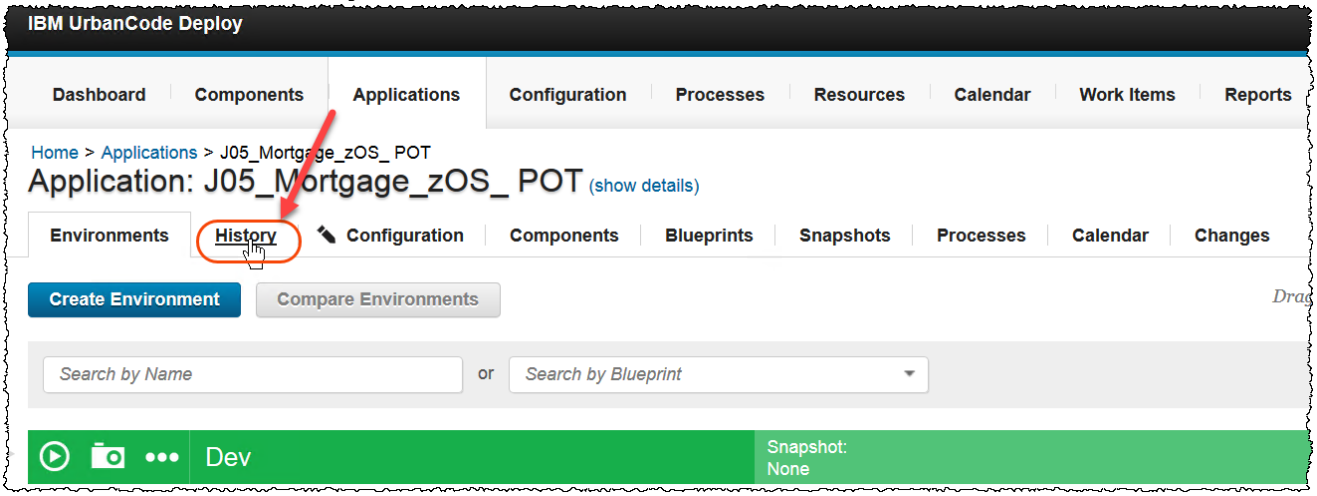


7.2.2 If required, logon using **admin** and password **admin and click Login**

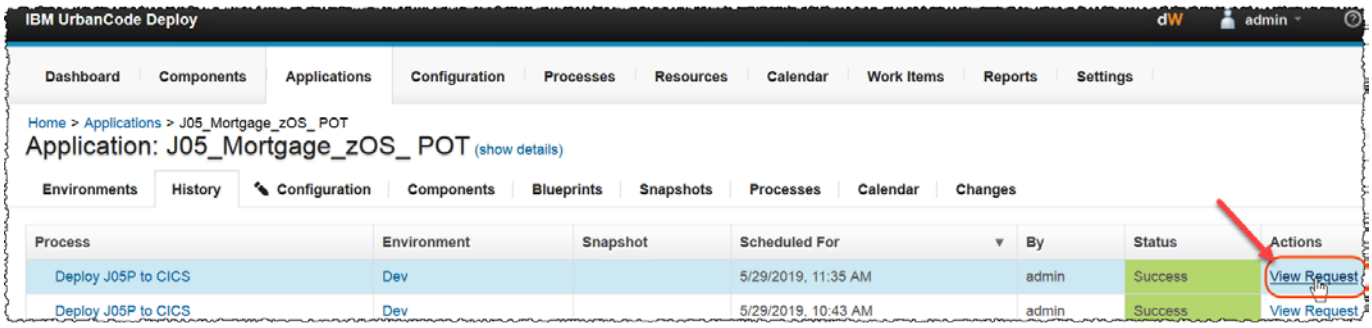
7.2.3 Find your application **`J05_Mortgage_zOS_POT`** and **click** on it



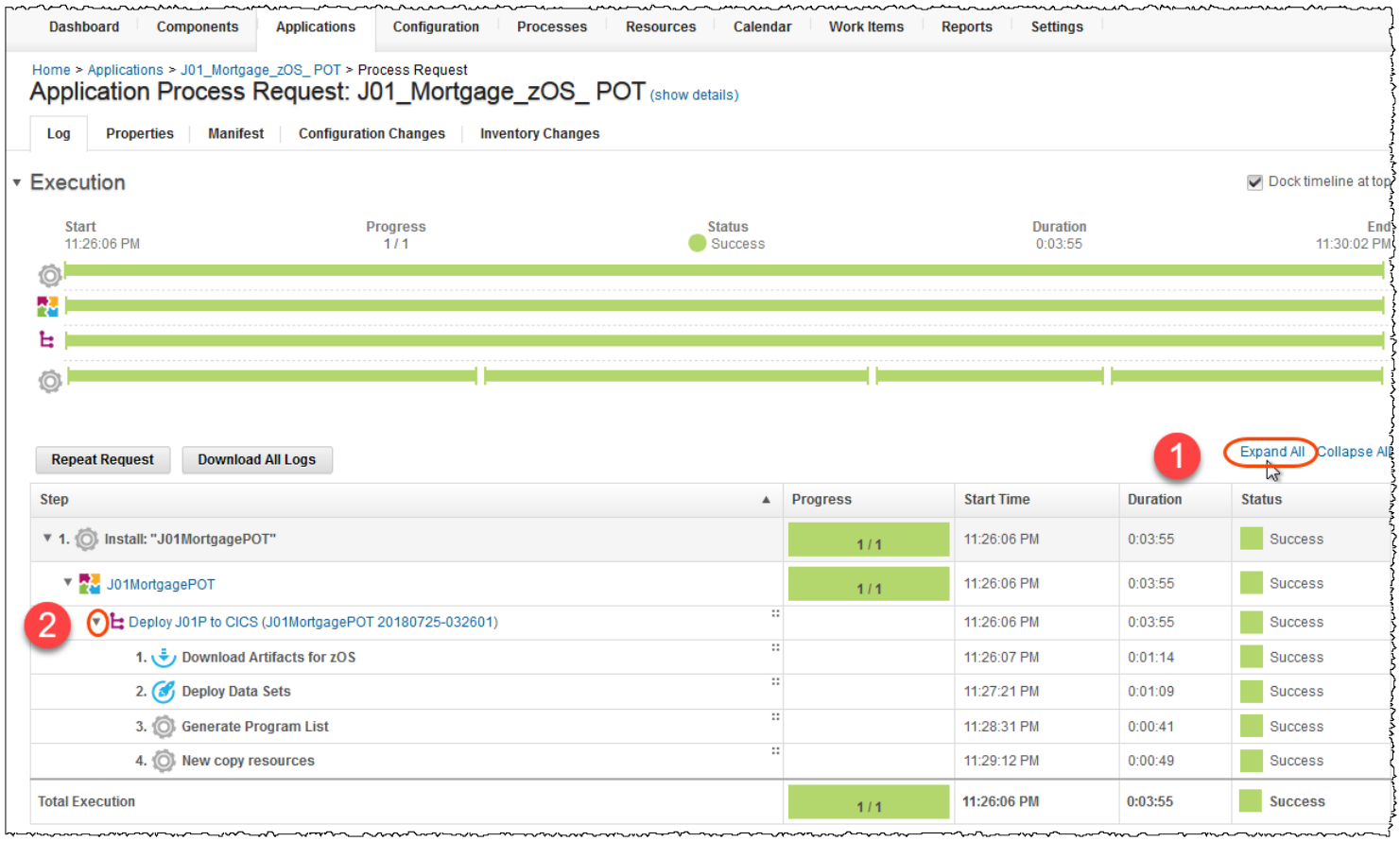
7.2.4 Click on the **History** tab



7.2.5 Click on the **View Request** entry that is related to your deploy (check the date/time)

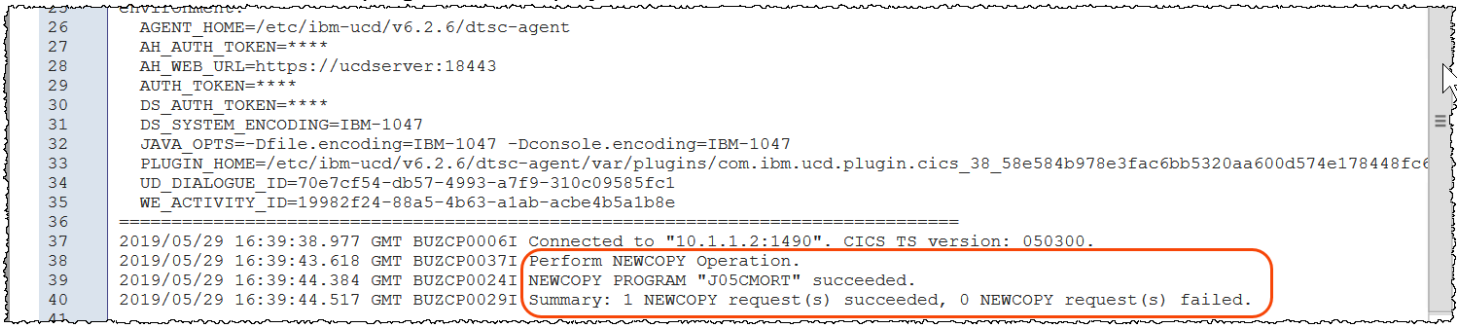


7.2.6 Click **Expand All** and **expand the node** as show in the #2 below



7.2.7 Click the output log of the last UCD step

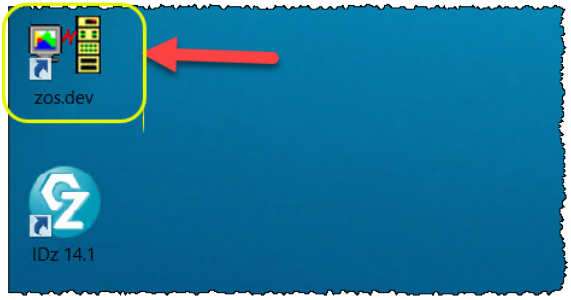
7.2.8 Notice that **J05CMORT** program was deployed to CICS Dev environment.



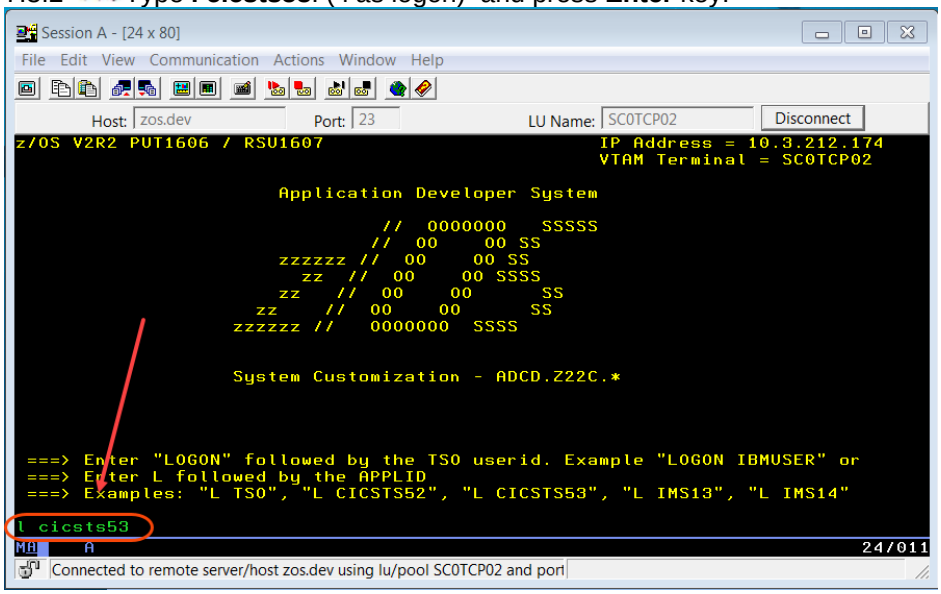
7.2.9 Minimize the browser if you will do the last optional step (#8) or close it.

___ 7.3 Testing the CICS code deployed to Dev environment

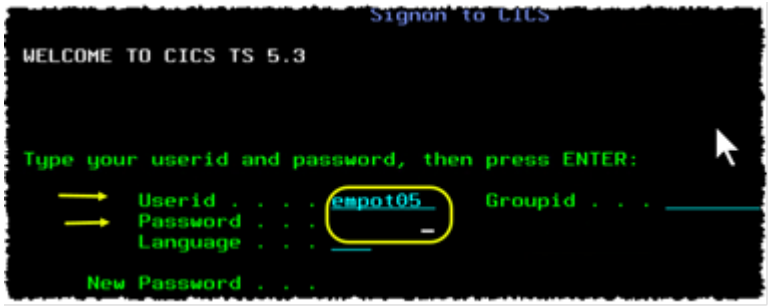
7.3.1 Double click on the **3270 terminal emulator** that is on the Windows desktop



7.3.2 Type **l cicsts53**. (l as logon) and press **Enter** key.

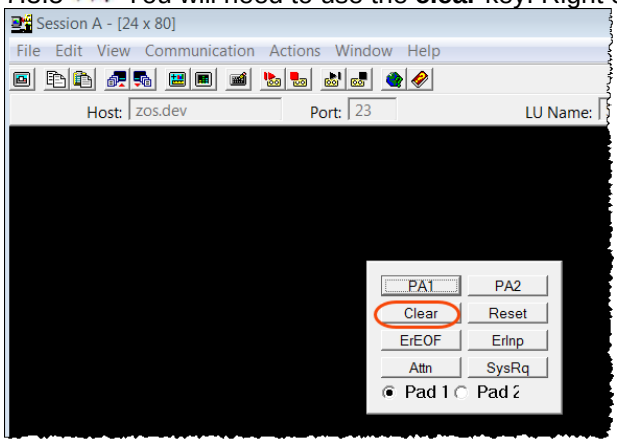


7.3.3 Logon using your z/OS user id and password (**empot05/empot05)** and press **Enter**.

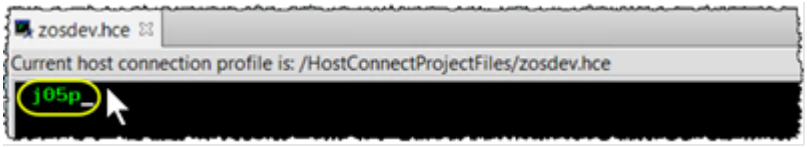


7.3.4 The sign-on message is displayed

7.3.5 You will need to use the **clear** key. Right click and select Clear.

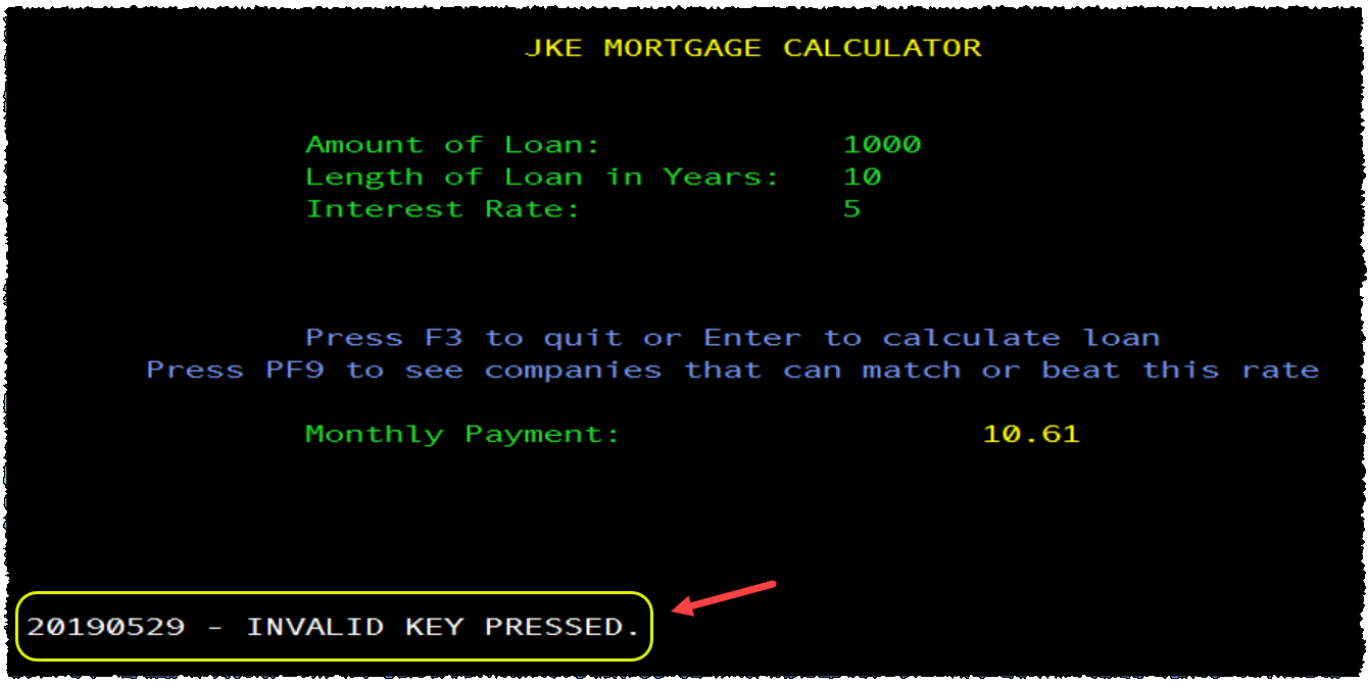


7.3.6 Type the CICS transaction **`j05p`** and press the **Enter** key.

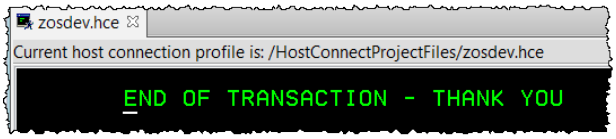


7.3.7 Press **enter** and then the **F1** key
and verify the message displayed "**YYYYMMDD - INVALID KEY PRESSED**" .

This is your change correct?



7.3.8 Press **F3** to end the CICS transaction.



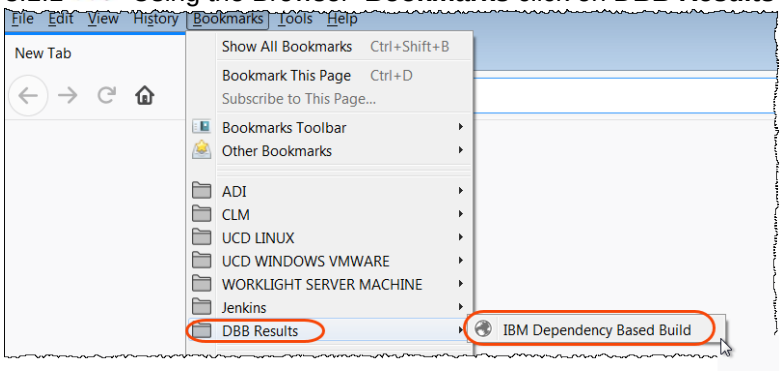
7.3.9 Close the terminal emulator.

Section 8. (Optional) Understanding DBB Build Reports

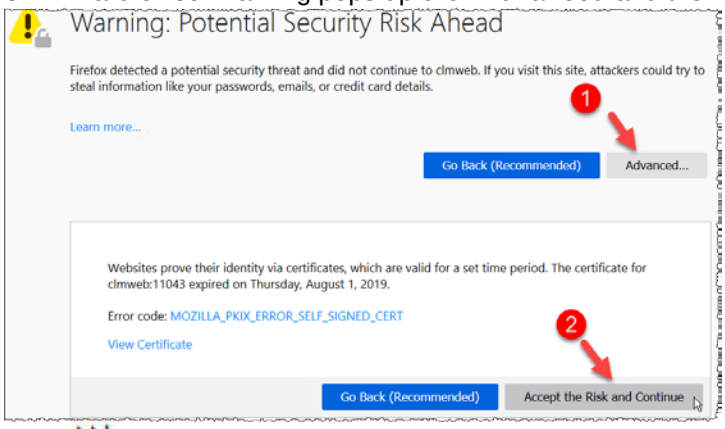
DBB has an application server (our is running on Linux) that capture the various build events. Here you will take a quick look of those reports.

___ 8.1 Login to the DBB server using the browser

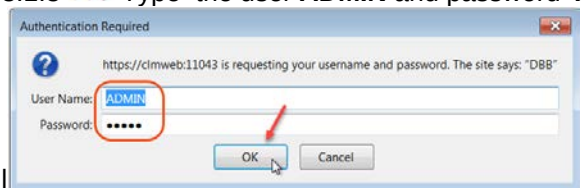
8.1.1 Using the Browser **Bookmarks** click on **DBB Results > IBM Dependency Based Build**



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**.

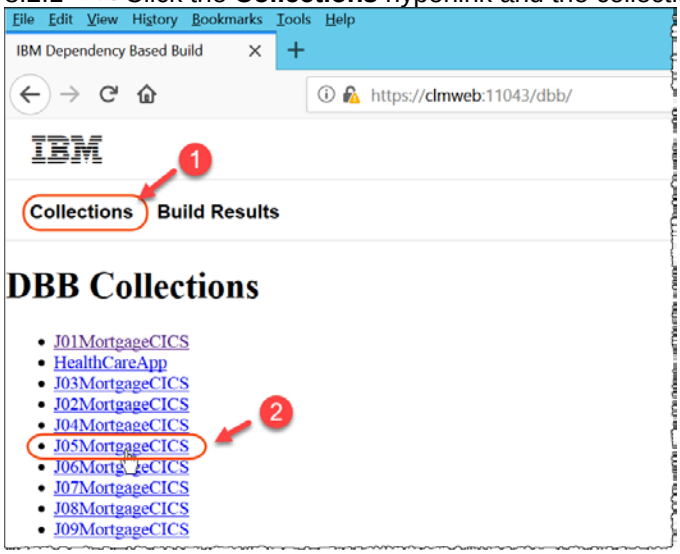
A collection is a repository container for logical files. The scope of a collection is determined by the user but generally a collection contains all the logical files of a Git branch. This way the logical files in a collection can use the scanned file's relative path from the *sourceDir* as a unique 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

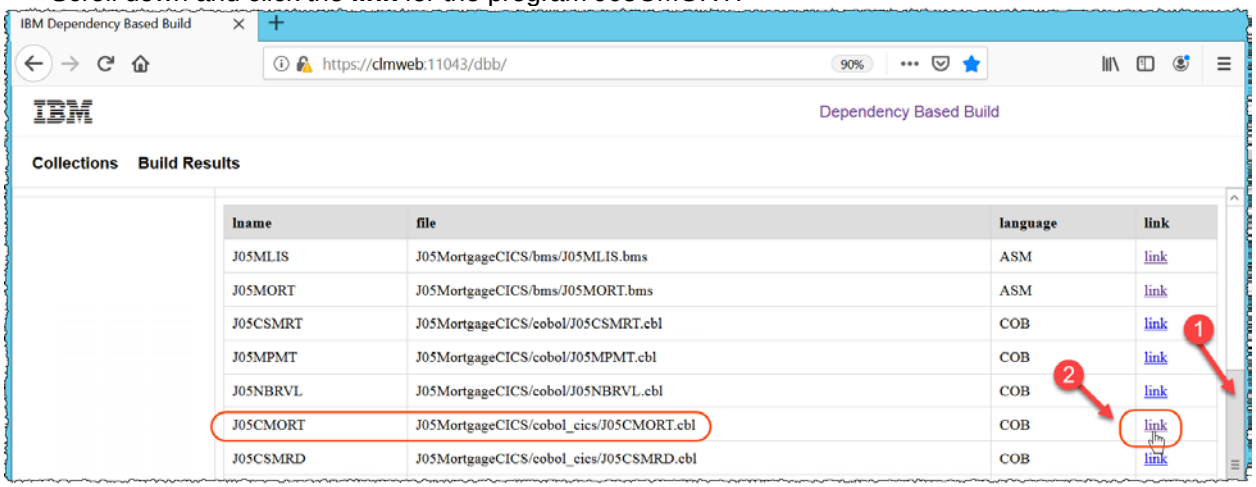
___ 8.2 Understand the DBB Collections

8.2.1 ⏭ Click the **Collections** hyperlink and the collection name **J05MortgageCICS**

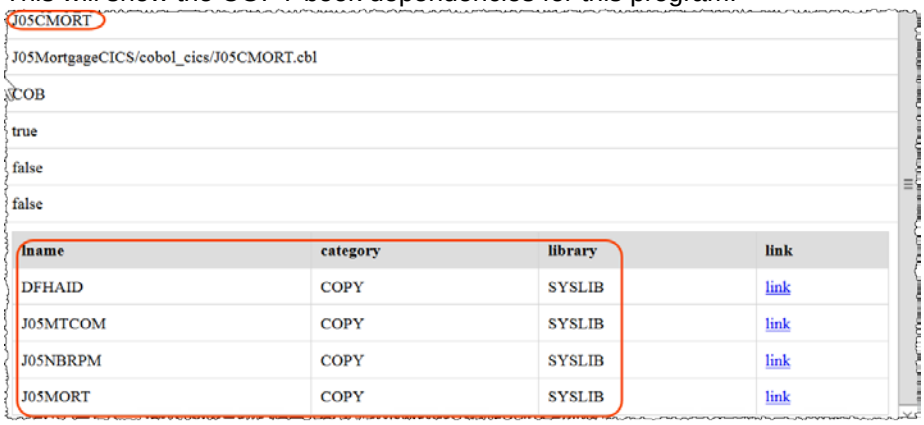


8.2.2 This opens the DBB Collection details..

⏭ Scroll down and click the **link** for the program J05CMORT.

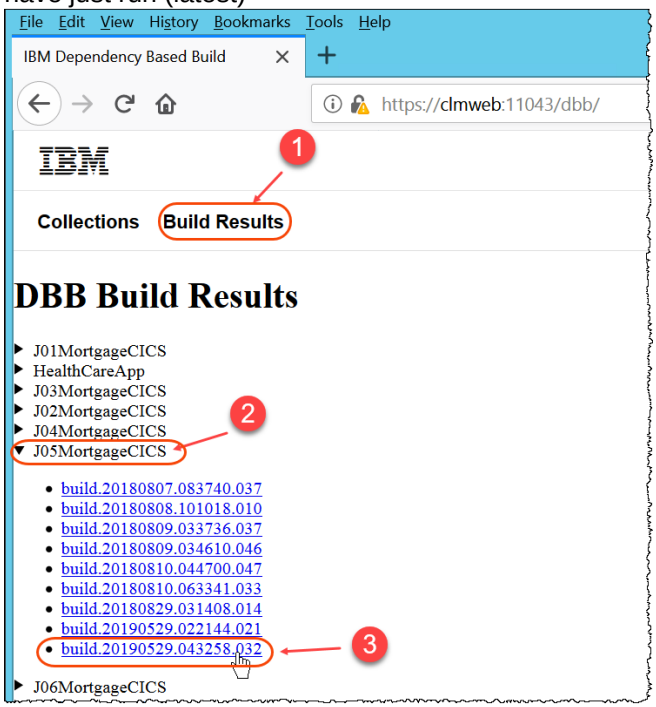


This will show the COPY book dependencies for this program.

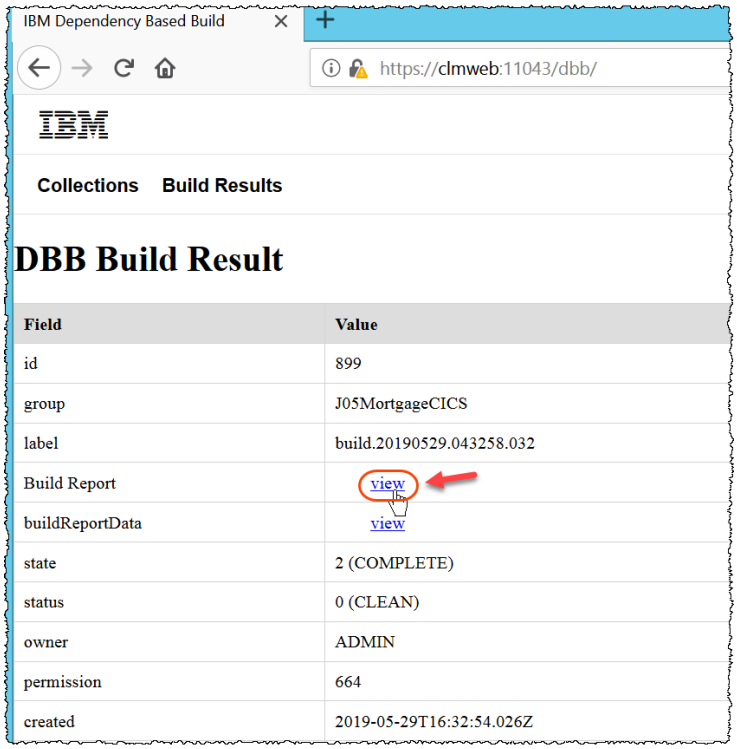


___ 8.3 Understand the Build Results

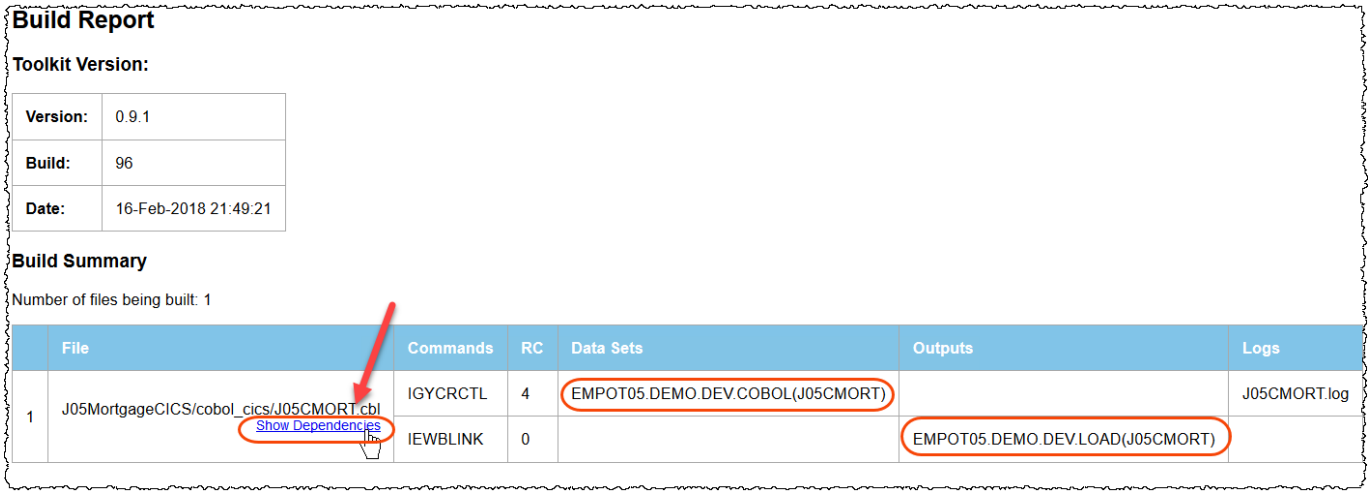
8.3.1 Click on the **Build Results** hyperlink, your collection name **J05MortgageCICS** and **the build** that you have just run (latest)



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



8.3.3 This opens the *Build Report*.. Notice the load module created on this build
You also may explorer other hyperlinks. Click **Show dependencies**.



Build Report

Toolkit Version:

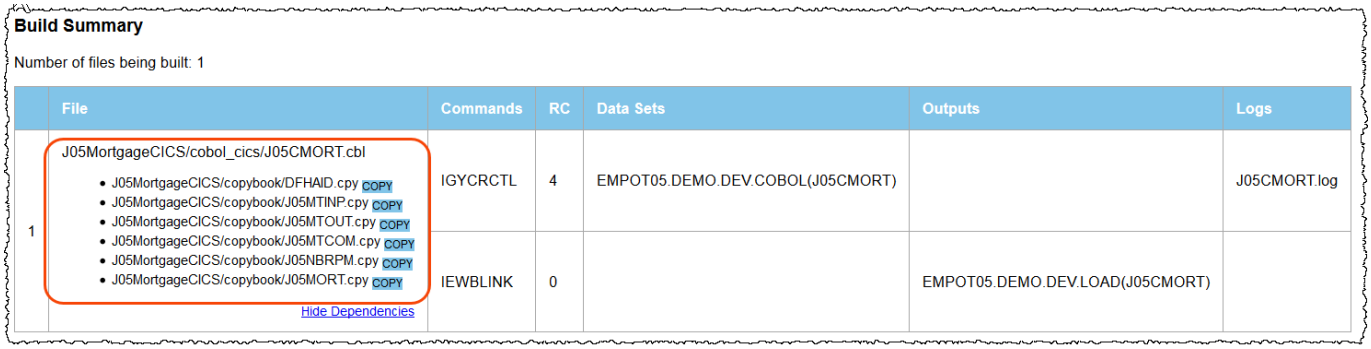
Version:	0.9.1
Build:	96
Date:	16-Feb-2018 21:49:21

Build Summary

Number of files being built: 1

	File	Commands	RC	Data Sets	Outputs	Logs
1	J05MortgageCICS/cobol_cics/J05CMORT.cbl Show Dependencies	IGYCRCTL	4	EMPOT05.DEMO.DEV.COBOL(J05CMORT)		J05CMORT.log
		IEWBLINK	0		EMPOT05.DEMO.DEV.LOAD(J05CMORT)	

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 • 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	4	EMPOT05.DEMO.DEV.COBOL(J05CMORT)		J05CMORT.log
		IEWBLINK	0		EMPOT05.DEMO.DEV.LOAD(J05CMORT)	

8.3.5 Close the browser.

Congratulations! You have completed the Lab. .

LAB 8 – Using Application Performance Analyzer (APA) (60 minutes)

Modified August 13 2019 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

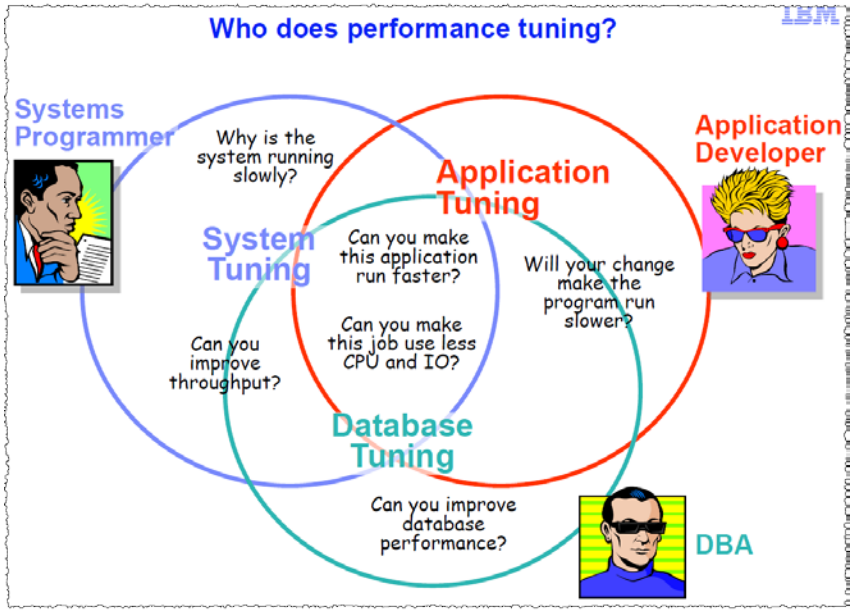
To complete this tutorial, you will perform the following tasks:

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.

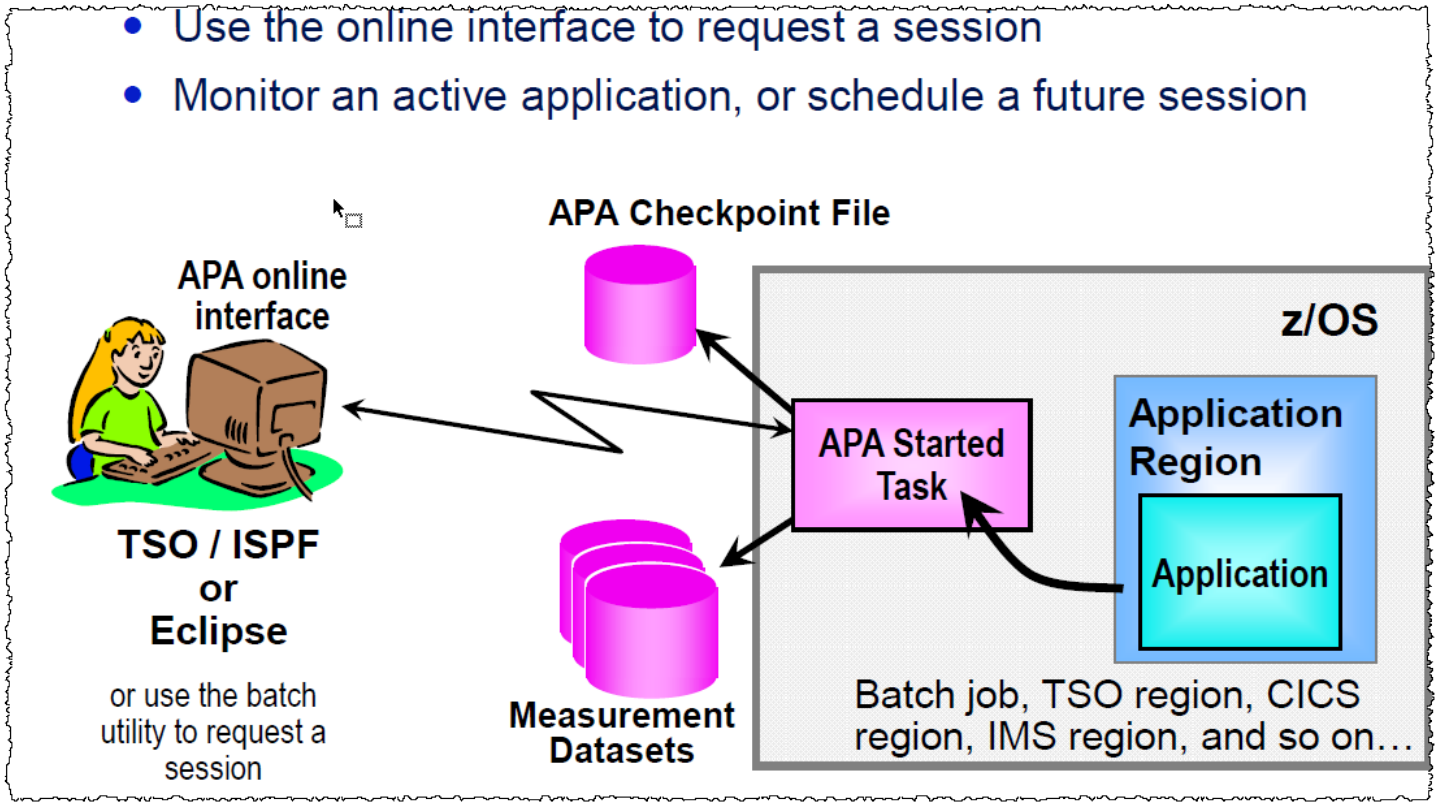
The tool aids application design, development and maintenance cycles. It helps evaluate applications in the design phase, measure the impact of increased data volume or changes in business requirements, and generate historical reports to analyze performance trends.



Section 1. Create a new observation request for a job that is not running yet

For this lab you may use empot05 user id

You will create an **Application Analyzer Observation** for an existing program to be executed via JCL submission.



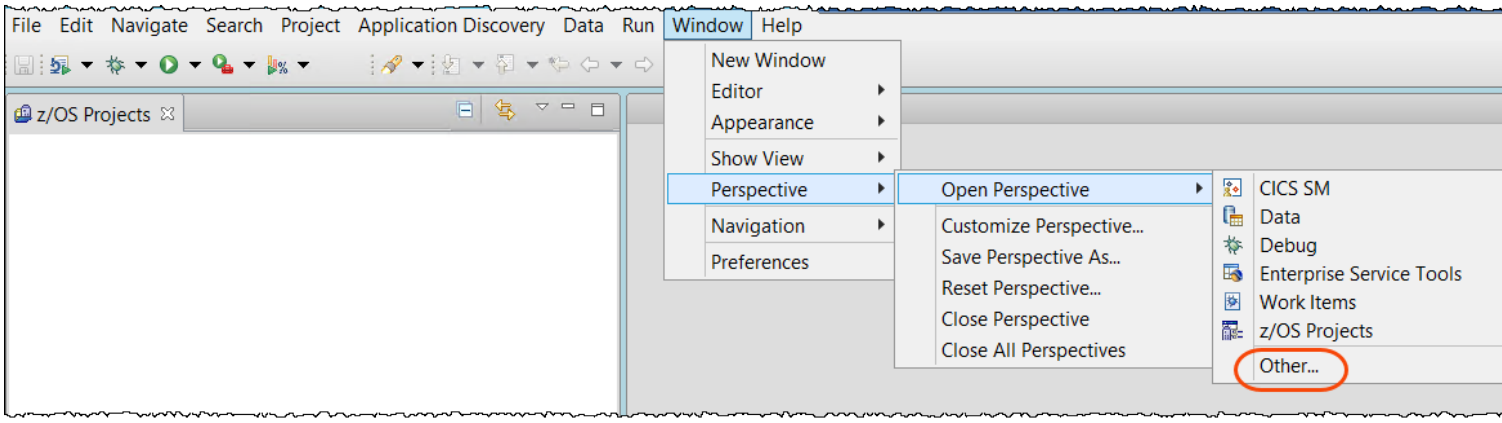
___ 1.1 Connect to the ADFz Common Components

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.

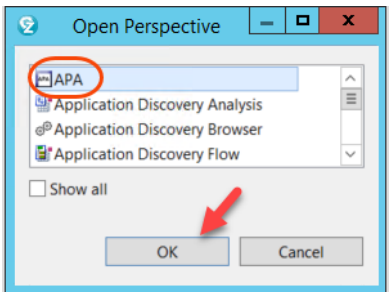
1.1.1 Start *IBM Developer for z Systems* If not already started..

⏭ Double click **IDz 14.1** icon

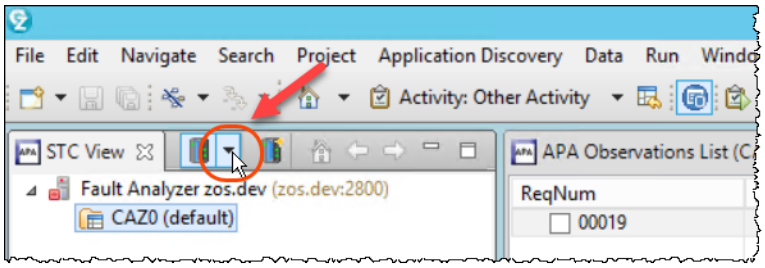
1.1.2 ⏭ Open the **APA** perspective by selecting
Window > Perspective > Open Perspective > Other..



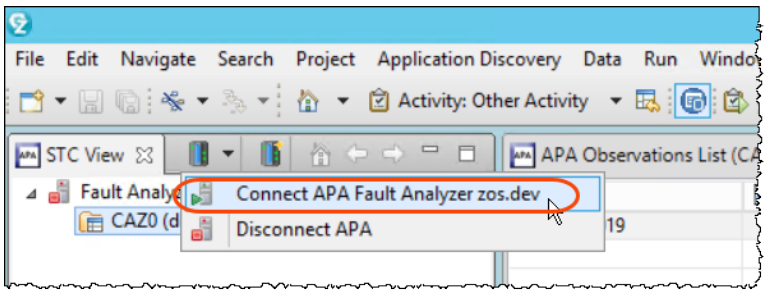
1.1.3 ⏭ Select **APA** and click **OK**



1.1.4 ⏭ On the STC View, **click APA Connection Actions icon**



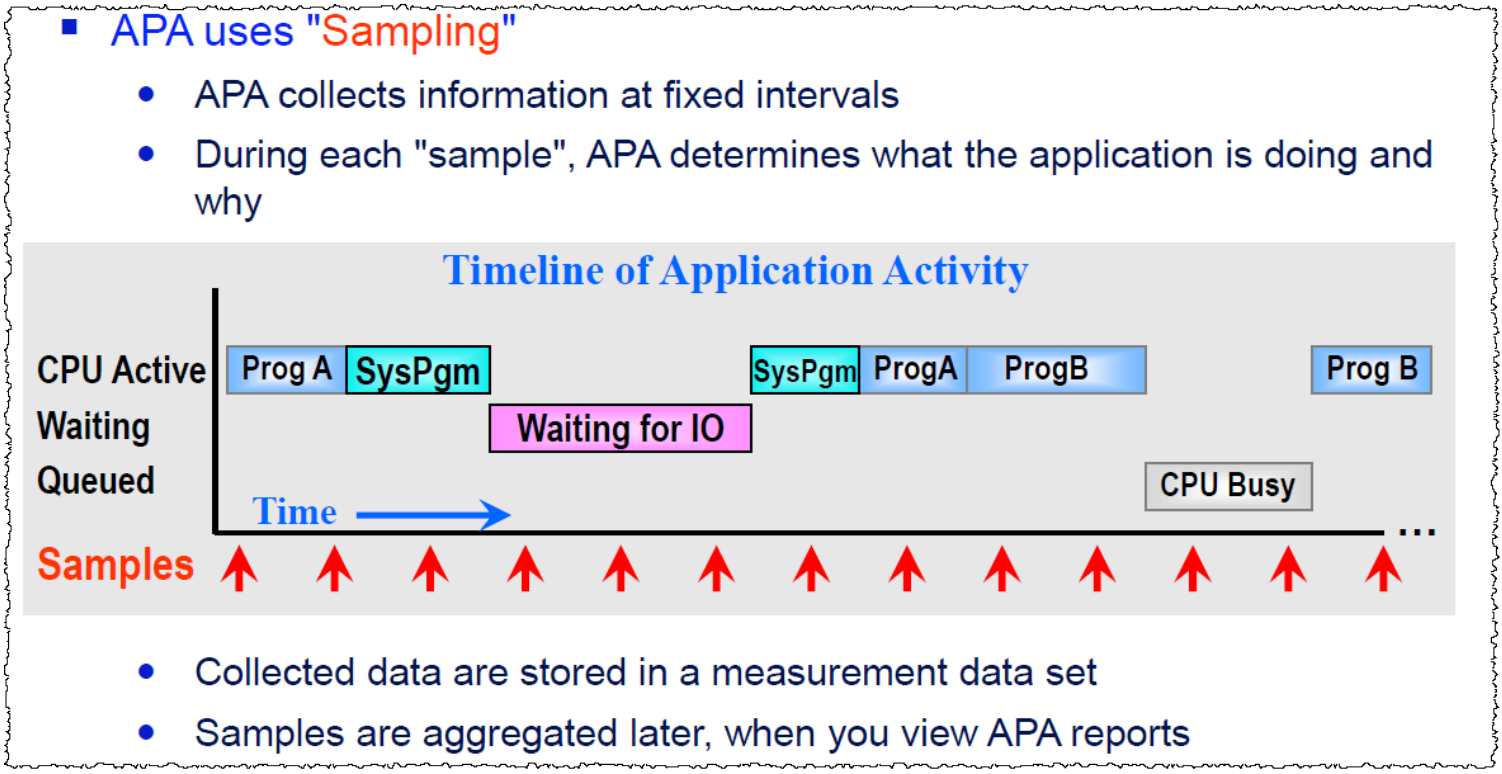
1.1.5 ⏭ Then select **Connect APA Fault Analyzer zos.dev**



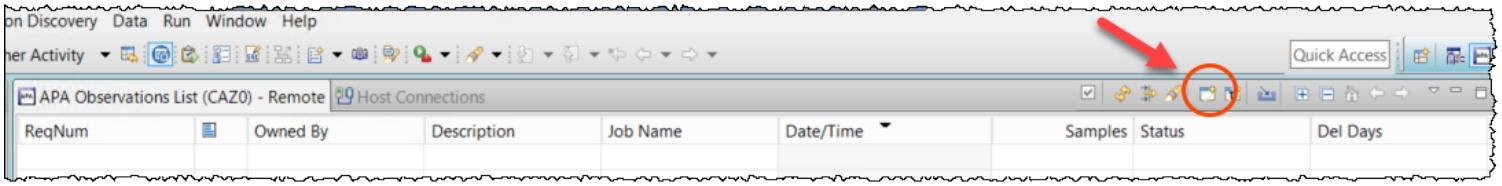
You will see some few pop-ups and this view will lose the focus once the connection succeeded.

___ 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)



1.2.1 Click on the **New Observation** icon .

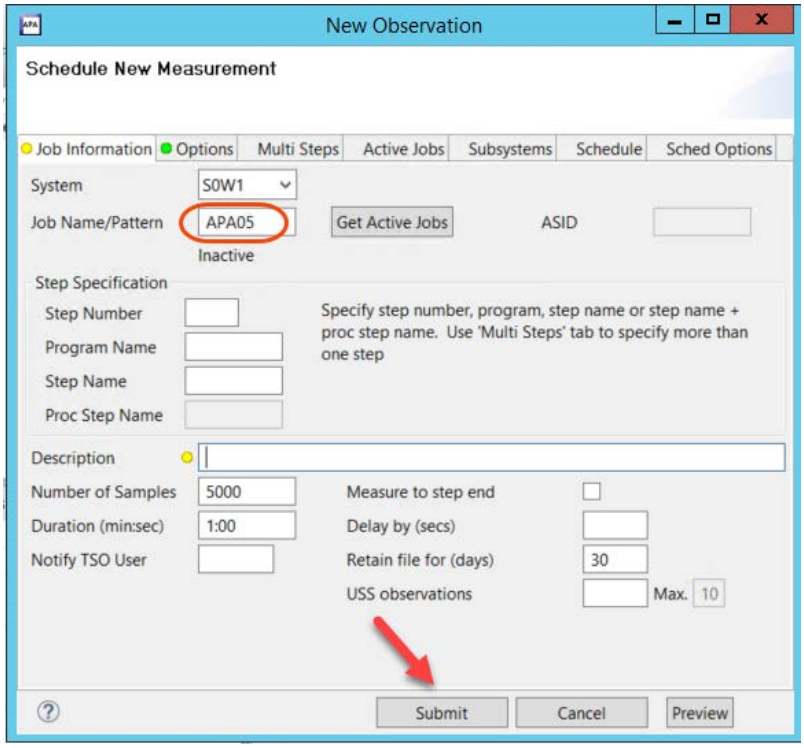


This will open the Schedule New Measurement dialog.

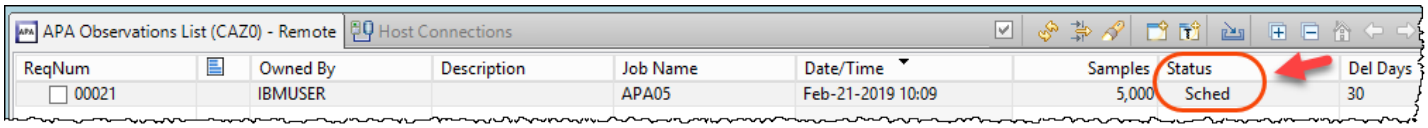
1.2.2 ⏭ On *Job Name/Pattern* type **APA05**.

The job name that you will submit later will be APA05.

You will keep the number of samples as 5000 and duration 1 minute (defaults)



1.2.3 ⏭ Click **Submit**.
Your new observation request has been added and APA is now waiting for your job to begin (*Sched*)



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.

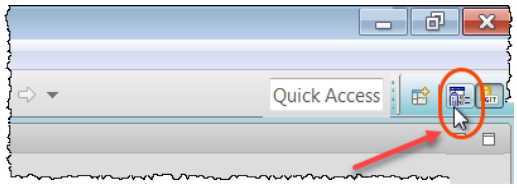
Section 2 – Run a sample batch job to collect performance data

You will submit a COBOL/DB2 batch job named **APA05** and let APA collect performance data.

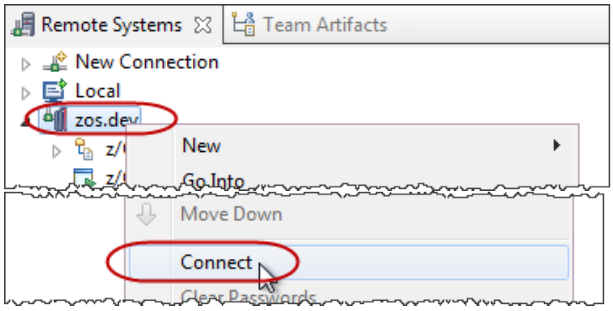
___ 2.1 Connect to z/OS

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 empot05, 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

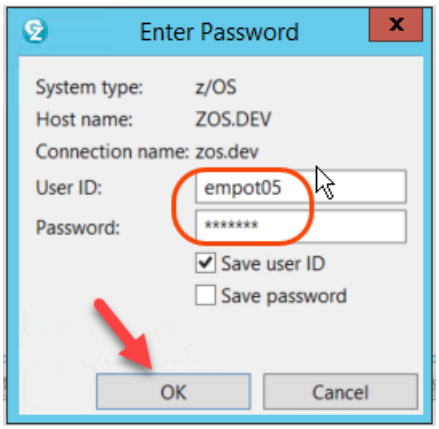


2.1.2 ⏭ If you are connected using other userid than empot05 (like empot01), you must disconnect first.
⏭ Right click on **zos.dev** and select **Connect**



2.1.3 ⏭ 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.

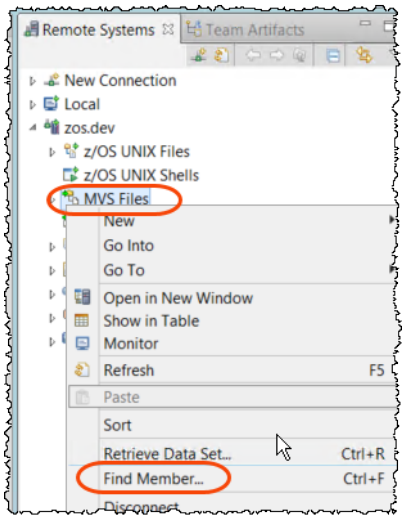
Notice → On this lab your userid will be **empot05** (not empot01 used in Lab 1)



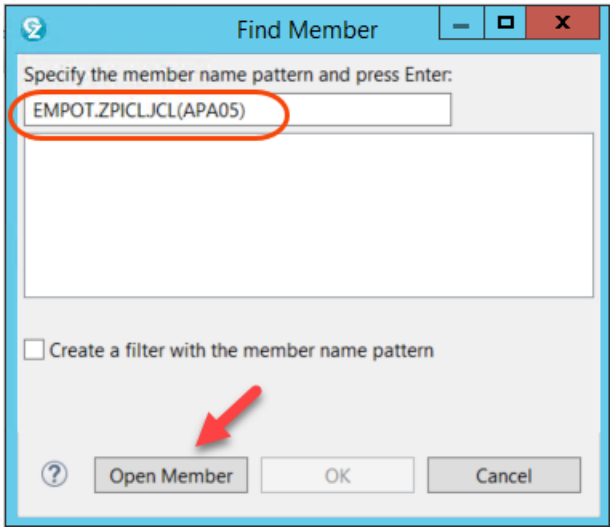
___ 2.2 Find the JCL to be submitted

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…**.

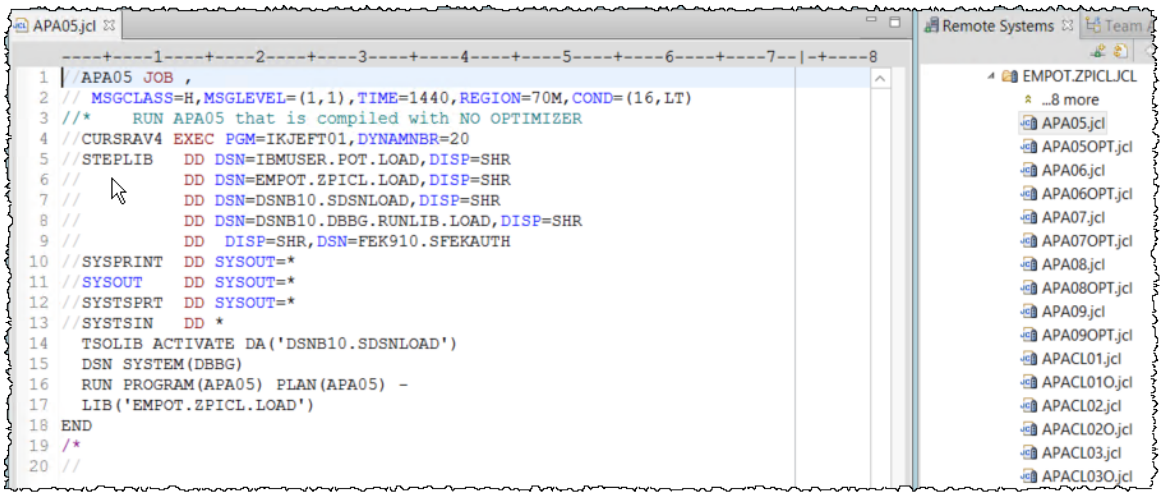


2.2.2 ⏭ Type **`EMPOT.ZPICL.JCL(APA05)`** and click **Open Member**.

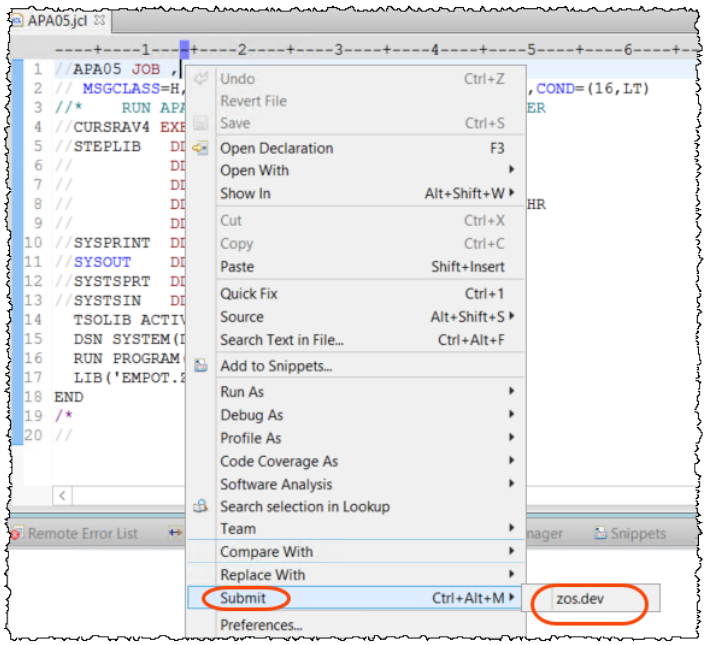


___ 2.3 Submit the JCL to execute the COBOL/DB2 batch program

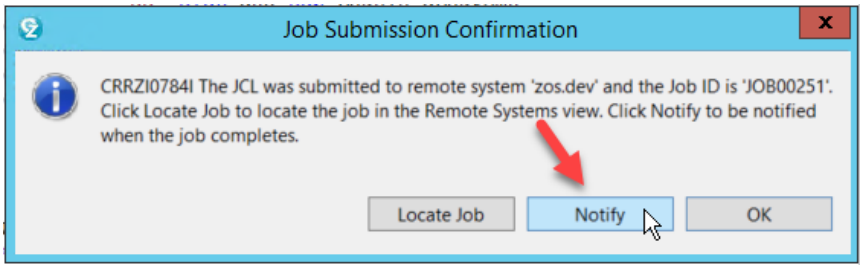
The member **APA05.jcl** will open and you must submit for execute on z/OS.



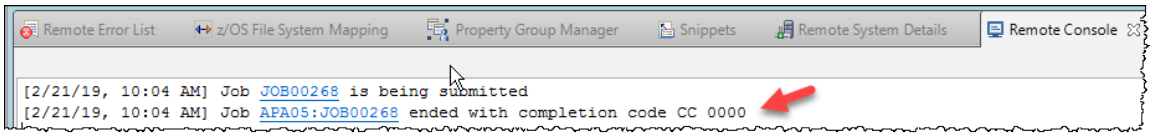
2.2.1 **Right click** in any place of the member being edited and select **Submit** > **zos.dev**



2.2.2 Click **Notify** . Note on Remote Console view that the job is being submitted



2.2.3 On the bottom you will see the Remote Console view. The completion code must be 000. If this is not the case call the instructor.



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

What have you done so far?

You created an APA observation request for a job that you have submitted.
The observation collected less than 5000 samples since it run in less than 1 minute.

What information does Application Performance Analyzer provide?

- APA reports detailed information about:

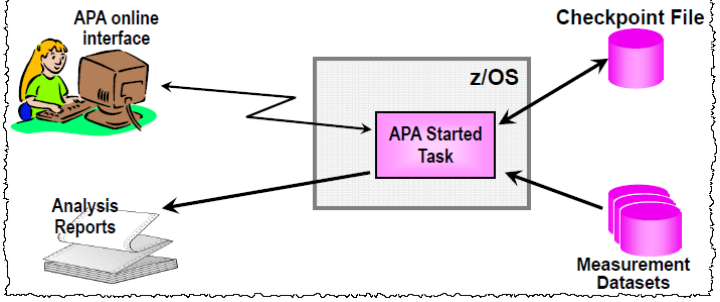
Category:	Sub-categories (partial list) :
CPU usage	by Module, by Statement, by Timeline, …
WAIT time	by Module, Statement Attribution, …
DB2	various Metrics by SQL stmt, Plan, Thread, …
CICS	various Metrics by Txn and by Task, …
IMS calls	by CPU Time, Call Service Time
DASD usage	by Device, by File, by Timeline, …
HFS usage	by Device, by File, by Timeline, …
MQ	by Queue, by Request, by Transaction …
Storage	by Device, by Task, by Timeline, …
And more	

Section 3 – Review some of the reports created.

APA produces analysis reports. On this section you will explore some of the reports produced. Notice that you could print, create a PDF or export in XML format.

APA produces analysis reports

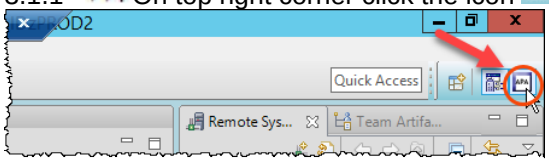
- View and print analysis reports
  - View sample file data from the APA online interface
  - Or print performance reports to SYSOUT or to a PDF file
  - Sample file data can also be written in XML format



___ 3.1 Download the your most recent APA observation reports

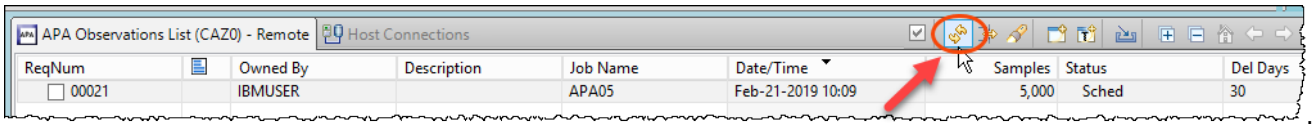
The reports can be seen using the APA perspective.

3.1.1 ⏭ On top right corner click the icon to switch to the APA perspective

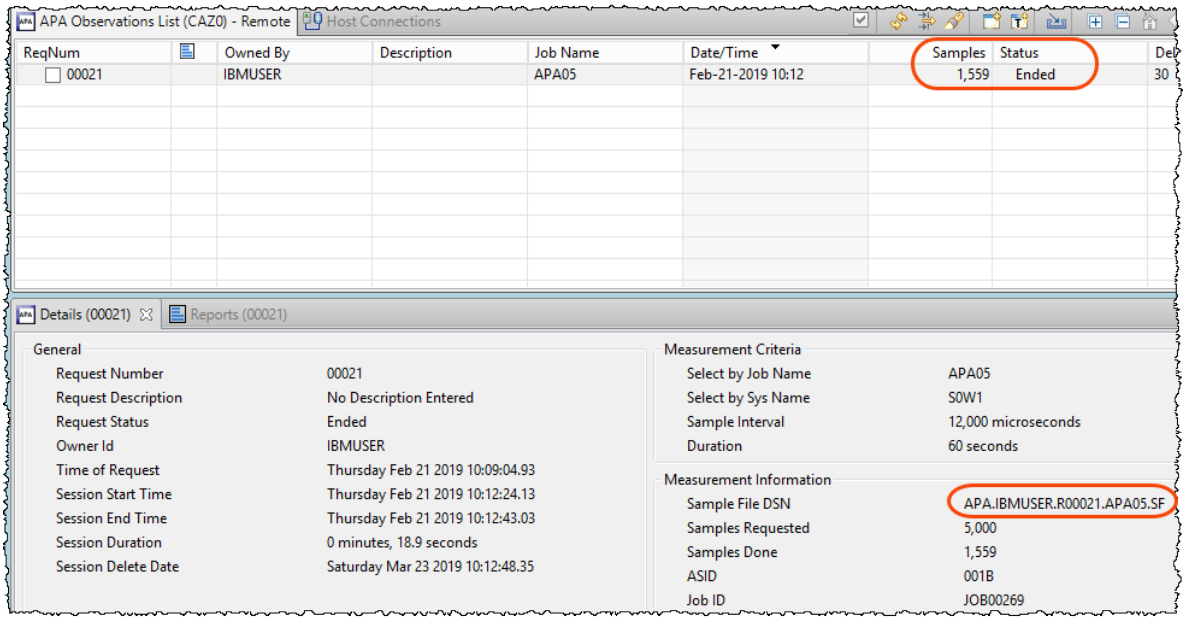


Observation Reports List Overview
The reports list is a 2-level tree-view. The first level (parent) rows represent the report category while the second-level (child) rows are for the individual reports.
The category rows are for informational/organizational purposes only and result in no action if clicked (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

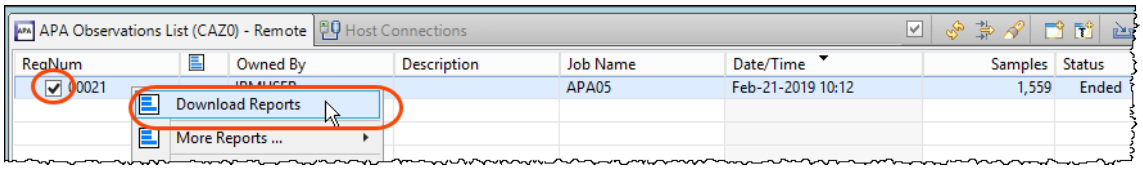


Notice that the Status change to **Ended** and also the *Details* view depicts information about the APA measurement. The name of the sample file (‘*APA.IBMUSER.xxx*’) shown below, circled in red, is displayed, which was automatically created by APA to hold the results of the observation session. This file is tied to the observation request. On your lab probably the owner will be EMPOT01 instead of IBMUSER shown on the screen captures..

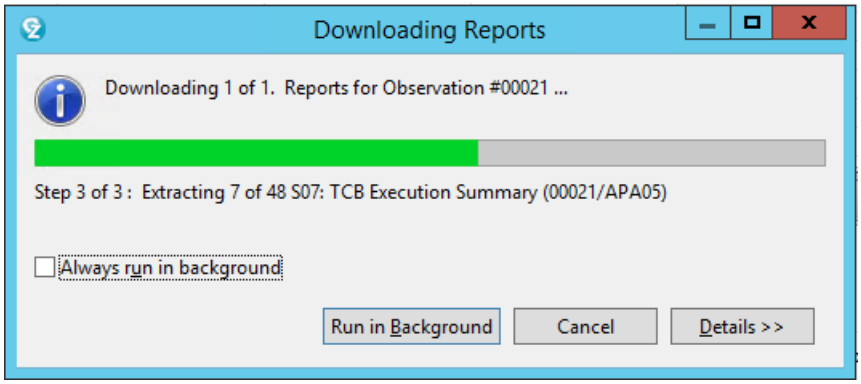


Notice that around 1,500 samples were collected instead of the 5,000 requested.
The reason is that the job duration was less than one minute requested (your value may be different).

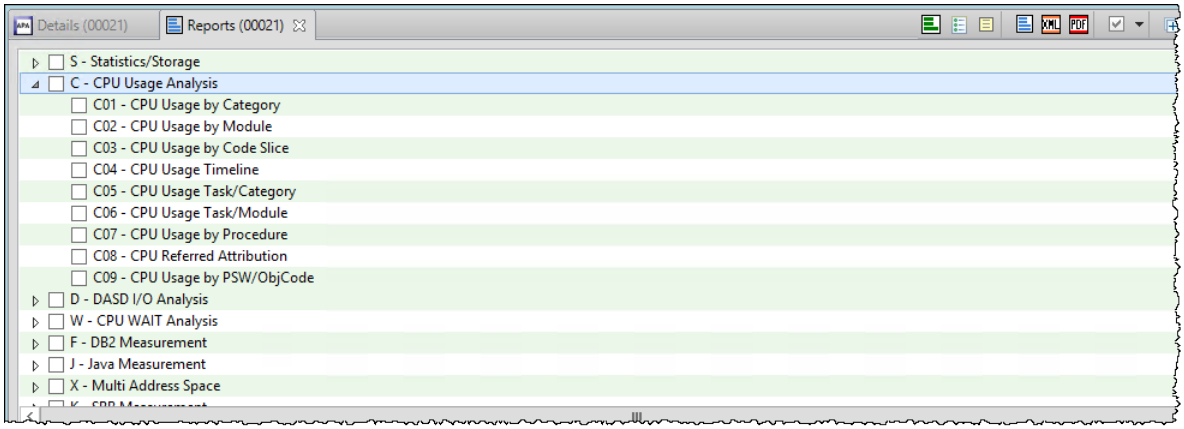
3.1.3 ⏭ Click on the box ☐ (on left) to select the observation, **Right click** on observation and select **Download Reports**. This will download the created reports to your workspace.



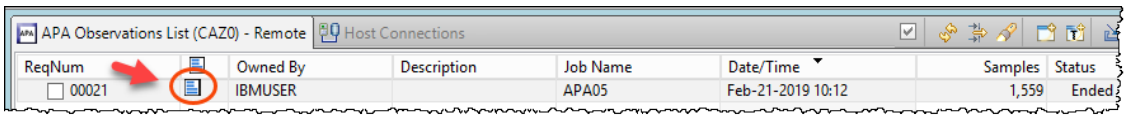
This operation may take a while, depending on your system, network, etc..



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).

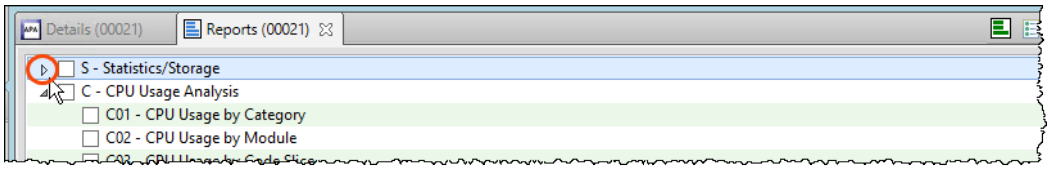
You may have access if interested on see the source code
This program is compiled with **NOOPTIMIZE** option. It executes OPEN/FETCH/CLOSE on a small DB2 table 300 times:

```
       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 **Expand the S** reports above by clicking on the icon on the left of **S - Statistics/Storage**

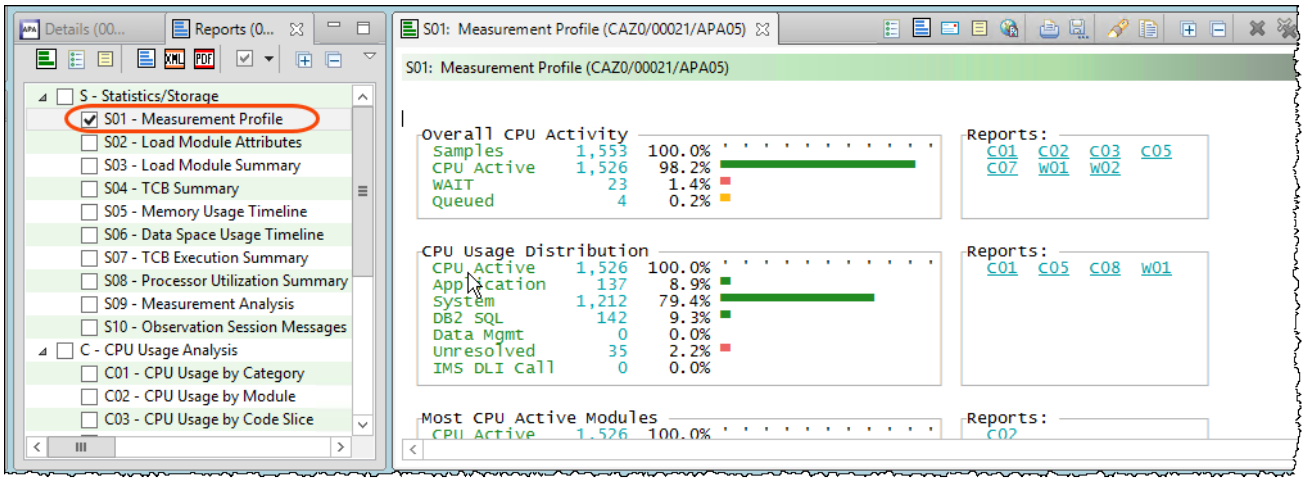


3.2.2 **Double click** on the **S01** report. The *Measurement Profile report* is displayed.

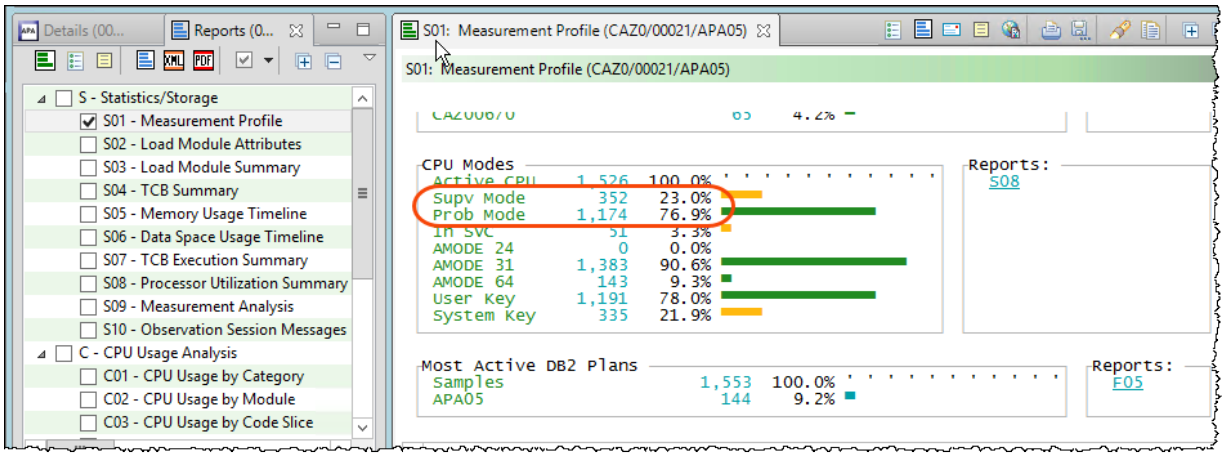
This is a good report to examine first when analyzing a measurement. It provides an at-a-glance summary of various aspects of the measurement data and helps you choose which other reports to concentrate on. The first section of this report consists of a series of mini performance graphs illustrating 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**.



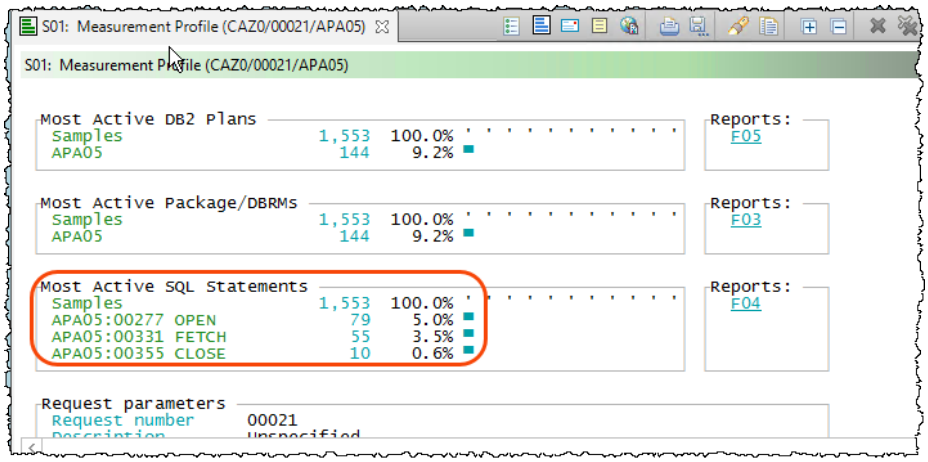
3.2.3 **Scroll down** to the remainder of the S01 report.



Notice 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**.

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.

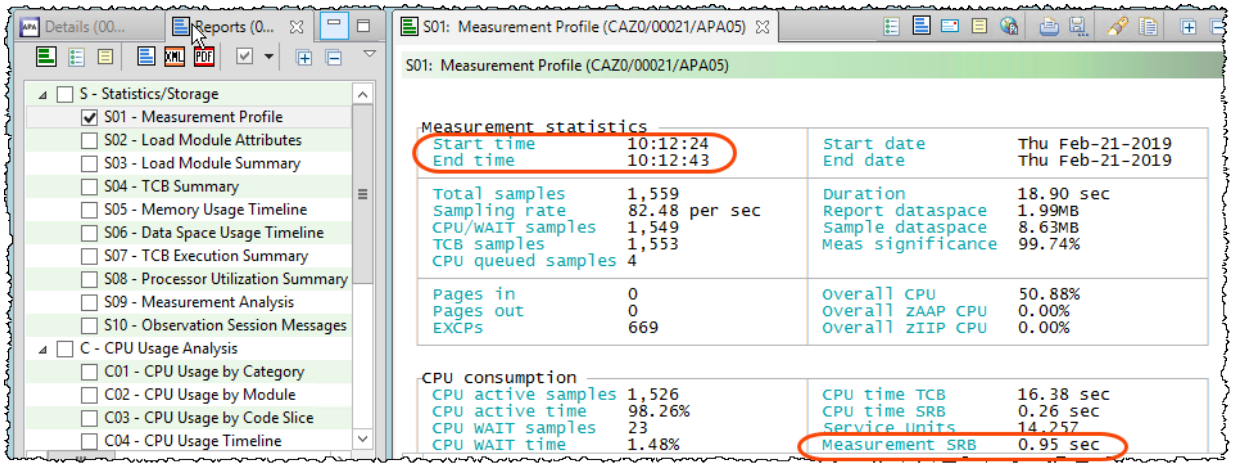


Notice that 5.0% of the DB2 CPU usage is on the SQL OPEN statement.

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.. .

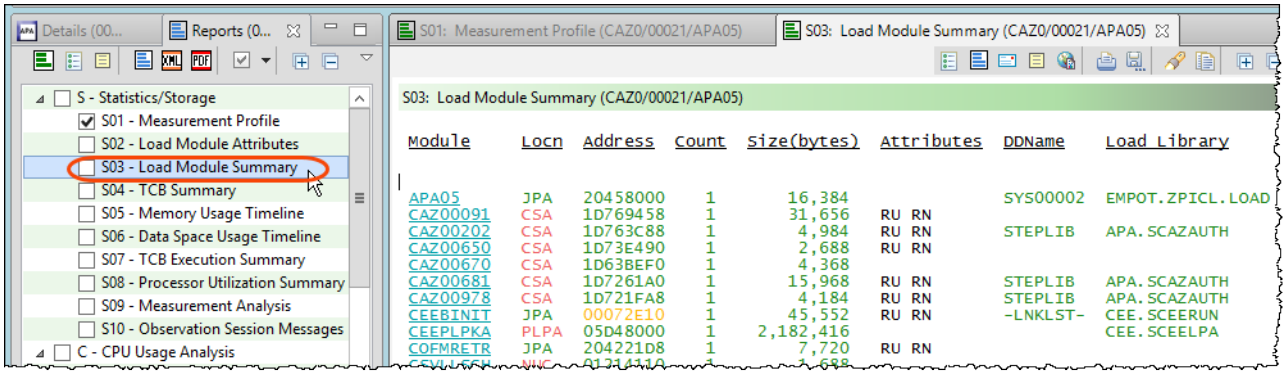
3.2.4 **Scroll down** to the bottom of the S01 report.
During monitoring, APA typically does not attach anything to the application in order to collect performance data. The run time of a monitored application should not be affected by APA. It does require CPU time in its own address space, and this is reported as the **Measurement SRB value** (0.95 sec).



___ 3.3 Analyzing the APA Load Module Summary report (S03)

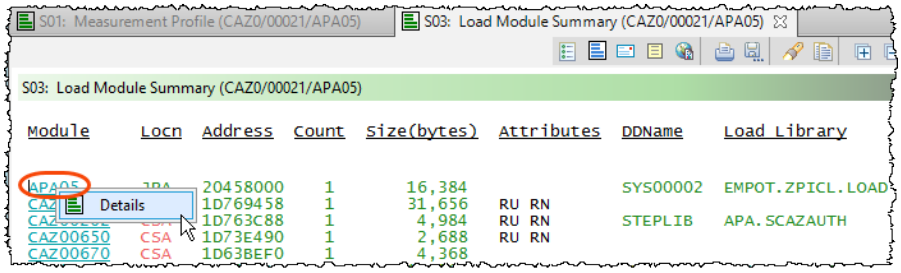
This report can be useful to find which compile options were used for the module creation.

3.3.1 **Double click** on the **S03** report. The *Load Module Summary* report is displayed.

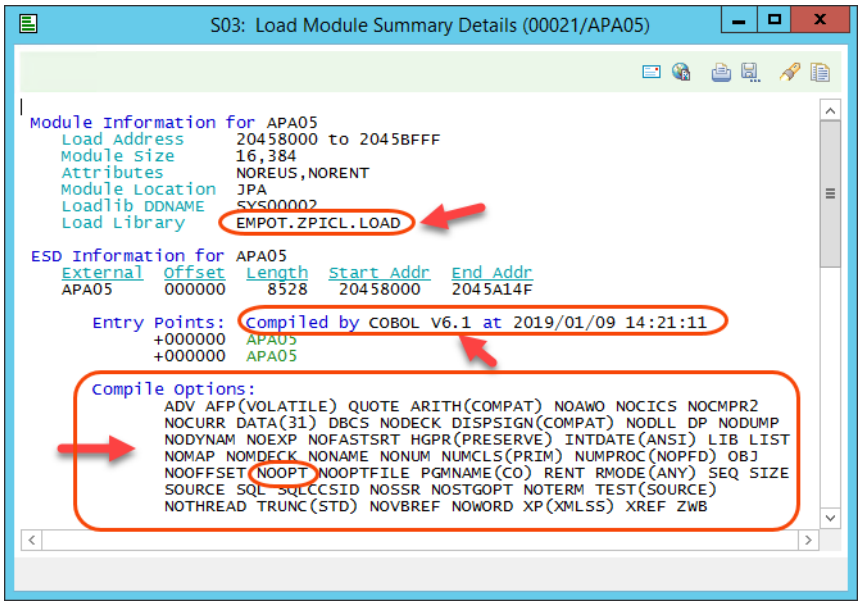


From here you can find that this program was loaded from the dataset `EMPOT.ZPICL.LOAD`.

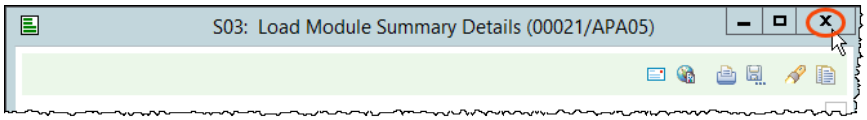
3.3.2 **Right click** on **APA05** and select **Details**



You may find interesting information as seen below, like which COBOL version the program was compiled, the compile options (like **NOOPT** – NO Optimizer), etc…..



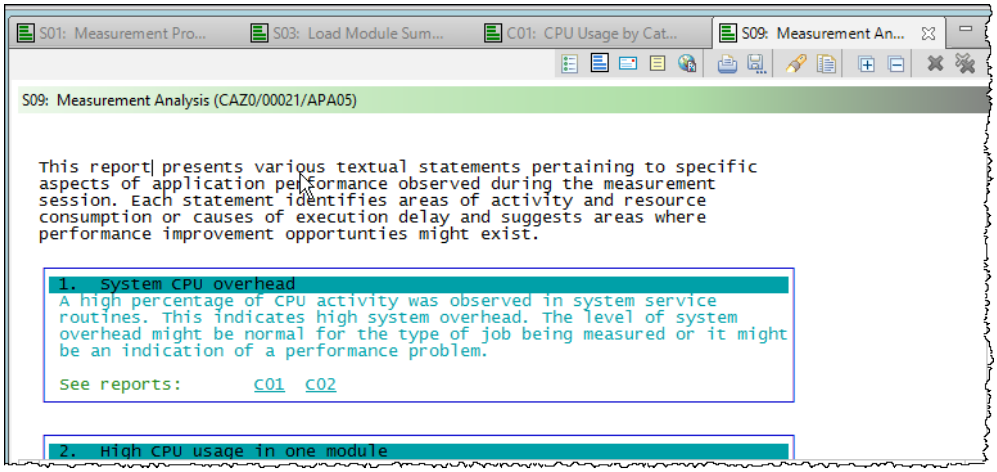
3.3.3 ⏭ Close the *Load Module Summary Details* clicking on the ✕



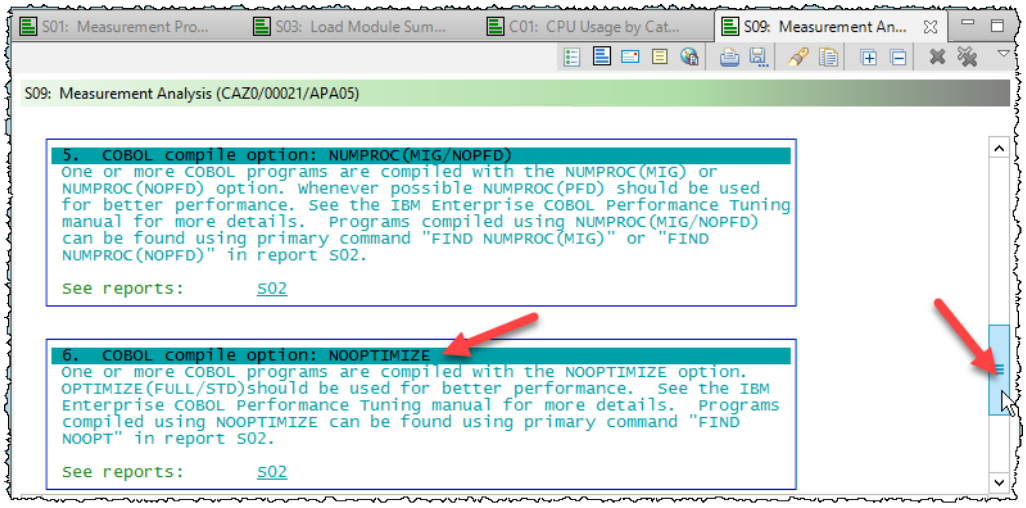
___ 3.4 Measurement Analysis report (S09)

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 OPTMIZER that could affect the performance.

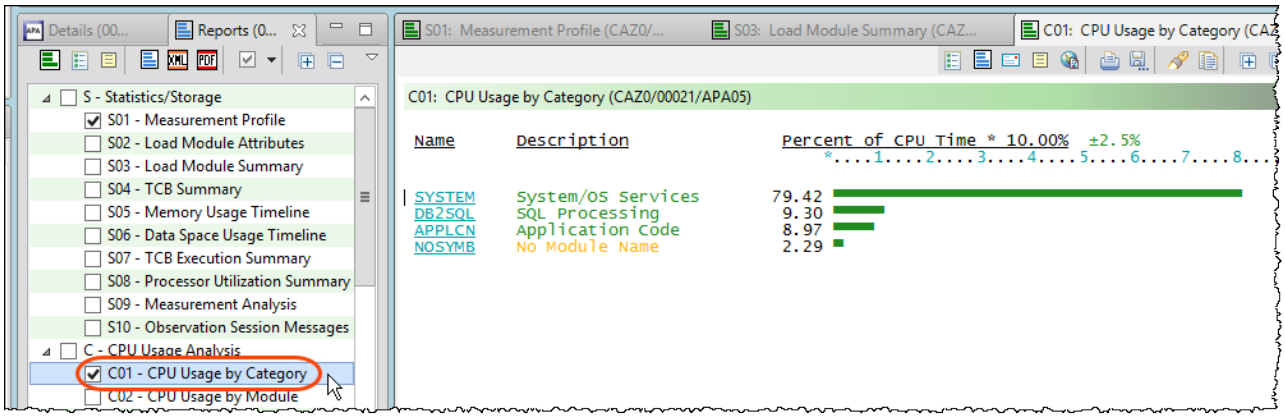


TIP: *Interested in execute this program again but compiled with OPTMIZE(2)?*
The JCL stored in **`EMPOT.ZPICL.JCL(APA05OPT)`**executes the SAME COBOL program, but compiled with **`OPTIMIZE(2).`**
Once you finish this lab you may create another observation for this job and analyze the results…
You will notice some CPU reduction…

___ 3.5 Analyzing the APA CPU Usage by Category report (C01)

This report can be useful to find which compile options were used for the module creation.

3.5.1 ⏭ **Double click** on the **C01** report. The *CPU Usage by Category* report is displayed.
The CPU Usage by Category shows the major collections of CPU activity with drill down capabilities to the Load Modules, CSECTS, and Program Source statements in each category.

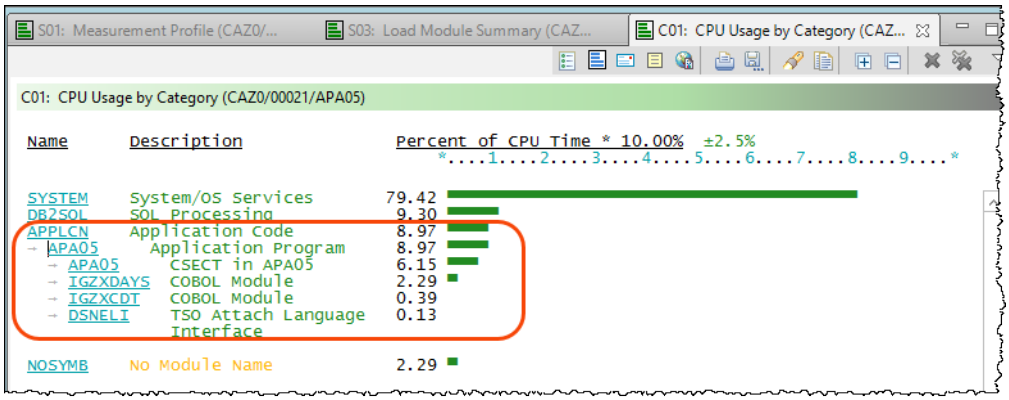


3.5.2 ⏭ Click on the **APPLCN** collection to see the names of the Load Modules which appear in the sampling

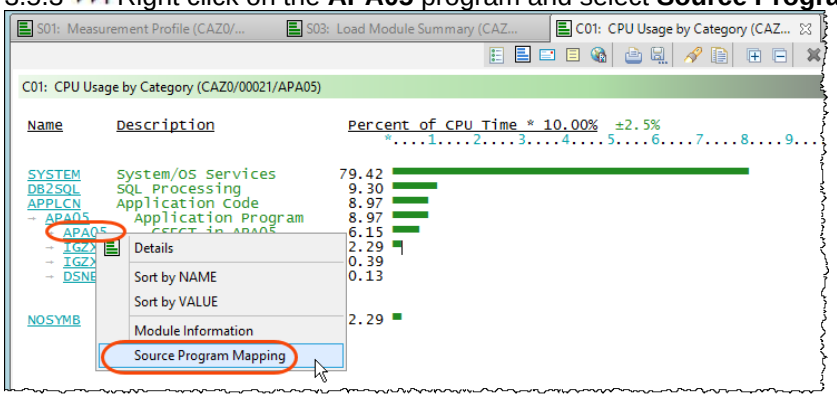
The only Load Module in the *APPLCN* collection is the **APA05** Load Module. In this example **APA05** consumed all of the CPU resource attributed to the *APPLCN* collection.

Since Load Modules can contain multiple programs or CSECTS which are link-edited together, expand the **APA05** program to reveal the individual programs that were observed during the Observation Session.

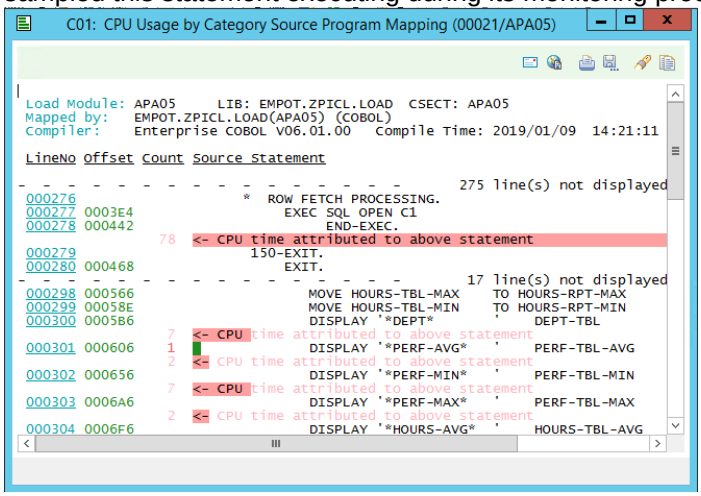
⏭ Click on the **APA05** program to expand the information



3.5.3 Right click on the **APA05** program and select **Source Program Mapping**

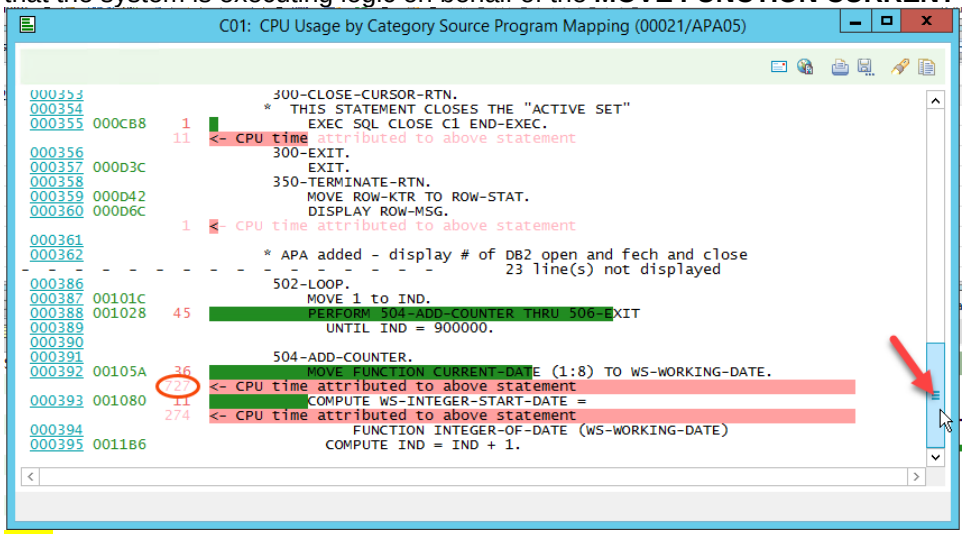


3.5.4 Some of the program source statements are displayed. The **Count** column indicates how many times APA sampled this statement executing during its monitoring process.



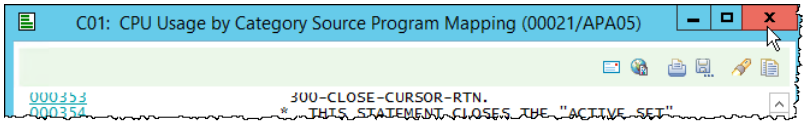
3.5.5 **Scroll up to the end**

APA also indicates when system level activity is taking place on behalf of a source statement. In this example, the Count column carries a high number and the statement under the **MOVE FUNCTION** statement (727) indicates that the system is executing logic on behalf of the **MOVE FUNCTION CURRENT-DATE** statement.



Tip: A Possible performance improving here would define **WS-WORKING DATE** as packed or binary numeric?

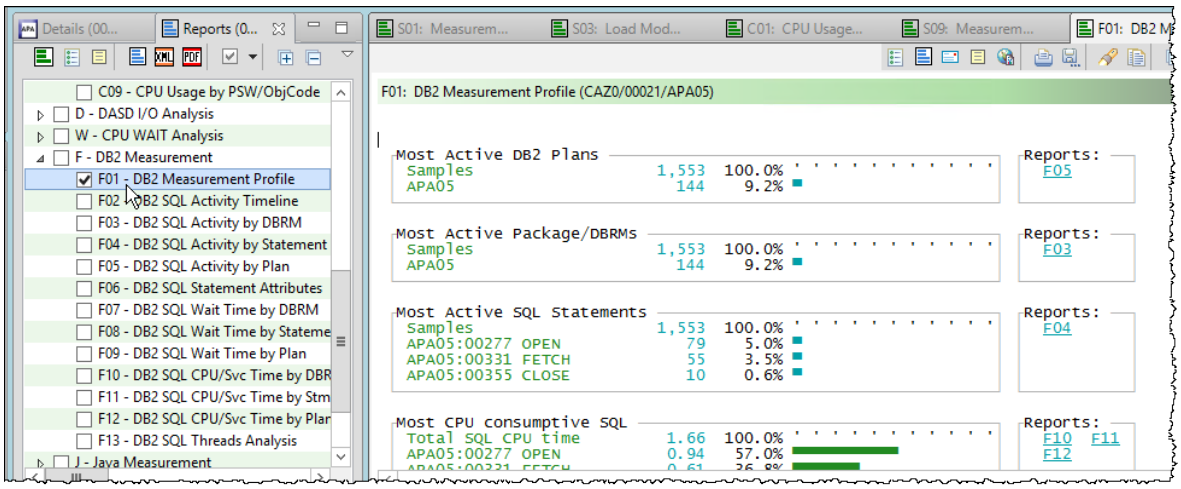
3.5.6 ⏭ Close the panel clicking on the x



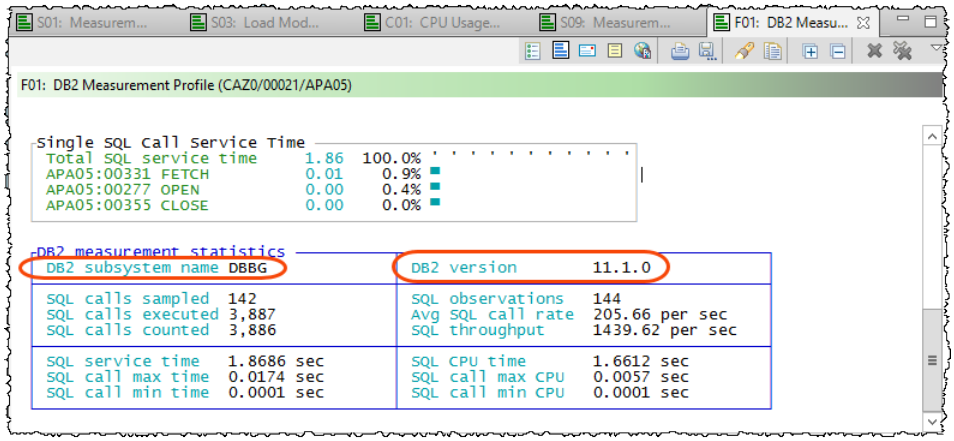
___ 3.6 Analyzing the APA DB2 Measurement Profile report (F01)

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



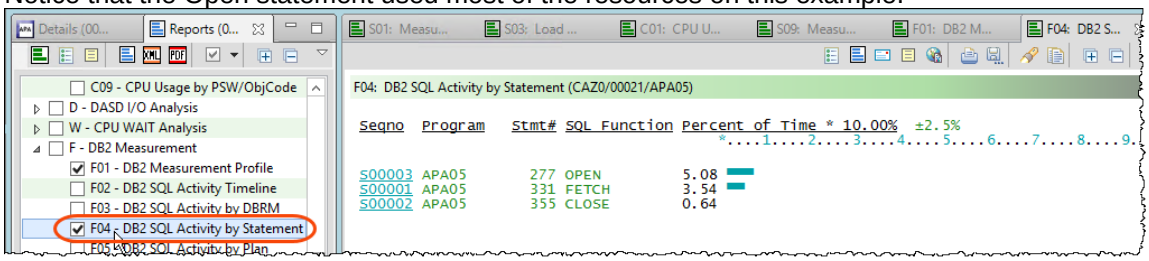
3.6.2 ⏭ **Scroll down** the *DB2 Measurement Profile Reports* view and verify which DB2 level we used, the DB2 subsystem, etc.



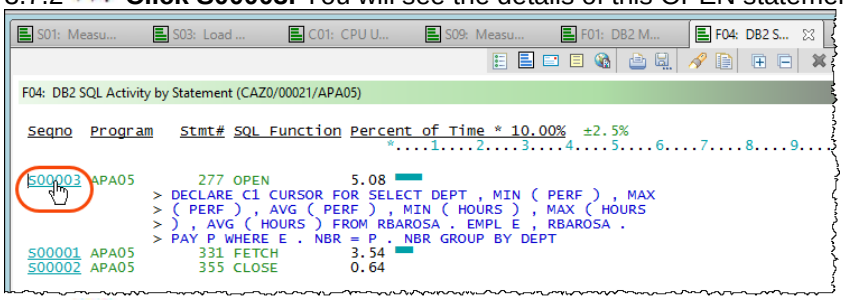
___ 3.7 Analyzing the APA DB2 Activity by Statement report (F04)

Use this report to see how time was consumed by SQL request processing. The percentage of time is reported by each SQL request.

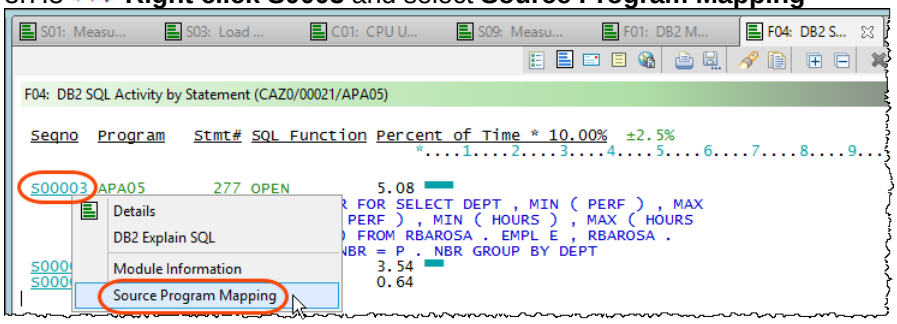
3.7.1 **Double click** on the **F04** report. The *DB2 SQL Activity by Statement* report is displayed. Notice that the Open statement used most of the resources on this example.



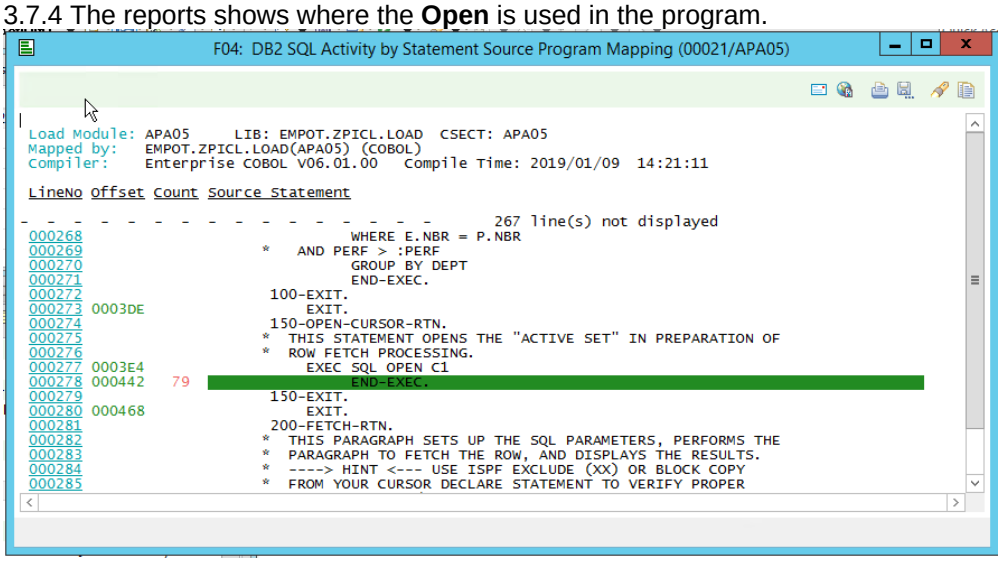
3.7.2 **Click S00003.** You will see the details of this OPEN statement in the program **APA05.**



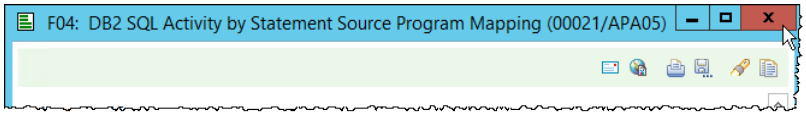
3.7.3 **Right click S0003** and select **Source Program Mapping**



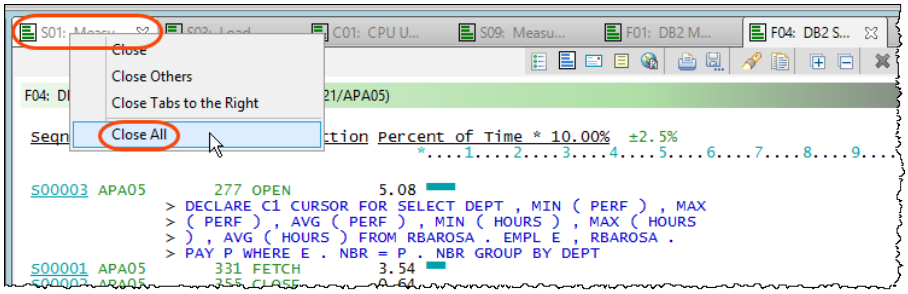
3.7.4 The reports shows where the **Open** is used in the program.



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. .

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

Please Recycle