

CBSA Enacting Scenario 2 – Java Code

Introduction

This “CBSA Enacting Scenario 2- Java” document is provided as part of the Nazare project documentation, it provides a step by step guide to amending an existing Java/Liberty application (CBSA). It assumes that CBSA has already been installed and is up and running (separate installation instructions are provided in the /doc folder in the repo).

What projects do we have?

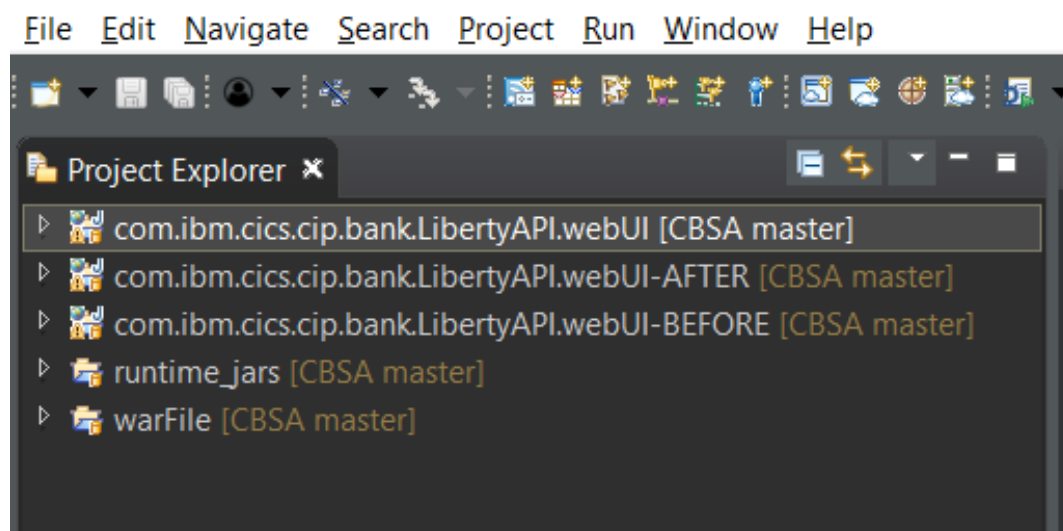
There are three Java projects in the workspace.

- `/com.ibm.cics.cip.bank.LibertyAPI.webUI`
This is what the project we will be working with.
- `/com.ibm.cics.cip.bank.LibertyAPI.webUI-AFTER`
This is a copy of the project after all the changes have been made. You can copy .java files from here into the top project to save time.
- `/com.ibm.cics.cip.bank.LibertyAPI.webUI-BEFORE`
This is a copy of the project before any of the changes have been made. You can copy .java files from here into the top project if you need to back out any changes.

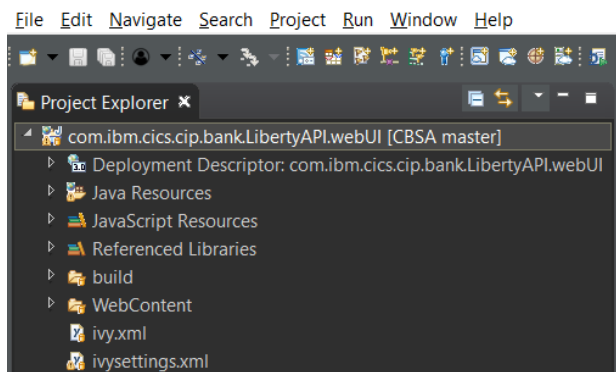
There are two other projects:

- `runtime_jars`
- `warFile`
We will use these later.

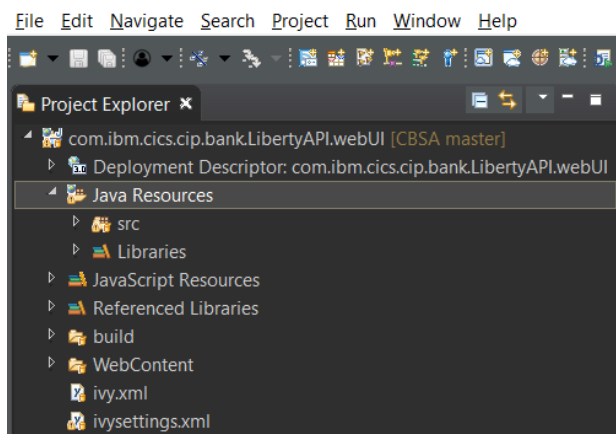
Opening the project



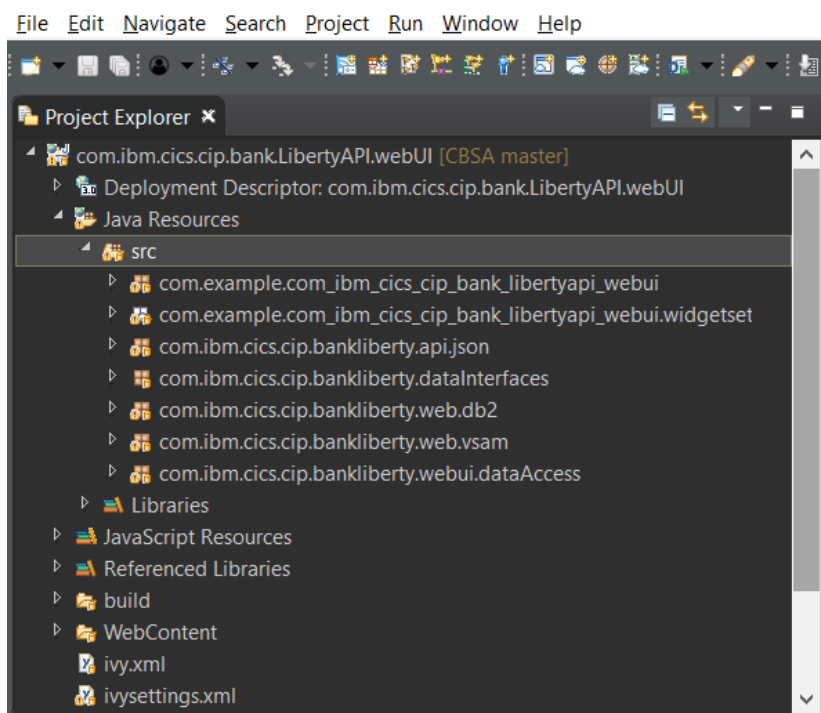
Double click on `com.ibm.cics.cip.bank.LibertyAPI.webUI` to open the project.



There are various folders and directories, but the Java source that we need to change is within the Java Resources folder. Double click to open it.



Within Java Resources there is a `src` folder. Double click on that.



There are a number of folders in here which are the Java packages that make up the application. These are as follows:

- `com.example.com_ibm_cics_cip_bank_libertyapi_webui`

The front-end of the application is an open source browser-based interface based on Vaadin. The code that deals with that is in here.

- `com.example.com_ibm_cics_cip_bank_libertyapi_webui.widgetset`
- `com.ibm.cics.cip.bankliberty.api.json`

The browser interface talks to the business logic using a RESTful interface that passes data in a JSON format. The Java classes that provide that interface are in here.

- `com.ibm.cics.cip.bankliberty.dataInterfaces`

Java classes that act in a similar way to copybooks in other languages can be found here.

- `com.ibm.cics.cip.bankliberty.web.db2`

All access to DB2 is carried out by the classes in this package. The accounts at the bank and the processed transaction records are implemented in DB2 so this is where they are accessed.

- `com.ibm.cics.cip.bankliberty.web.vsam`

All access to VSAM is carried out by the classes in this package. The customer records at the bank are held on a VSAM file so this is where they are accessed.

- `com.ibm.cics.cip.bankliberty.webui.dataAccess`

These are classes used by the browser code to handle data on screen.

What needs changing?

The account numbers in DB2 are changing from 8 digits to 9. Follow these instructions to enact that change before changing the Java code.

[Backing up the data on the ACCOUNT and PROCTRAN Db2 tables, making the Db2 table changes and reloading the affected tables:](#)

Having made all of the BMS, program and copybook changes to make the account number 9 bytes, we now need to alter the Db2 ACCOUNT and PROCTRAN tables to also make these reflect a 9 byte ACCOUNT number too.

The following jobs need to be submitted in the sequence listed below:

A. Offload ACCOUNT table data, change the ACCOUNT table, reload the ACCOUNT table data:

1. Submit Job CBSA.CICSBSA.REORG(TMPACCF). This job off-loads the ACCOUNT table's data to a VSAM dataset (the account numbers are all 8 bytes long at this point) called CBSA.CICSBSA.TMPAFF.
2. Submit CBSA.CICSBSA.REORG(DB2REDA). This job drops/removes all of the ACCOUNT related artefacts e.g. Db2 table, indexes, tablespace, and storage group.
3. Submit CBSA.CICSBSA.REORG(DB2REDB). This job recreates the ACCOUNT table, indexes, tablespace, and storage group, but now with the ACCOUNT NUMBER at 9 bytes in length.
4. Submit CBSA.CICSBSA.REORG(DB2BIND). This rebinds the programs against the newly changed ACCOUNT table.
5. Submit CBSA.CICSBSA.REORG(TMPACCL). This takes the offloaded ACCOUNT data, reads in a record at a time, adds an additional leading '0' to the account number and INSERTs the record onto the newly revised ACCOUNT table.

B. Offload PROCTRAN table data, change the PROCTRAN table, reload the PROCTRAN table data:

6. Submit Job CBSA.CICSBSA.REORG(TMPPROF). This job off-loads the PROCTRAN table's data to a VSAM dataset (with the account numbers all 8 bytes long at this point) called CBSA.CICSBSA.TMPPFF.
7. Submit CBSA.CICSBSA.REORG(DB2REDC). This job drops/removes the PROCTRAN related Db2 table, tablespace, and storage group.
8. Submit CBSA.CICSBSA.REORG(DB2REDD). This job recreates the PROCTRAN table, tablespace, and storage group, with the ACCOUNT NUMBER at 9 bytes in length.
9. Submit CBSA.CICSBSA.REORG(DB2BIND). This rebinds the programs against the newly changed PROCTRAN table.
10. Submit CBSA.CICSBSA.REORG(TMPPROL). This takes the offloaded PROCTRAN data, reads in a record at a time, adds an additional leading '0' to the account number (making the account number 9 bytes long) and INSERTs the record onto the newly revised PROCTRAN table.

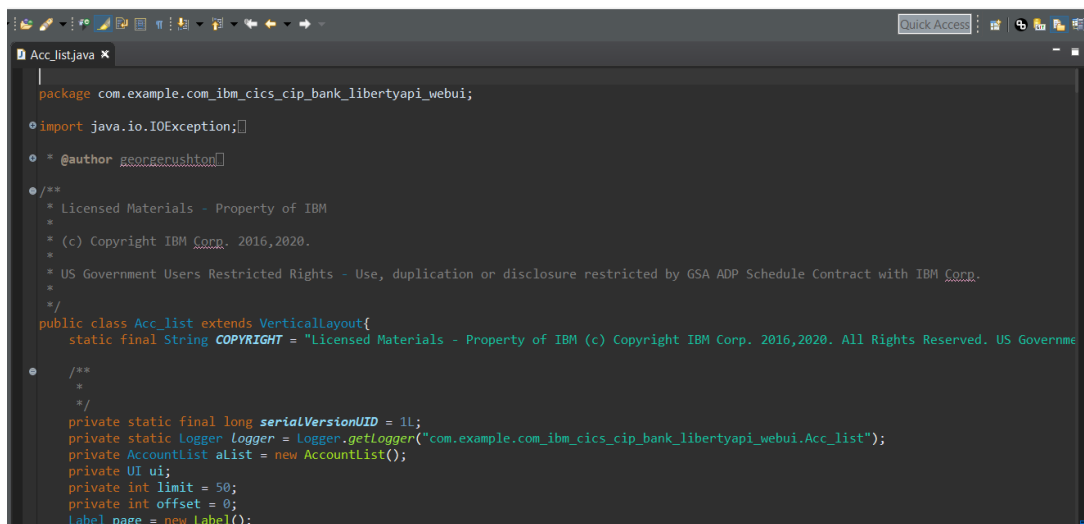
What needs changing in Java?

The good news is that the data in DB2 is treated as a string. The bad news is that for presentation purposes the Java code pads the number with leading zeroes up to 8 digits. We will need to change this wherever appropriate to 9 digits. The Java code also sometimes turns the string into a numeric format that isn't big enough to hold all the digits.

There are four Java classes we need to change.

1. /com/example/com_ibm_cics_cip_bank_libertyapi_webui/Acc_list.java
2. /com/ibm/cics/cip/bankliberty/api/json/AccountsResource.java
3. /com/ibm/cics/cip/bankliberty/web/db2/Account.java
4. /com/ibm/cics/cip/bankliberty/web/db2/ProcessedTransaction.java

1. Acc_list.java



```
package com.example.com_ibm_cics_cip_bank_libertyapi_webui;

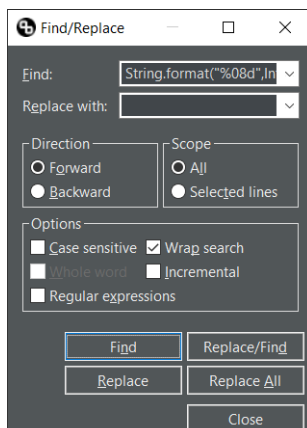
import java.io.IOException;

/*
 * @author geongerushton
 */
/*
 * Licensed Materials - Property of IBM
 *
 * (c) Copyright IBM Corp. 2016,2020.
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
public class Acc_list extends VerticallyLayout {
    static final String COPYRIGHT = "Licensed Materials - Property of IBM (c) Copyright IBM Corp. 2016,2020. All Rights Reserved. US Governme

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private static Logger logger = Logger.getLogger("com.example.com_ibm_cics_cip_bank_libertyapi_webui.Acc_list");
    private AccountList alist = new AccountList();
    private UI ui;
    private int limit = 50;
    private int offset = 0;
    Label page = new Label();
}
```

Open the source. There are four lines in this program that need changing. We could search for them based on line number but instead we are going to look for the string that needs changing using Find to be sure we don't miss any.

Press Ctrl+F to open up the Find dialogue.



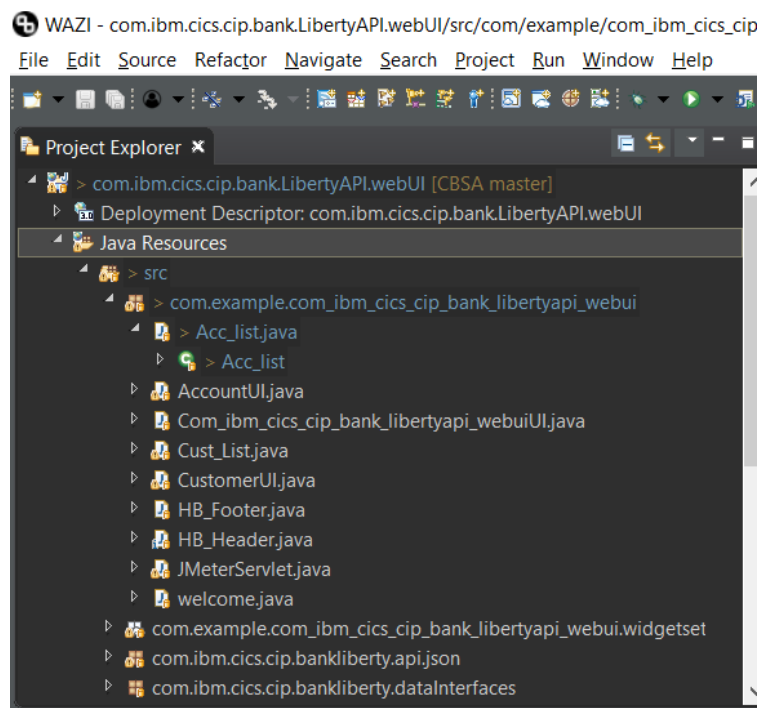
```
String.format("%08d", Integer.valueOf(accNumT.getValue())));
```

This is the string we are looking for. We need to change it to the following:

```
String.format("%09d", Long.valueOf(accNumT.getValue())));
```

We need to do this four times. We can do this all at once using the Find/Replace interface and Replace/All, or we can find and then edit the source manually.

Once this is done, save the source. Note that the source file and containing folders change colour to indicate that a change has been made but not committed to the source repository.



2. AccountsResource.java

CICS Bank Sample Application is configured to behave differently if certain special values are entered. One of these values is a customer or account number which consists entirely of the number 9. This value means “return me the very last record of this time”. As we are changing account numbers from 8 digits to 9 digits, we have to update the code to look for this new value.

Use Ctrl+F to bring up the search dialogue, and look for 999999999L (that is EIGHT nines and then an uppercase L). You will find it TWICE, the first being “999999999L” for the customer number, and the second being the one we need to change.

```
if(db2Account == null && id_safe == 999999999L)
```

needs to be changed to

```
if(db2Account == null && id_safe == 9999999999L)
```

In addition, the account number is treated as an int in places in the code. This will no longer work if the account number is nine digits long.

a. `getAccountInternal(Long accountNumber)`

This method is used to return a single account. There is some code within it that treats the account number as an integer, which will not work with the new account number as it will not be long enough.

```
db2Account = db2Account.getAccount(accountNumber.intValue(), sortCode);
```

must be changed to

```
db2Account = db2Account.getAccount(accountNumber.longValue(), sortCode);
```

This will flag as an error but you will fix this later.

b. `transferLocalInternal(@PathParam("id") String
accountNumber, TransferLocalJSON transferLocal)`

This method is used to move money between accounts at the same bank. The code performs validation to make sure that the two accounts are not the same. This is currently assuming that the account numbers are integers so this must be changed.

```
if(new Integer(accountNumber).intValue() == new  
Integer(transferLocal.getTargetAccount().intValue()))
```

must be changed to

```
if(new Long(accountNumber).longValue() == new  
Long(transferLocal.getTargetAccount().longValue()))
```

c. `deleteAccountInternal(Long)`

This method is used to delete a single account. There is some code within it that treats the account number as an integer, which will not work with the new account number as it will not be long enough.

```
db2Account = db2Account.deleteAccount(accountNumber.intValue(), sortCode.intValue());
```

must be changed to

```
db2Account = db2Account.deleteAccount(accountNumber.intValue(), sortCode.longValue());
```

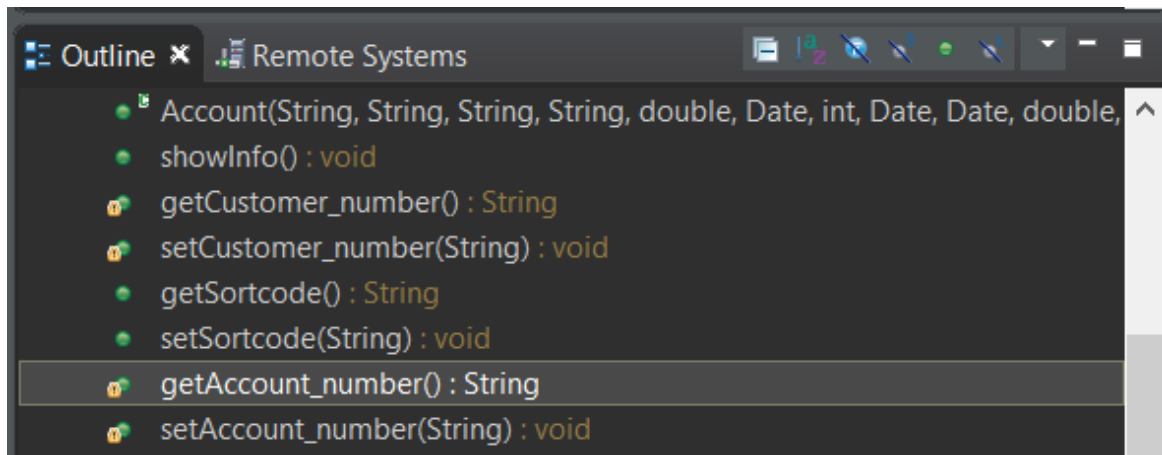
This will flag as an error but you will fix this later.

3. `Account.java`

This Java class controls access to the ACCOUNT table in DB2.

- a. There are two methods that control access to the account number field, via “getter” and “setter” classes. These are the ones that we need to change, as they contain rudimentary padding routines in case the digit is less than 8 digits long. We must change these so that they are padding up to NINE digits.

In the Outline view for this Java class, look for the methods
`getAccount_number()` and `setAccount_number(String)`.



Clicking on either of these will position the editor at the correct position in the file.

The “get” method contains

```
if(this.account_number.length()<8)
```

which must be changed to

```
if(this.account_number.length()<9)
```

and

```
for (int i=8;this.account_number.length()<8;i--)
```

which must be changed to

```
for (int i=9;this.account_number.length()<9;i--)
```

the “set” method contains

```
if(account_number.length()<8)
```

which must be changed to

```
if(account_number.length()<9)
```

and

```
for (int i=8;account_number.length()<8;i--)
```

which must be changed to

```
for (int i=9;account_number.length()<9;i--)
```

- b. The method `getAccount(int account,int sortCode)` must be changed to `getAccount(long account,int sortCode)`.

This method gets a single account and takes two parameters, the account number and the sortcode. The account number being 9 digits now must be changed to be a long.

CICS Bank Sample Application is configured to behave differently if certain special values are entered. One of these values is a customer or account number which consists entirely of the number 9. This value means “return me the very last record of this time”. As we are changing account numbers from 8 digits to 9 digits, we have to update the code to look for this new value.

```
if(account == 99999999)
```

must be changed to

```
if(account == 999999999L)
```

c. **The method** deleteAccount(int,int)

This is used to delete one account.

```
StringBuffer myStringBuffer = new StringBuffer(new
Integer(account).toString());
for(int z = myStringBuffer.length(); z < 8;z++)
{
    myStringBuffer = myStringBuffer.insert(0, "0");
}
String accountNumberString = myStringBuffer.toString();
```

becomes

```
StringBuffer myStringBuffer = new StringBuffer(new
Long(account).toString());
for(int z = myStringBuffer.length(); z < 9;z++)
{
    myStringBuffer = myStringBuffer.insert(0, "0");
}
String accountNumberString = myStringBuffer.toString();
```

d. **The method** updateAccount(AccountJSON account)

When we created the object db2Account we must now give the account number as a long. Change

```
Account db2Account = this.getAccount(new Integer(account.getId()).intValue(), new
Integer(sortcode).intValue());
```

To

```
Account db2Account = this.getAccount(new Long(account.getId()).longValue(), new
Integer(sortcode).intValue());
```

Then

```
StringBuffer myStringBuffer = new
StringBuffer(accountNumber.toString());

for(int z = myStringBuffer.length(); z < 8;z++)
{
    myStringBuffer = myStringBuffer.insert(0, "0");
}
```

Becomes

```
StringBuffer myStringBuffer = new
StringBuffer(accountNumber.toString());

for(int z = myStringBuffer.length(); z < 9;z++)
{
    myStringBuffer = myStringBuffer.insert(0, "0");
}
```

e. **The method** `debitCredit(BigDecimal apiAmount)`

Although not directly used in the Web interface, this method is used to debit or credit an amount of money from a customer's account.

```
Account temp = this.getAccount(new Integer(this.getAccount_number()).intValue(),
new Integer(this.getSortcode()).intValue());
```

Must become

```
Account temp = this.getAccount(new Long(this.getAccount_number()).longValue(), new
Integer(this.getSortcode()).intValue());
```

4. ProcessedTransaction

There are eight instances of code similar to below, but not always with the same variable name. In each case the “%08d” must be changed to “%09d”.

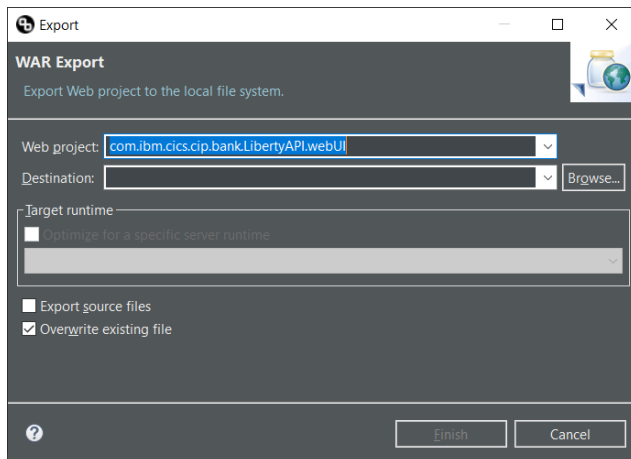
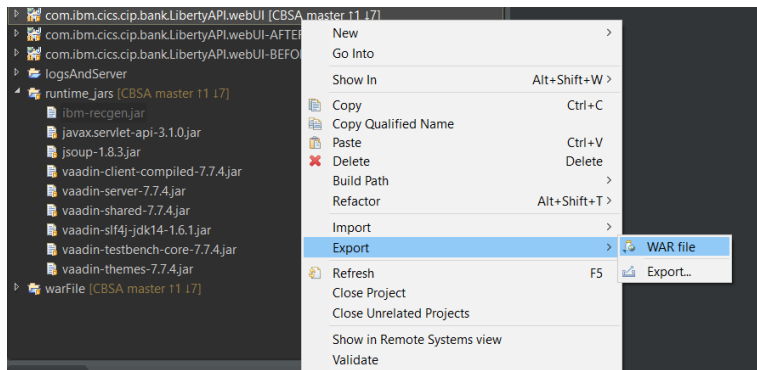
```
String.format("%08d", Integer.parseInt(accountNumber))
```

to

```
String.format("%09d", Integer.parseInt(accountNumber))
```

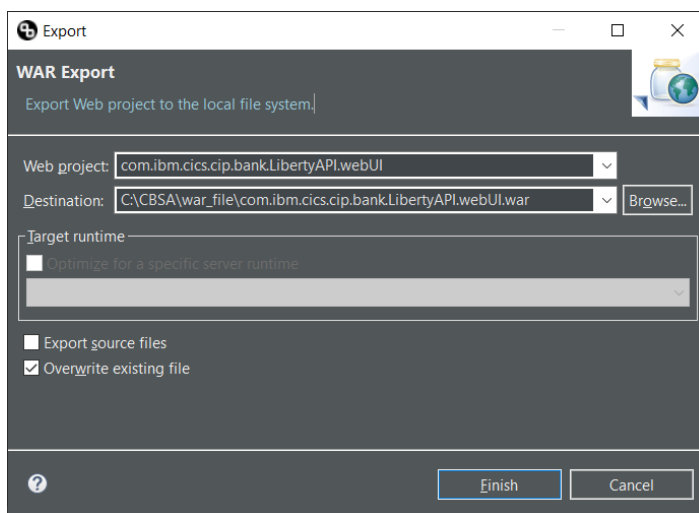
Exporting the code as a “war” file

Right-click the project in the Project Explorer view and select **Export** and then **WAR file**, as shown here:



The name of the Web project will be highlighted but you will need to select a destination.

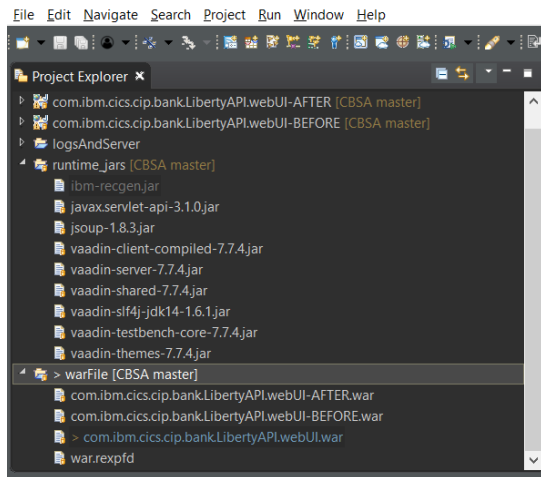
Select the `war_file` folder in the current environment.



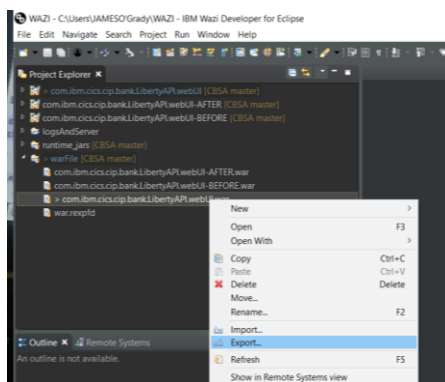
And then click **FINISH**

Exporting the WAR file to z/OS

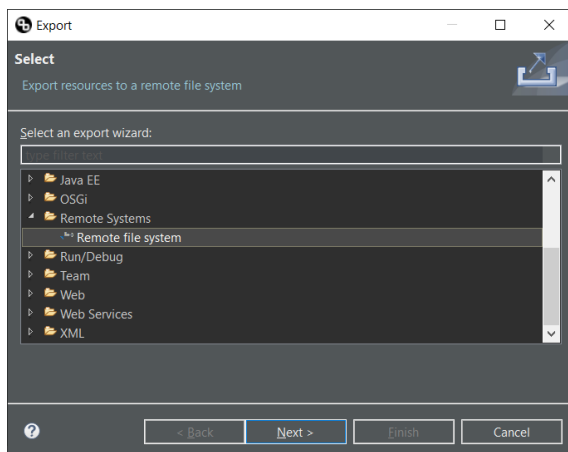
Expand the `war_file` project in WAZI Developer for Eclipse.

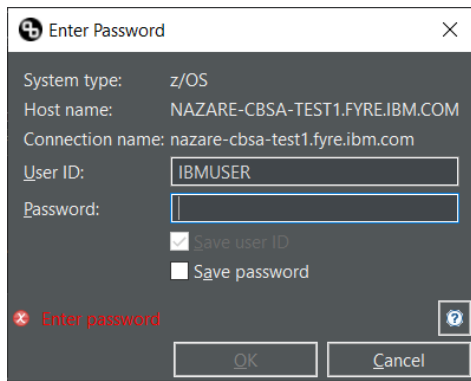


Right-click on `com.ibm.cics.cip.bank.LibertyAPI.webUI.war` and select **Export**.

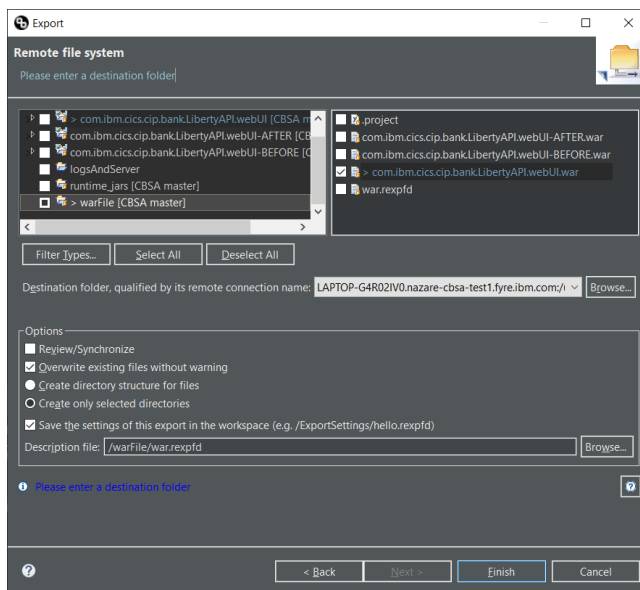


Select **Remote Systems** and then **Remote File System**:

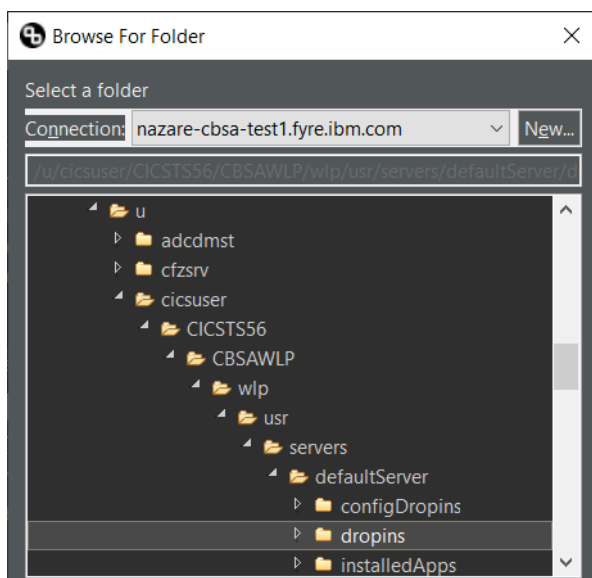




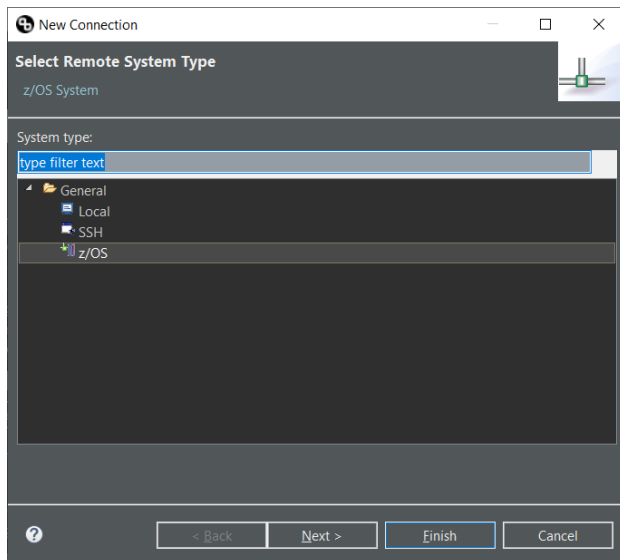
You may be prompted for your userid and password.



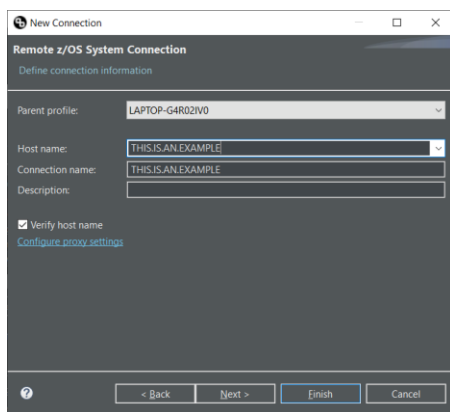
Click on **BROWSE**.



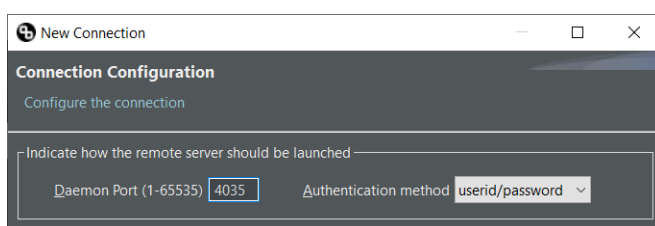
If you do not already have a connection to your z/OS system, click **NEW**.



Select **z/OS** as the System Type.



Complete the host name you were supplied with and then click **NEXT**.



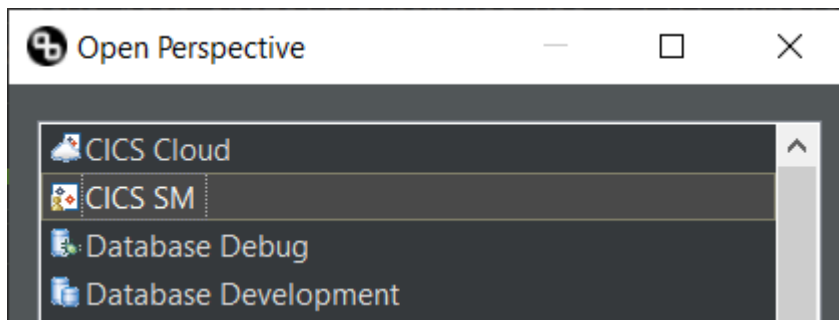
And then click **FINISH**. You will be prompted for authentication.

You need to deploy the WAR file to the dropins directory that your JVM Server has been configured with.

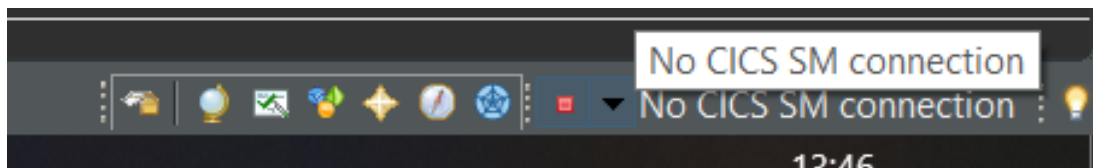
/u/cicsuser/CICSTS56/CBSAWLP/wlp/usr/servers/defaultServer/dropins

Refreshing the code

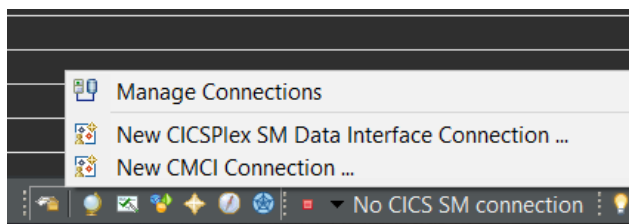
In Wazi Developer for Eclipse, change to the CICS SM perspective. You do this by selecting Window → Perspective → Open Perspective → Other and then finding CICS SM.



At the bottom right of the screen you will see a red square. Hovering over this will tell you that you have no CICS SM Connection.



Click on the arrow to the right of the square and select New CMCI Connection.



Complete with the hostname and the CMCI port. The port is NOT the same as the JVM server.

Add CMCI Connection

Specify the host, port, and any additional details for the new connection

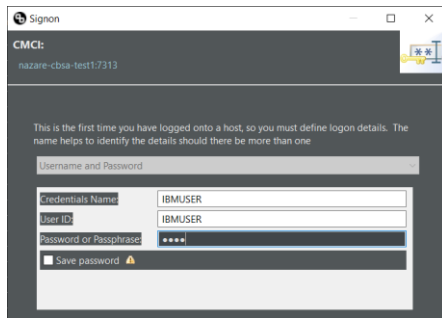
Name: nazare-cbsa-test1:1490

Location

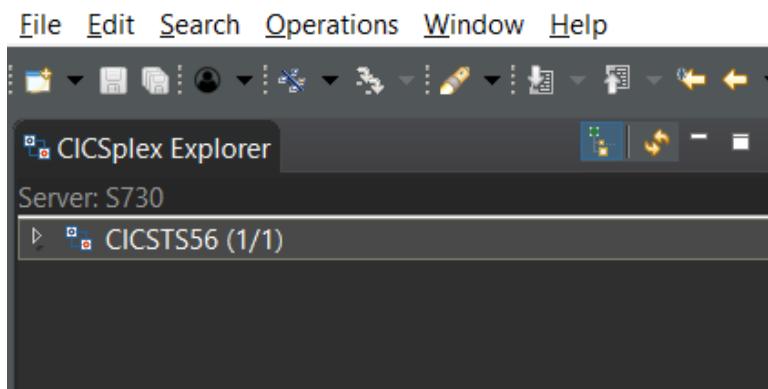
Host name: nazare-cbsa-test1.fyre.ibm.com

Port number: 1490 ☐ Secure connection (TLS/SSL)

Click “Save and Connect”. You will be prompted for a userid and password.



On the left hand side you will see that you are connected to the CICS region.

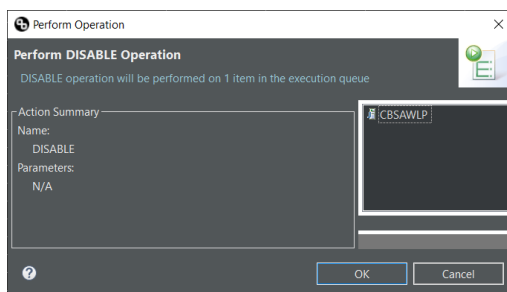


Select Operations → Java → JVM Servers.

View for Eclipse

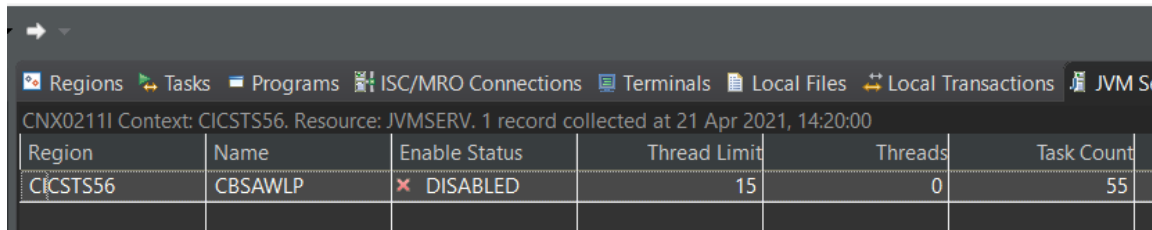
Region	Name	Enable Status	Thread Limit	Threads	Task Count	...
CICSTS56	CBSAWLP	✓ ENABLED	15	6	55	839

Right-click the JVM server in the JVM Servers view and select Disable → Phase Out.



Click OK to confirm.

The JVM server will change to a DISABLING status. Wait until it has disabled, pressing F5 or the refresh button until it has.

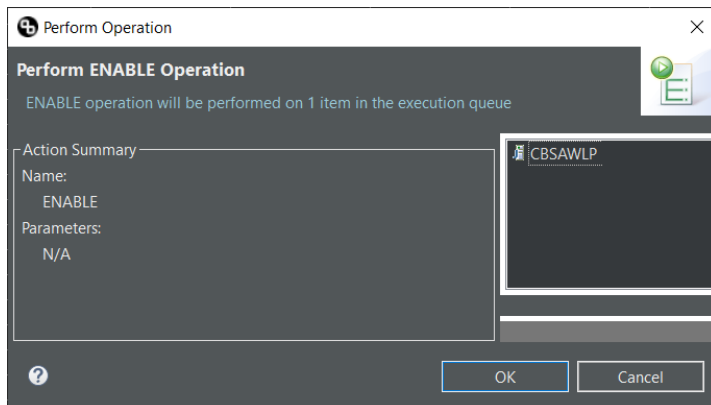


Regions Tasks Programs ISC/MRO Connections Terminals Local Files Local Transactions JVM S

CNX0211I Context: CICSTS56. Resource: JVMSESV. 1 record collected at 21 Apr 2021, 14:20:00

Region	Name	Enable Status	Thread Limit	Threads	Task Count
CICSTS56	CBSAWLP	✗ DISABLED	15	0	55

The JVM server is now disabled. Right-click and select Enable.



Confirm the operation and wait for the JVM server to enable and for all applications to start. This may take a few minutes.

Checking the results

Access the Liberty UI using a web browser

<http://nazare-cbsa-test1.fyre.ibm.com:19080/com.ibm.cics.cip.bank.LibertyAPI.webUI/>

Click on List/Search Accounts

You can see that the account number is now 9 digits long.

Account Number	Customer Number
<input type="text"/>	<input type="text"/>
Account No.	Customer No.
000000001	0000000001
000000002	0000000001
000000003	0000000001