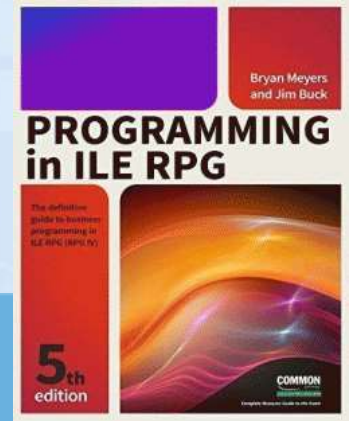
RPG, MVC Architecture and Modernization

Coding RPG for the 21ST Century

Jim Buck
Phone 262-705-2832
jbuck@impowertechnologies.com
Twitter - @jbuck_imPower
www.impowertechnologies.com



imPOWER TECHNOLOGIES
Powered by Jim Buck

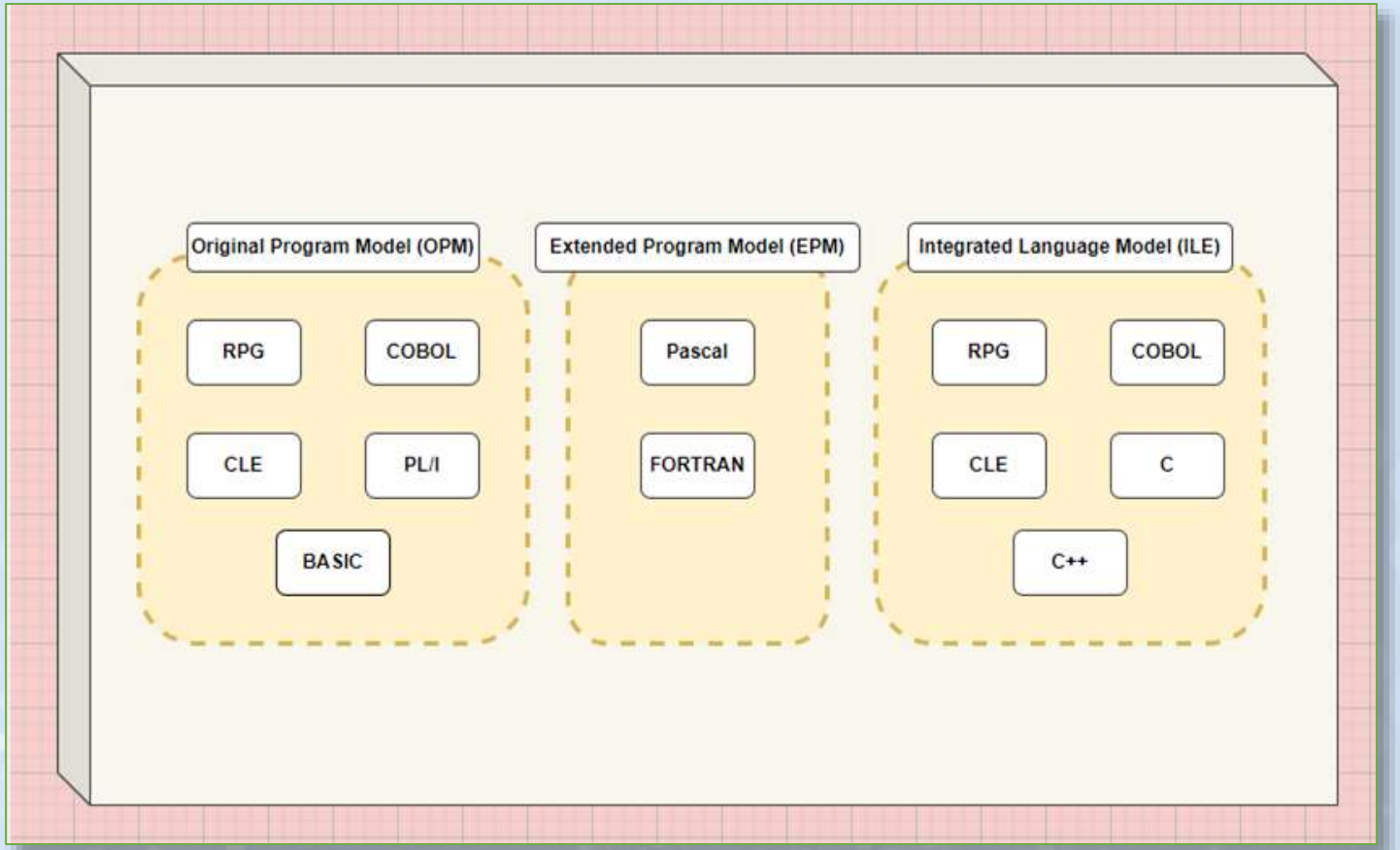
How can imPower Technologies help your company?

- IBM i Education
  - Online IBM i Classes: Unique offering
  - IBM i Concepts
  - Programming in ILE RPG
  - RDi / Modular Programming – **Updated Workshop**
  - SQL Queries Workshop – Birgitta Hauser **New Workshop**
- Onsite IBM i Classes: **Now Available**
  - Two-day hands-on lecture and exercises
  - Optional Third day - Let's design and code a new application
- Modernization: Getting started
  - Helping a company getting started down the modernization road
  - The thought process of modern development
  - Learn to use these new tools and concepts

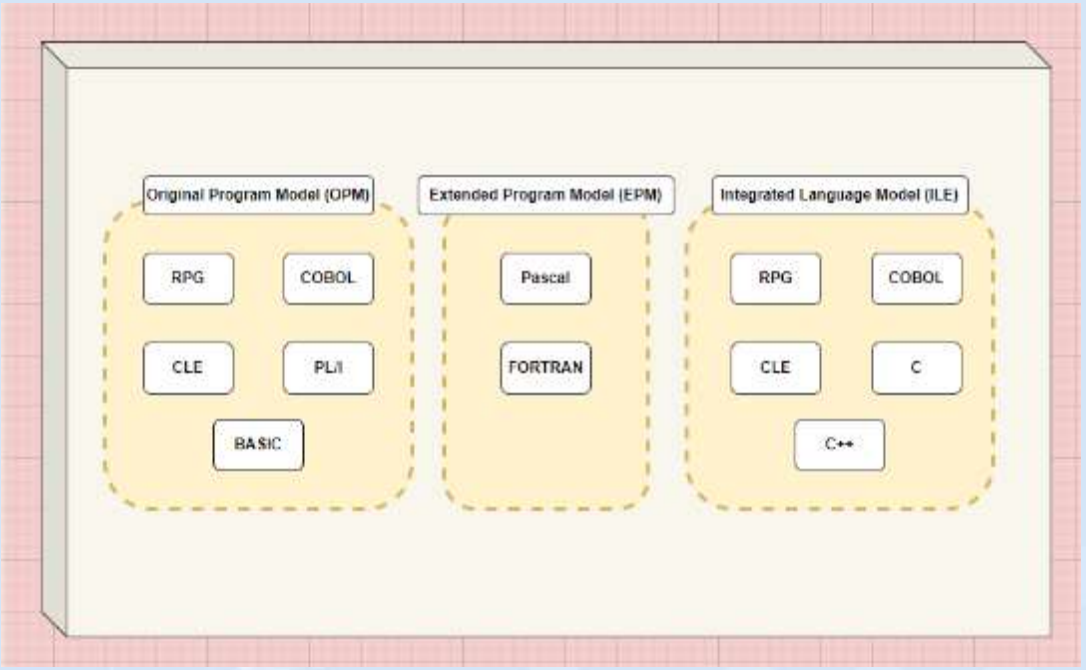
Session Objectives

- Relationship of OPM, EPM & ILE to IBM i OS
- Converting RPG III to RPG IV (ILE RPG)
- Additional Steps in the Modernization Process
- ILE vs OPM – 80's Thinking
- ILE ~~vs OPM~~ – A New way of thinking
- The Next Generation of RPG programs
- MVC Architecture!
- A Modern IBM i Application Overview
- Activation Groups
- Introduction to Binding Directories
- Introduction to Binder Source
- Create the Application

Relationship of OPM, EPM & ILE to IBM i OS



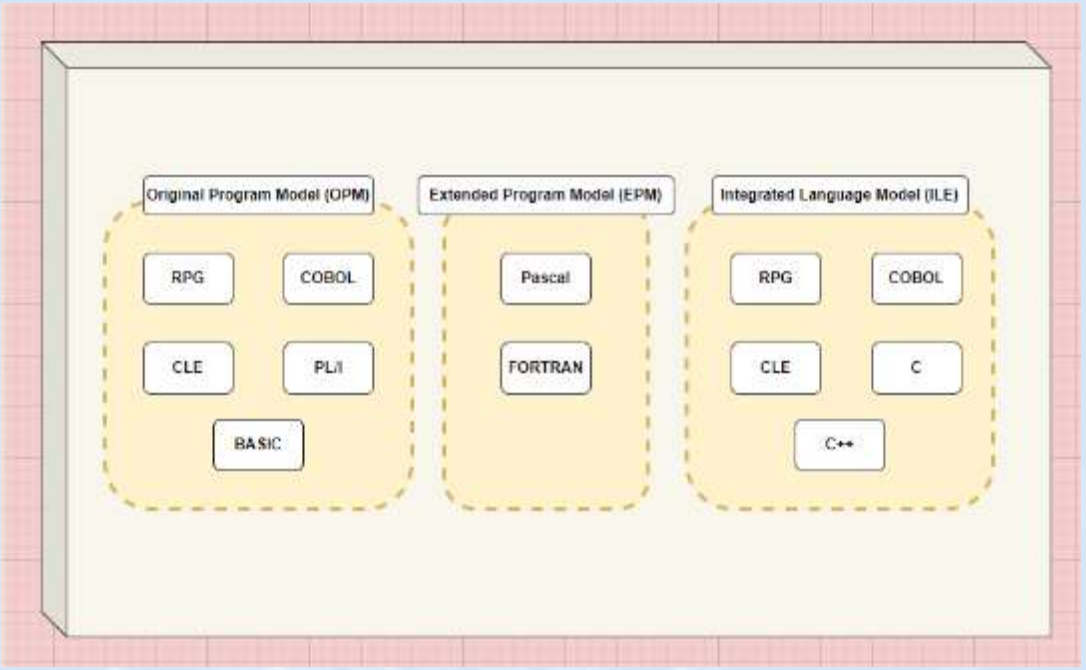
Original Program Model (OPM)



Introduced in OS/400 Version 1

- OPM compiler produced the program object and additional code to handle any special processing required. Example handling input parameters
- Special processing defined the ***Entry Point*** for the program.
- All Calls to a program were Dynamic (overhead)
- Dynamic calls can require significant resources. This led to large programs which reduced these calls
- Functions often provided by the compiler are included in the OS. This allows one language to call programs written in another
- As of version 6.1 BASIC compiler isn't available

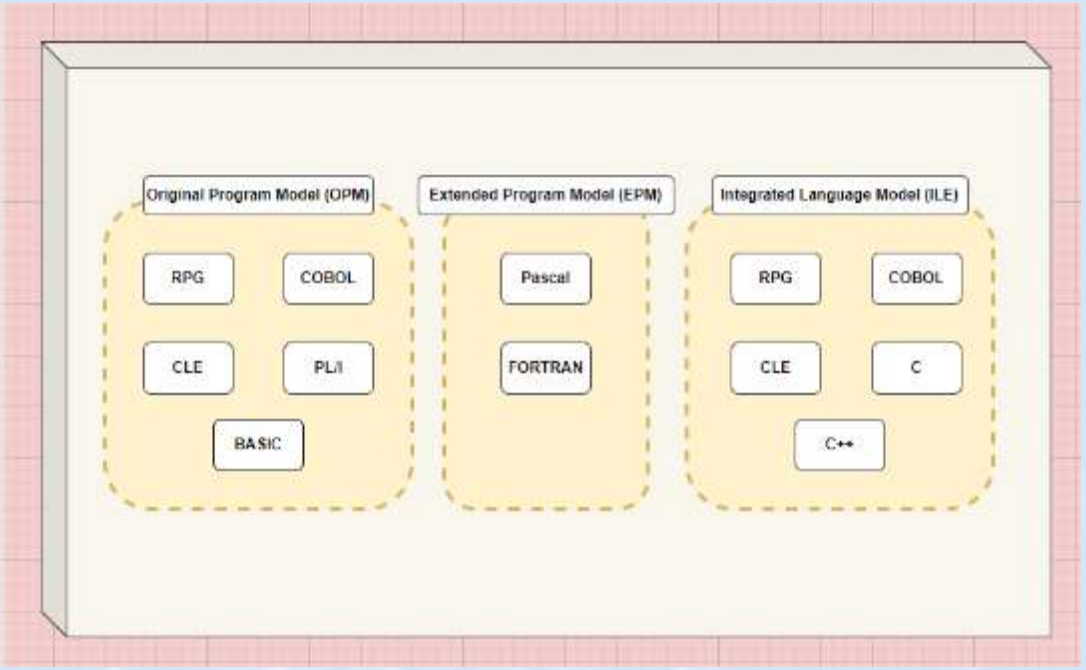
Enhanced Program Model (EPM)



Introduced in OS/400 Version 1, release 2

- An enhancement to OPM which allowed the definition of procedure calls.
- Interim solution that allowed calls to languages like C, FORTRAN and Pascal and return data from these procedures
- The system no longer provides EPM compilers

Integrated Language Environment



Introduced in OS/400 Version 2, release 3 (1994)

- Provided the same type procedure integration that EPM did but more robust and better performance
- The ILE compiler doesn't create an executable program object but a module object (CRTRPGMOD)
- The Executable is created in a separate step using the CRTPGM command.
- CRTBNDPGM combines the CRTRPGMOD and CRTPGM commands
- These program modules; written in COBOL, RPG, C, C++ and CL can be assembled into an executable program object

Converting RPG III to RPG IV (ILE RPG)

Convert RPG Source (CVTRPGSRC) command

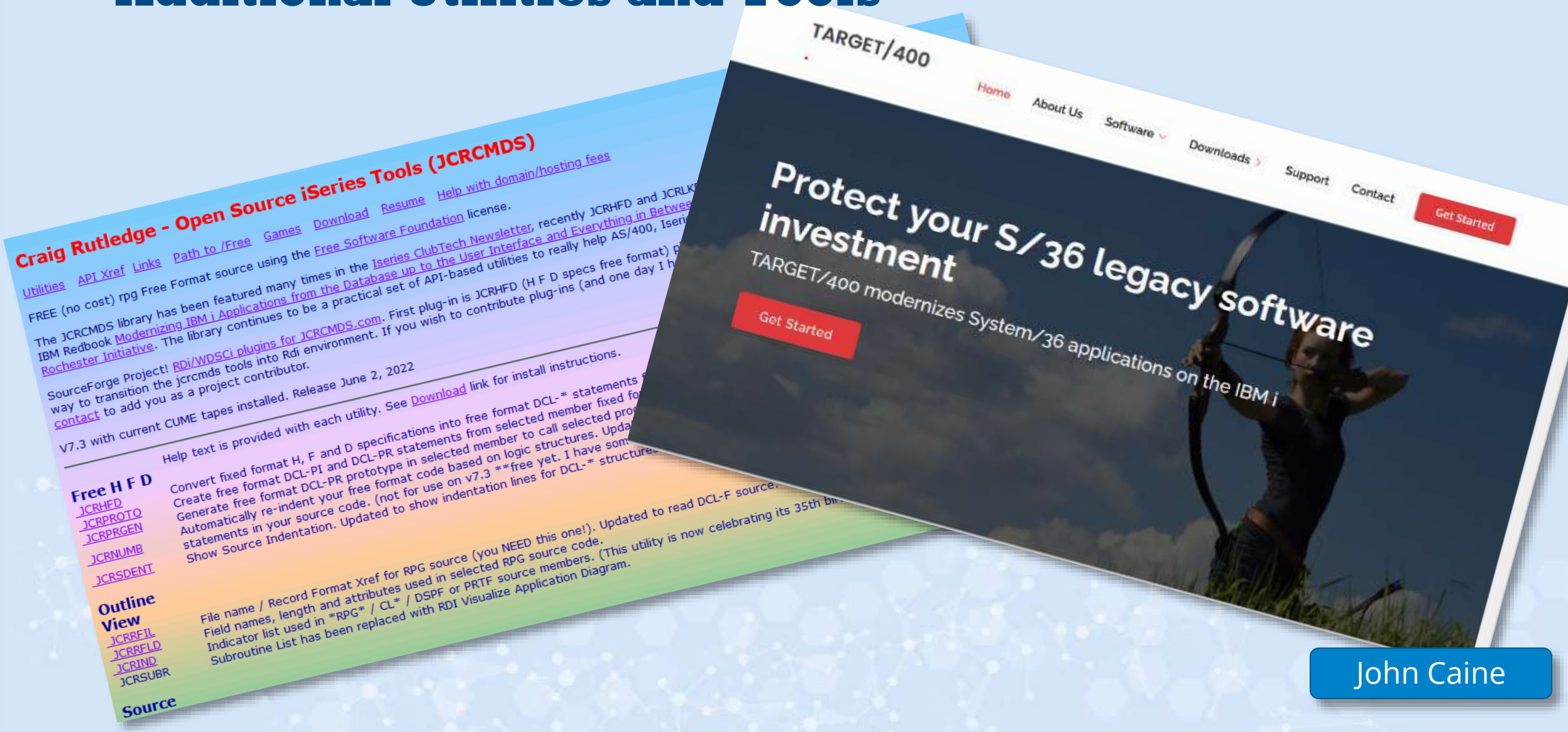
- Can convert single member, source file or member with a common source name
- Converts line-by-line and updates a log file of the results
- Assumes that your code will compile
- Will mark RPG III code that isn't supported and this is usually a small amount

Converting RPG III to RPG IV (ILE RPG)

What the conversion tool won't do!

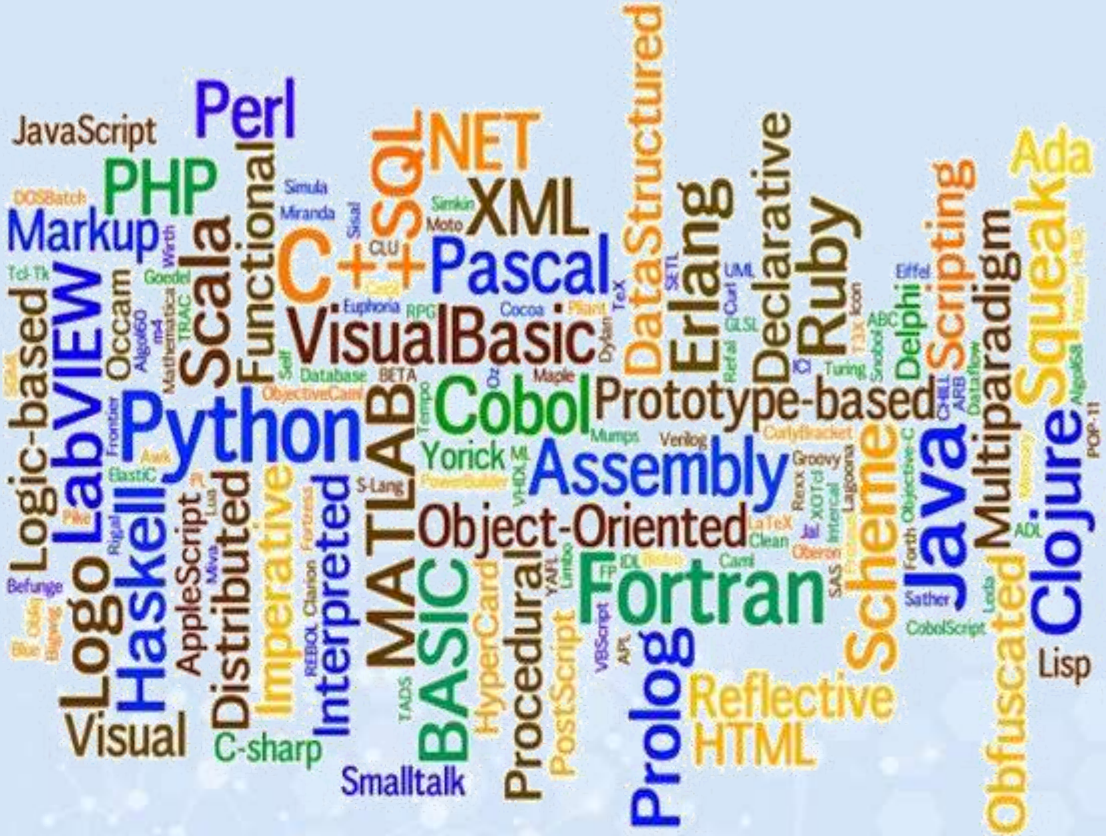
- Will not convert source back to RPG III or RPG/400
- Doesn't support conversion at compile time
- Doesn't support converting RPG II code, there is a utility for converting RPG II to RPG III
- The conversion utility **DOES NOT** re-engineer the source code, except where needed to convert to RPGILE
- Doesn't create files; the Log and output file must already exist

Additional Utilities and Tools



John Caine

Additional Steps in the Modernization Process



Additional Modernization steps:

- Convert to Free format RPG
- Replace Subroutines with Procedures
- Consolidate date conversions
- Create service programs for code used by multiple processes
- Develop new input and display applications
- Create service programs to be used as Web Services

Checkout third party software solutions to help with this process!

ILE vs OPM

- Integrated Language Environment (ILE)
- Main Benefit of ILE – Many small fast procedures
- ILE procedure calls optimized to be VERY FAST!
- Easier to maintain, when using smaller code components
- More Exception handling options
- Easier Access to APIs, including the entire C runtime library
- Results in better performance! ☺

ILE vs OPM – 80's Thinking

In the old days large programs used to do complicated applications

Inefficient

- Programs hard to understand
- Difficult to test, all or nothing
- Programs difficult to maintain! ☹
  - Small changes can cause much effort (and heartache) to implement
  - Sometimes won't support current technologies

Benefits?

- All your code in one place
- Torment young developers
- What else ????

ILE vs OPM – A New way of thinking

Today's Applications – Should be comprised of small procedures

- Program flow is easier to see
- Small procedure is easier to understand
- Easier to test a procedure at a time
- Easier to share across multi-applications
- Easier to divide up the application among your developers
- Good procedure names can make the code self-documenting
- Easily step over procedures when debugging
- Easier to define and debug where problems are occurring

More Control

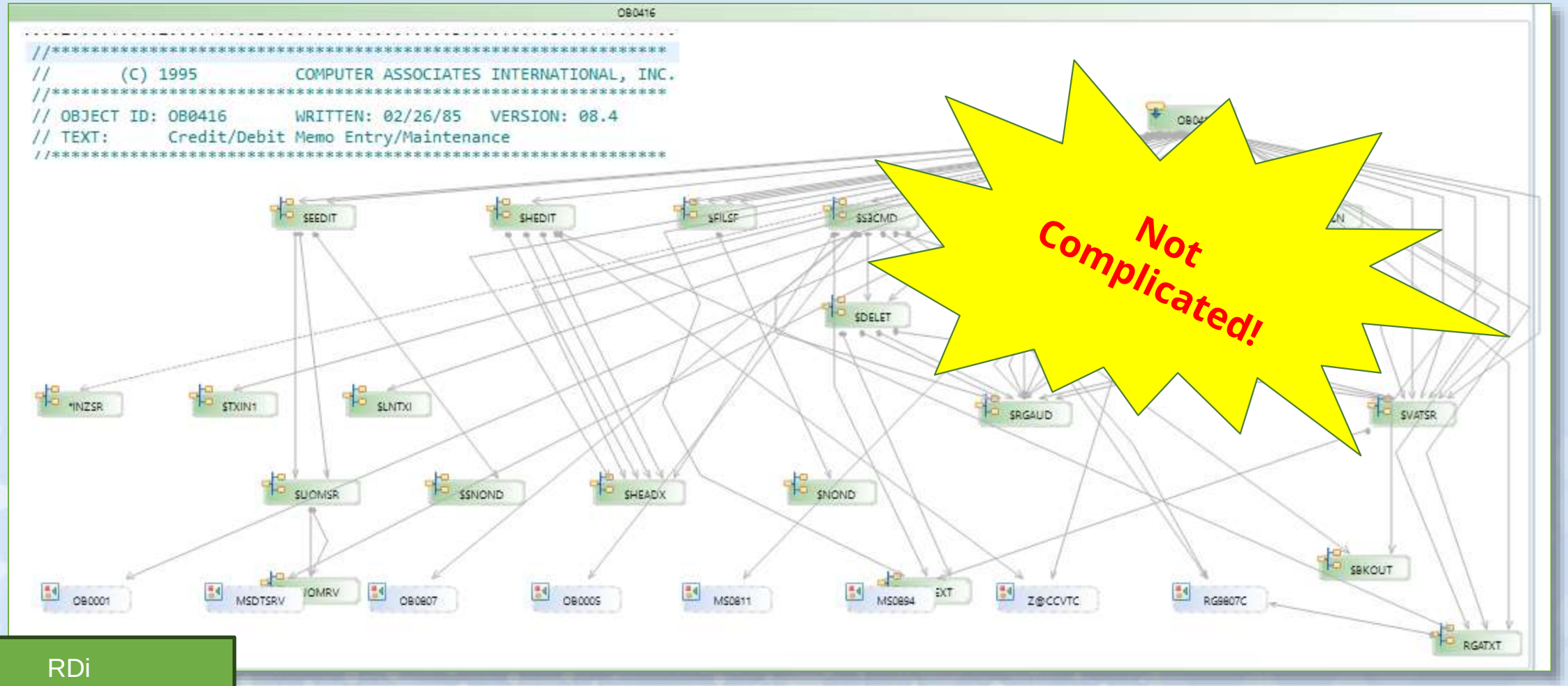
- Activation Groups
- Exception handling

What NOT to consider!

Is there an optimum????

- Size of a procedure
- Number of procedures per module
- Depth of code nesting level
- Number of Binding directories
- Service programs per binding directory
- Modules per Service Program

Traditional Monolith Program



RDi
Visual Application
Diagram

The Next Generation of RPG programs

The future of RPG

- RPG isn't going anywhere
- Need to expend resources to update older code. The tools are available!
- Develop new code that will be usable regardless of the changes made to the VIEW
- Write RPG programs that can be used for Web Services

MVC Architecture!

Break up your code into three components

- Introduced in 1979 by Trygve Mikkjel Heyerdahl Reenskaug
- His idea was to break-up complicated applications into smaller more manageable parts
- 80’s and early 90’s popular with desktop application
- Today it is widely used in Web Application Frameworks
- Some frameworks, ASP.NET, Angular, Ruby on Rails

MVC Architecture!

Break up your code into three components

- **Model** – responsible for the data-logic behind the application
- **View** - How the user sees and interacts with the application
- **Controller** - Makes decisions for the application based on the actions sent from the View and data from the Model. The *"Decision maker"* of the application

A Modern IBM i Application Overview

PROG175SQL.sqlrpgle
Program drives the 5250 screens assorted Subfiles and handles the interactions needed to maintain part orders for the company

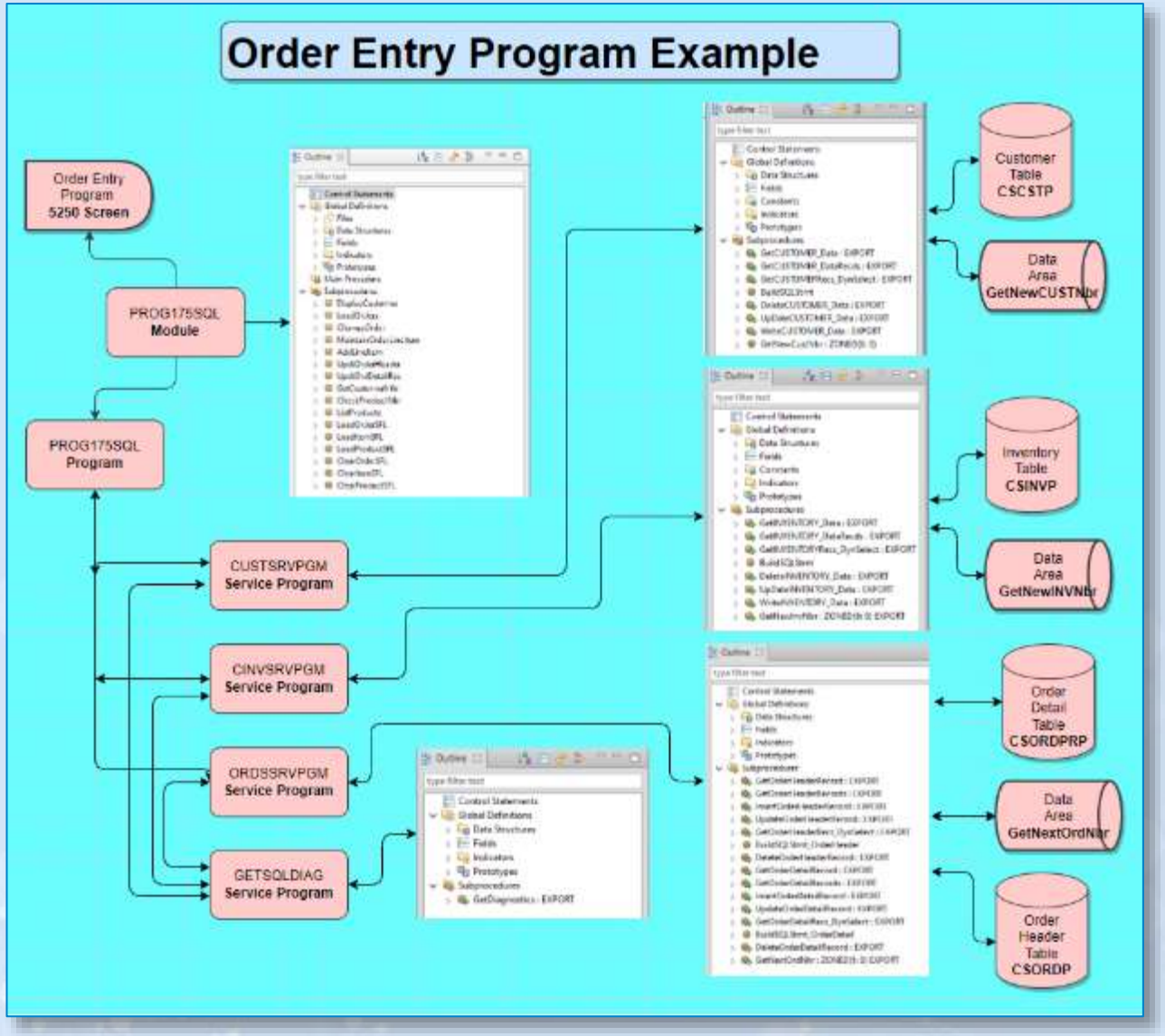
PROG175D Screen
DDS needed to Display, Delete and maintain orders

CUSTSRVPGM.sqlrpgle
Handles all of the Customer table database work

CINVSRVPGM.sqlrpgle
Handles all of the Inventory table database work

ORDSSRVPGM.sqlrpgle
Handles all of the Order Header/Detail table database work

GETSQLDIAG.sqlrpgle
Service program called to check results of any SQL Statement

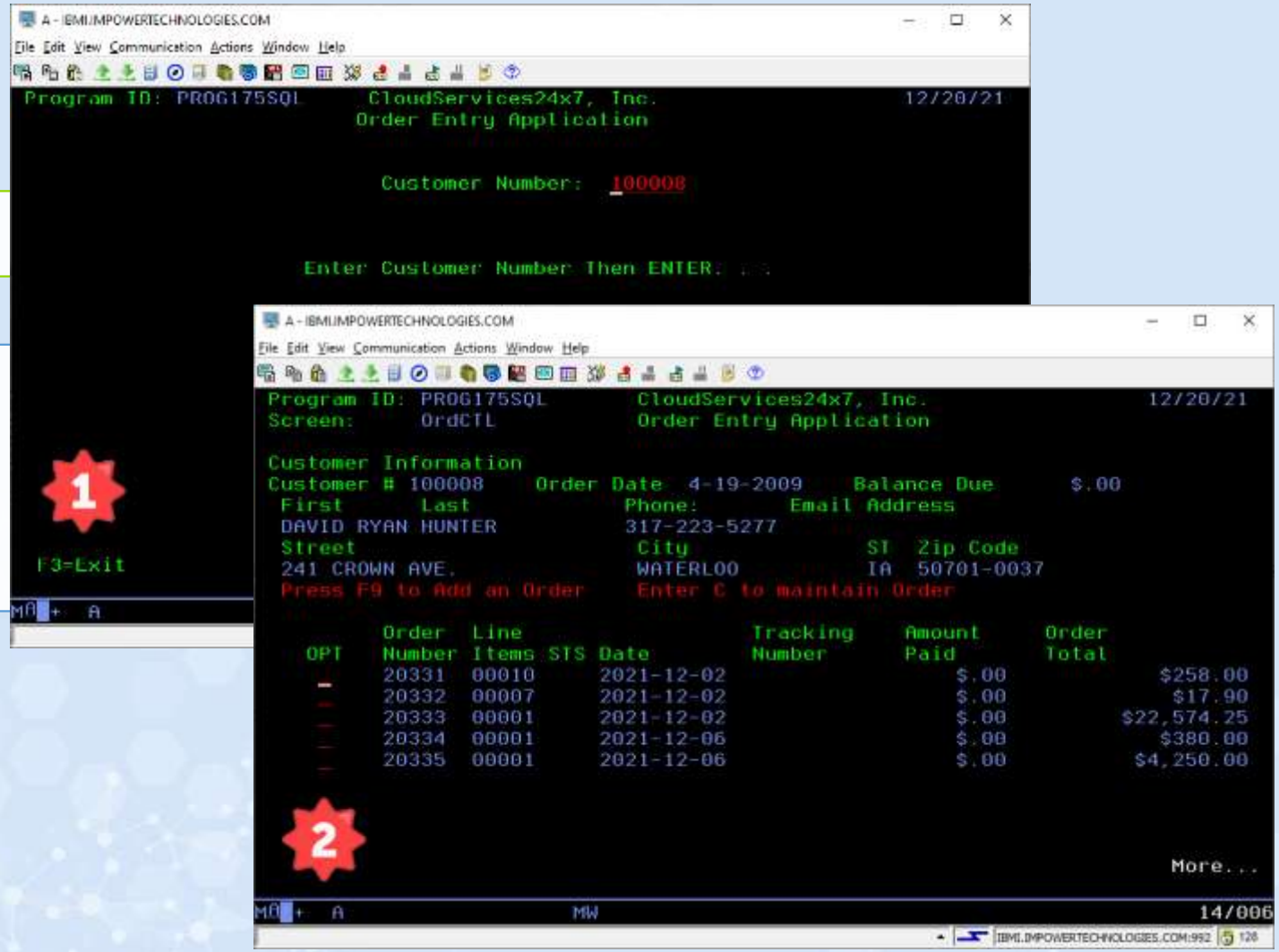


The View

Allow the user to:

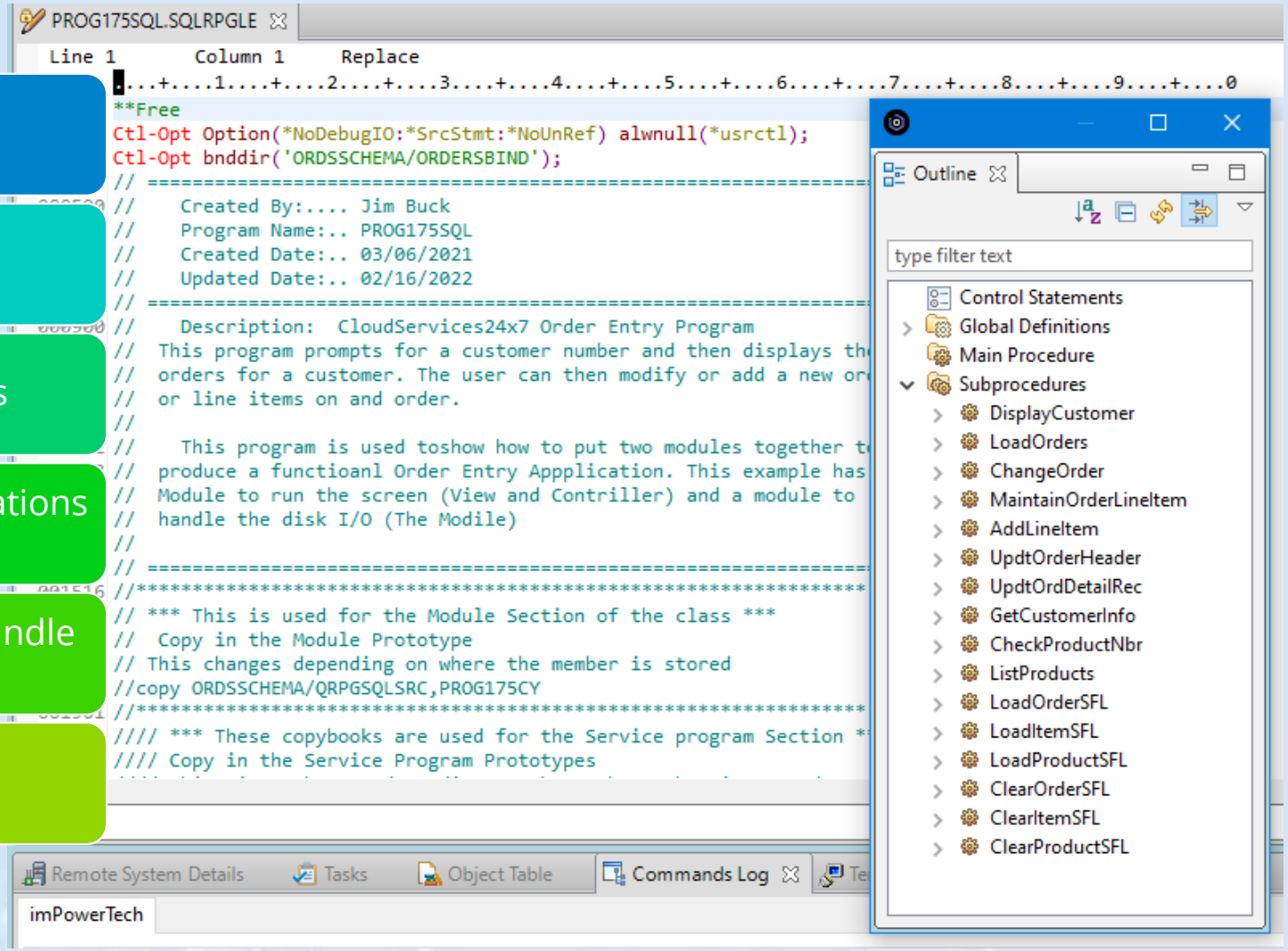
Add / change shipping orders

1. Enter Customer number
2. Lists and change orders for the selected customer



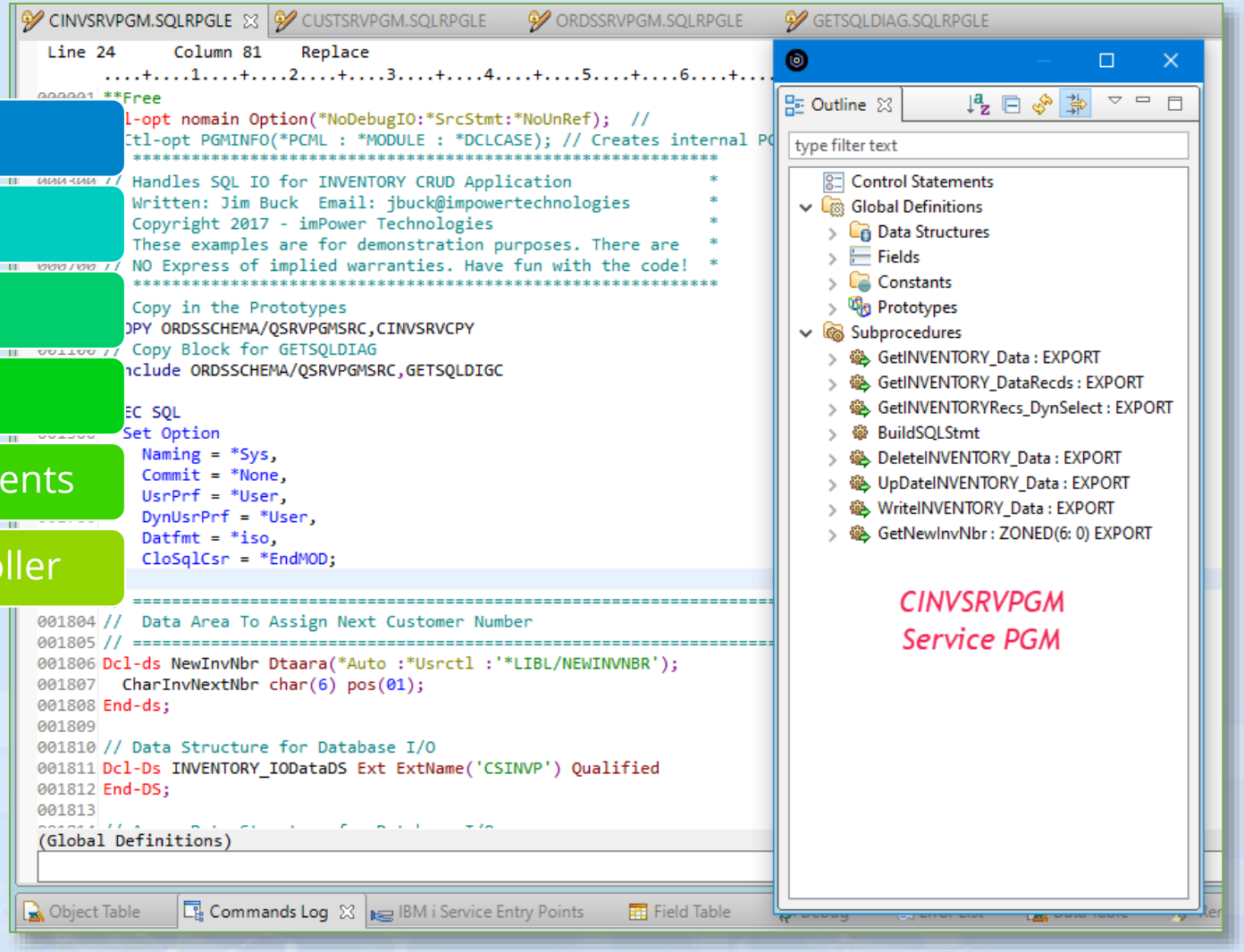
The Controller Program

- PROG175SQL – The Controller
- Contains Controller Subprocedures
- Displays the correct screen based on user interactions
- Based on user interactions contains logic to do calculations needed by the application
- Calls correct service program and subprocedure to handle database updates (The Model)
- Evaluates the results of database requests and sends these results to the user of the application (The View)



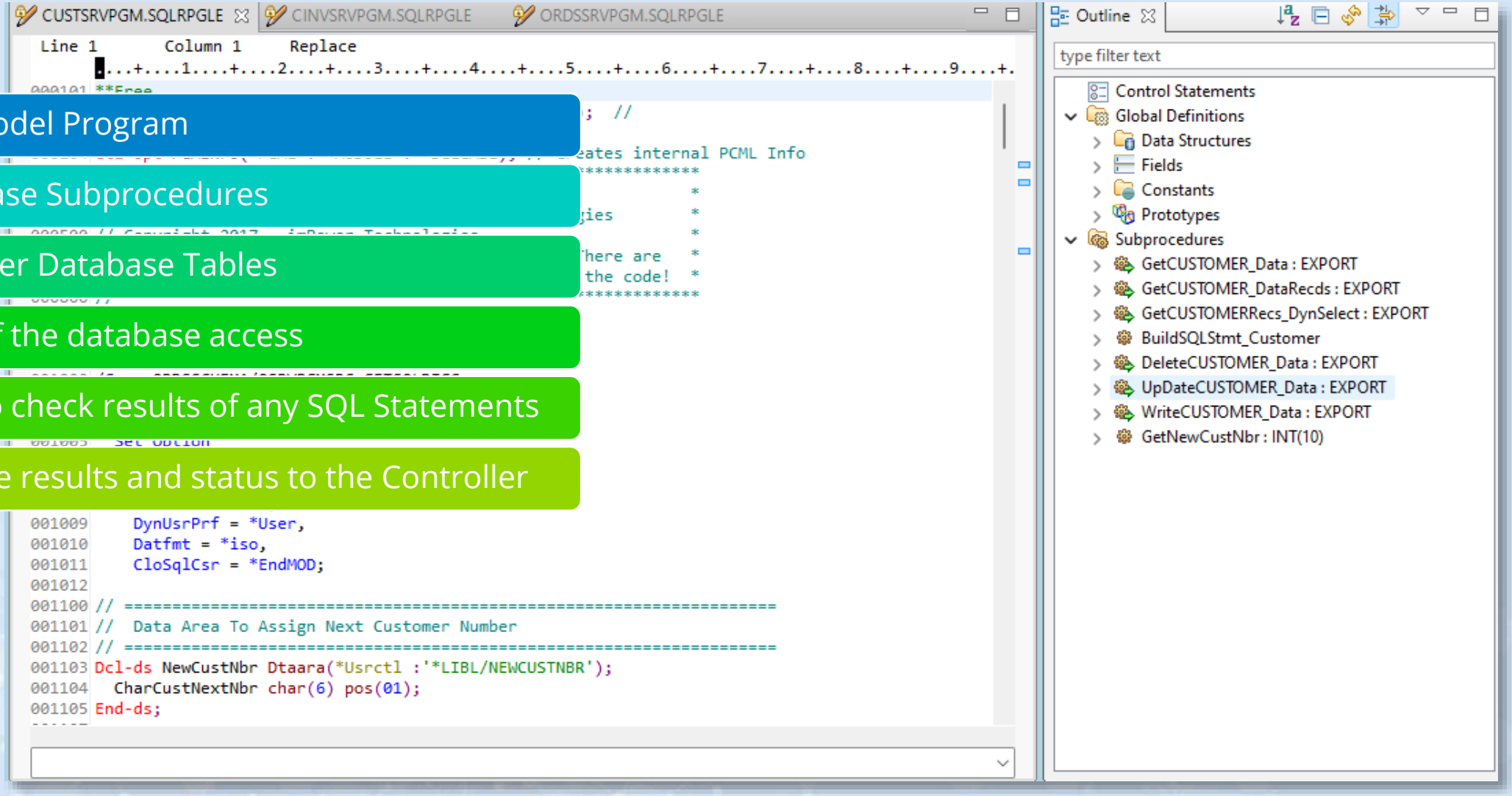
The Model Programs

- CINVSRVPGM – A model Program
- Contains the Data base Subprocedures
- Updates the Inventory Database Tables
- Checks the results of the database access
- Calls GETSQLDIAG to check results of any SQL Statements
- Returns the database results and status to the Controller



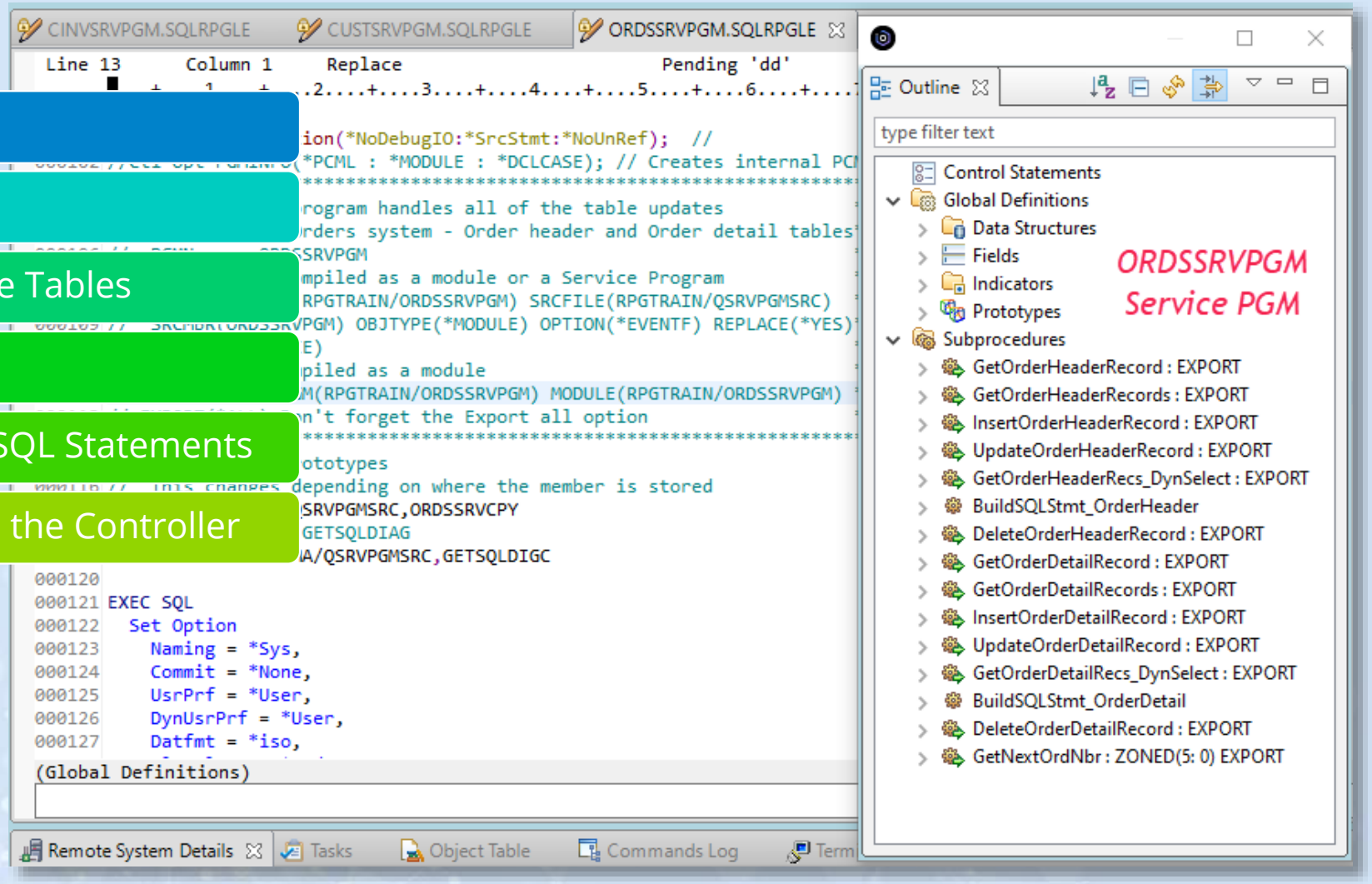
The Model Programs

- CUSTSRVPGM – A model Program
- Contains the Data base Subprocedures
- Updates the Customer Database Tables
- Checks the results of the database access
- Calls GETSQLDIAG to check results of any SQL Statements
- Returns the database results and status to the Controller



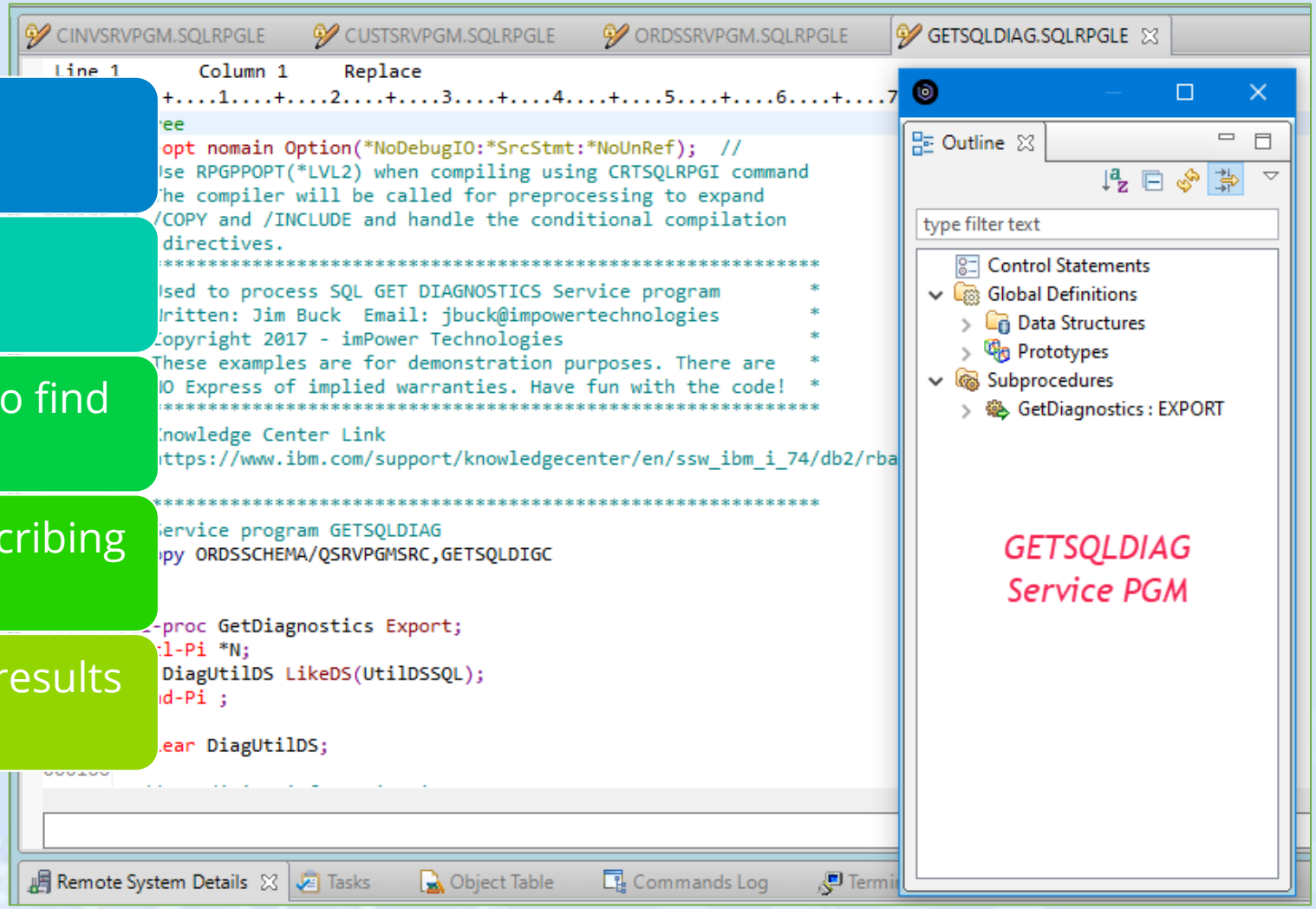
The Model Programs

- ORDSSRVPGM – A Model Program
- Contains the Database Subprocedures
- Updates the Order Header/Detail Database Tables
- Checks the results of the database access
- Calls GETSQLDIAG to check results of any SQL Statements
- Returns the database results and status to the Controller



The GETSQLDIAG program

- GETSQLDIAG – Ancillary Program
- Accepts Data Structures from the calling program
- Runs the GET DIAGNOSTICS Command to find results of last SQL statement
- Handles the logic to set an indicator describing the results of the SQL statement
- Returns a data structure containing the results of the Get Diagnostics command



Activation Groups

Activation Groups

- Nothing more than an isolated area with its own resources
- Contains shared-open, overrides and commitment control to a portion of your job
- Applications can’t interfere with each other
- When an activation group ends all the files used by the group are closed

Activation Groups

Basic Facts

- **OPM Programs** – run in their own environment. If a program is run multiple times the environment needs to be recreated
- **ILE Programs** – still need an environment to run in but they can share these environments and even keep them open between programs. Used properly results in better management
- What's the goal of using ILE Activation groups?
  - Minimize the number of resources consumed creating environments and reduce the overhead of multiple environments
  - Run related programs in the same activation group

Activation Groups

Types of Groups

- Using the CRTRPGPGM command the DFTACTGRP parameter is available
  - Two options
    - ***YES**, the ILE program will act like a OPM program, Why bother?
    - ***NO**, the ILE program will need a named Activation group
      - When set to *NO, the ACTGRP parameter appears

Activation Groups

ACTGRP Parameter - *NEW

- ***NEW** – Causes a NEW activation group to be created when the program is run
  - System will create and name the group every time the program is run.
  - This will create system overhead
  - The system will act like an OPM system

Activation Groups

ACTGRP Parameter – “NAME”

- **“NAME”** – If Specified on every compile, WORKS LIKE *NEW
  - The system will work as an OPM system
  - Except when the program ends the Activation group continues to run and can only be cleaned up is to run the **RCLACTGRP**.

Activation Groups

ACTGRP Parameter – *CALLER

- ***CALLER** – causes the program to run in the same activation group as the program that called it.
- Should only be used when you want a program to run in a known activation group
- Most often used with service programs.

Activation Groups

When the Activation group closes

- Files opened by the activation group close
- Activation-scoped file overrides are freed up
- Static storage used by the programs and service programs gets released
- Allocated storage (%ALLOC) gets released
- Activation-scoped Commitment Control will end and if commitment control is being used database changes will be committed

Error Handling

OPM Program or program created with DFTACTGRP(*YES)

- Handle Exceptions using error indicator, (e), *PSSR, INFSR or an Inquiry message

When using ILE there are additional options

- Percolate the exception to the caller (up the stack)
- Register a procedure to run if the procedure crashes (CEEUTX)
- Register a procedure to run when there is an exception (CEEHDLR)

Introduction to Binding Directories

Some Characteristics of Binding Directories

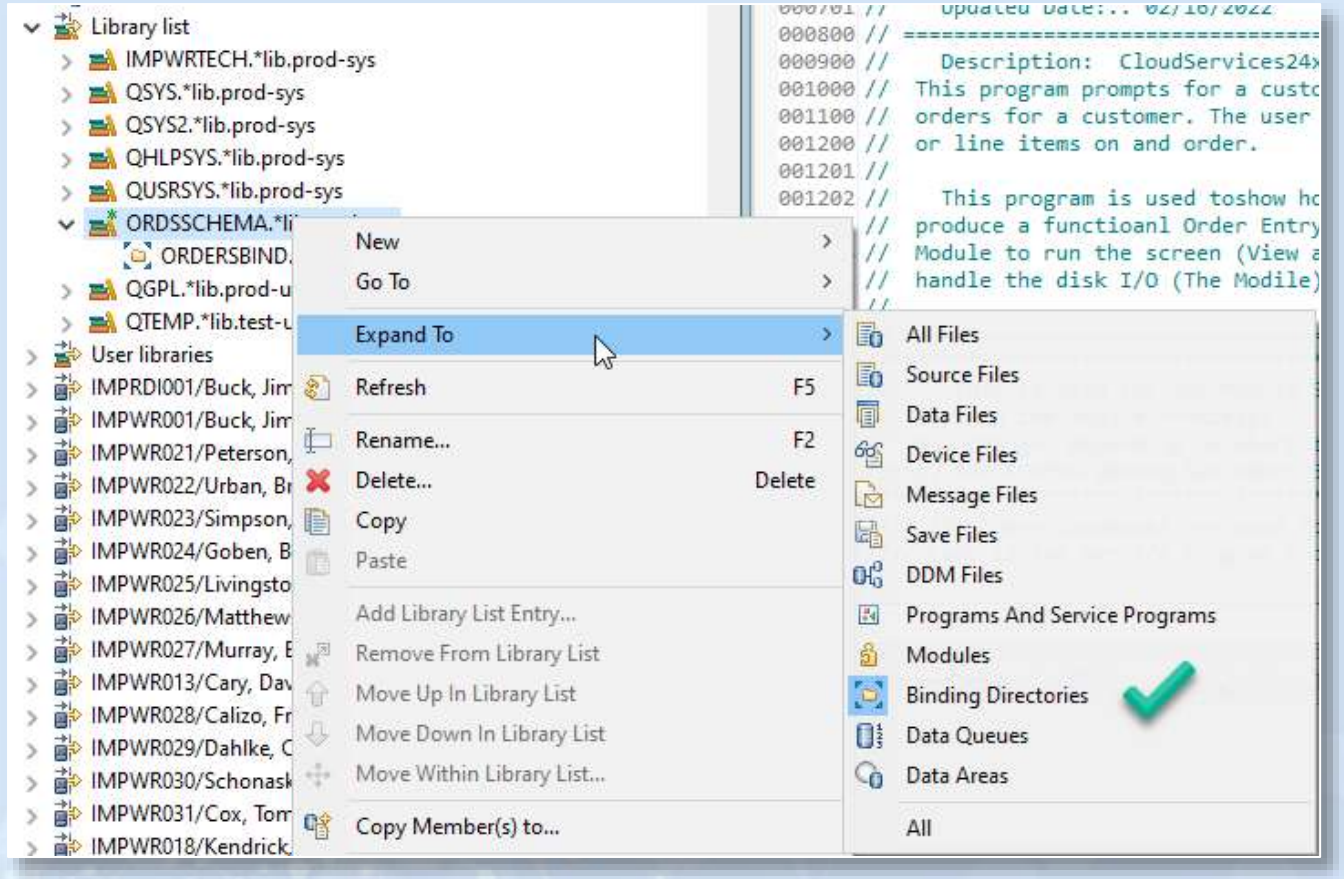
- Convenient method of grouping the names of modules and service programs ***that may be needed*** to create an ILE program or service program
- The object names listed do not have to exist at the time the binding directory is created and populated
- *LIBL or a specific library name are the only valid entries
- The object names listed are optional:
  - The named objects are used only if unresolved imports exist and if named object provides a needed export for an unresolved import request

Binding Directories

Used to list the names of modules and service programs that may be used in an application

They are optional and are used as a convenience and to reduce program size

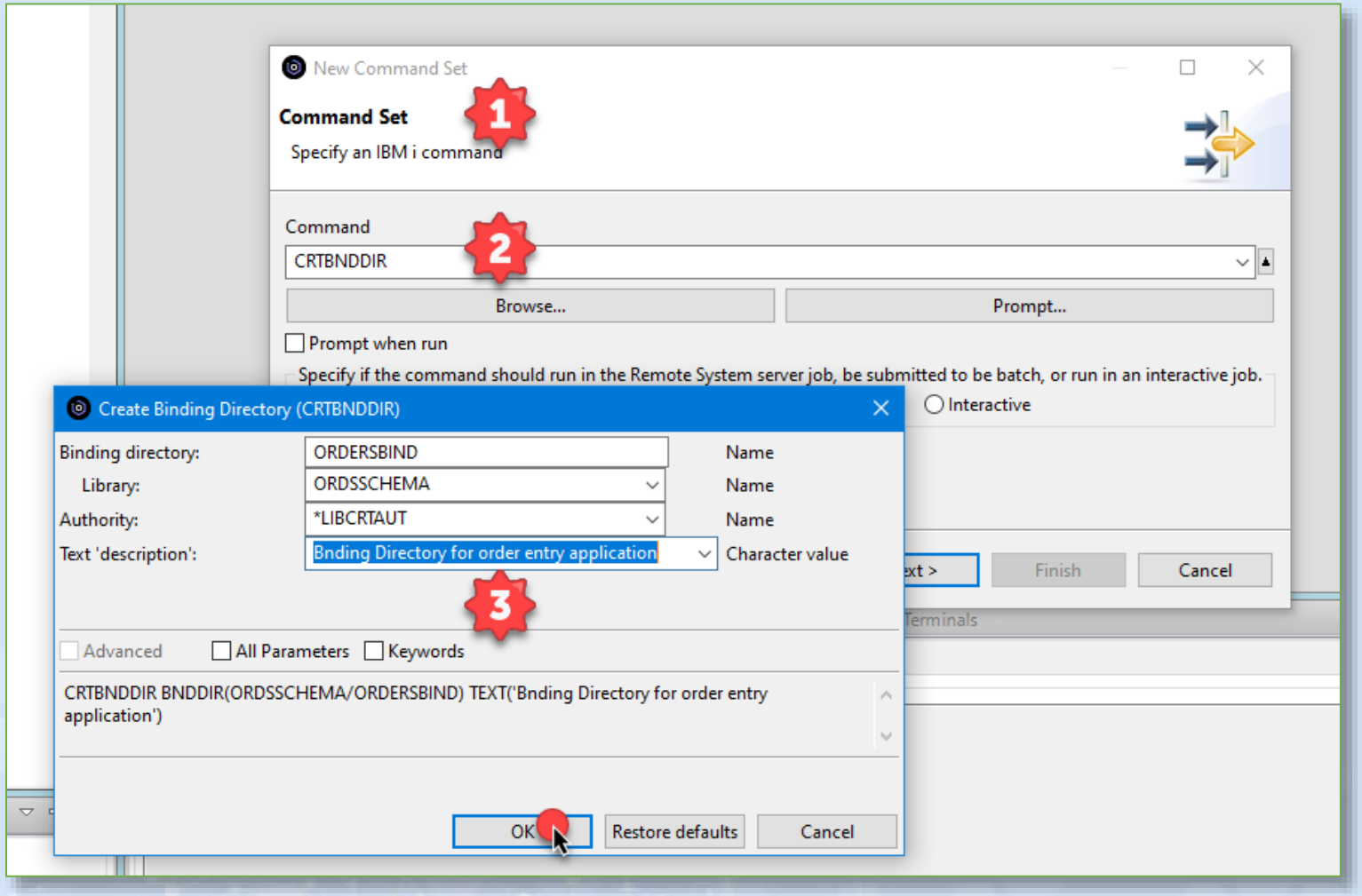
Important to list only modules and service programs that are potentially used



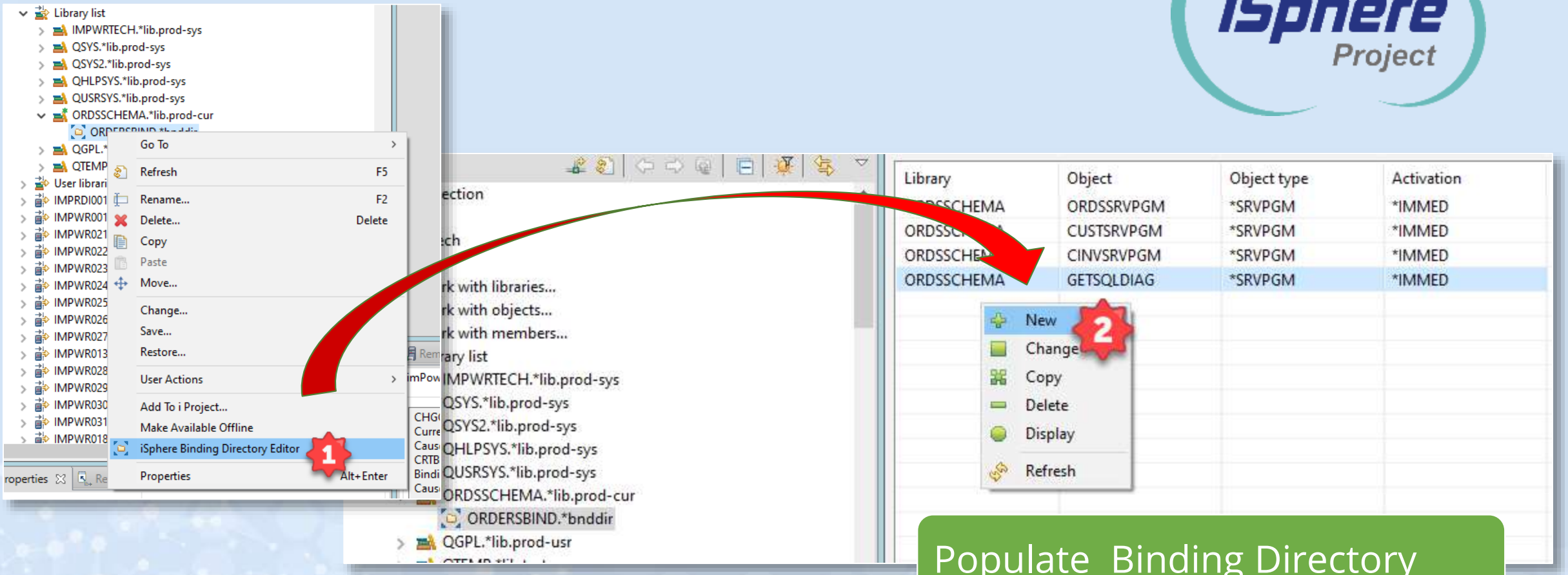
Binding Directories

Create a Binding Directory

1.Create a new Command
2.Type the command name
3.Fill in the blank
Then Ok



Binding Directories



Populate Binding Directory

Expand to: set to Binding Directory

1. Right-Click and select ***iSphere Editor***
2. Right-click and select New

Using Binding Directory

CINVSRVPGM.BND | PROG175D.DSPF | PROG175SQL.SQLRPGLE

```
Line 28      Column 1     Replace
      ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
000001 **Free
000002 Ctl-Opt Option(*NoDebugIO:*SrcStmt:*NoUnRef) alwnull(*usrctl);
000003 Ctl-Opt bnddir('ORDSSCHEMA/ORDERSBIND');
000300 // ================================================================
000500 //    Created By:.... Jim Buck                                    *
000600 //    Program Name:.. PROG175SQL                                  *
000700 //    Created Date:.. 03/06/2021                                  *
000701 //    Updated Date:.. 02/16/2022
000800 // ================================================================
000900 //    Description:  CloudServices24x7 Order Entry Program         *
001000 //  This program prompts for a customer number and then displays the  *
001100 //  orders for a customer. The user can then modify or add a new order*
001200 //  or line items on and order.                                     *
001201 //                                                                  *
001202 //    This program is used toshow how to put two modules together to  *
001203 //  produce a functioanl Order Entry Appplication. This example has a *
001204 //  Module to run the screen (View and Contriller) and a module to    *
001205 //  handle the disk I/O (The Modile)                                *
001206 //                                                                  *
001501 // ================================================================
001516 //****************************************************************
001517 // *** This is used for the Module Section of the class ***
001700 //  Copy in the Module Prototype
001800 // This changes depending on where the member is stored
001900 //copy ORDSSCHEMA/QRPGSQLSRC,PROG175CY
001901 //****************************************************************
001902 //// *** These copybooks are used for the Service program Section ***
001903 //// Copy in the Service Program Prototypes
(Main Procedure)
```

Specifying the Binding Directory

- Use the BNDDIR keyword on the Ctl-Opt spec of your RPG modules
- Use the BNDDIR keyword on the CRTPGM or CRTBNDRPG command
- Use the BNDSRVPGM keyword on the CRTPGM (or CRTSRVPGM) command

Intro to Binder source

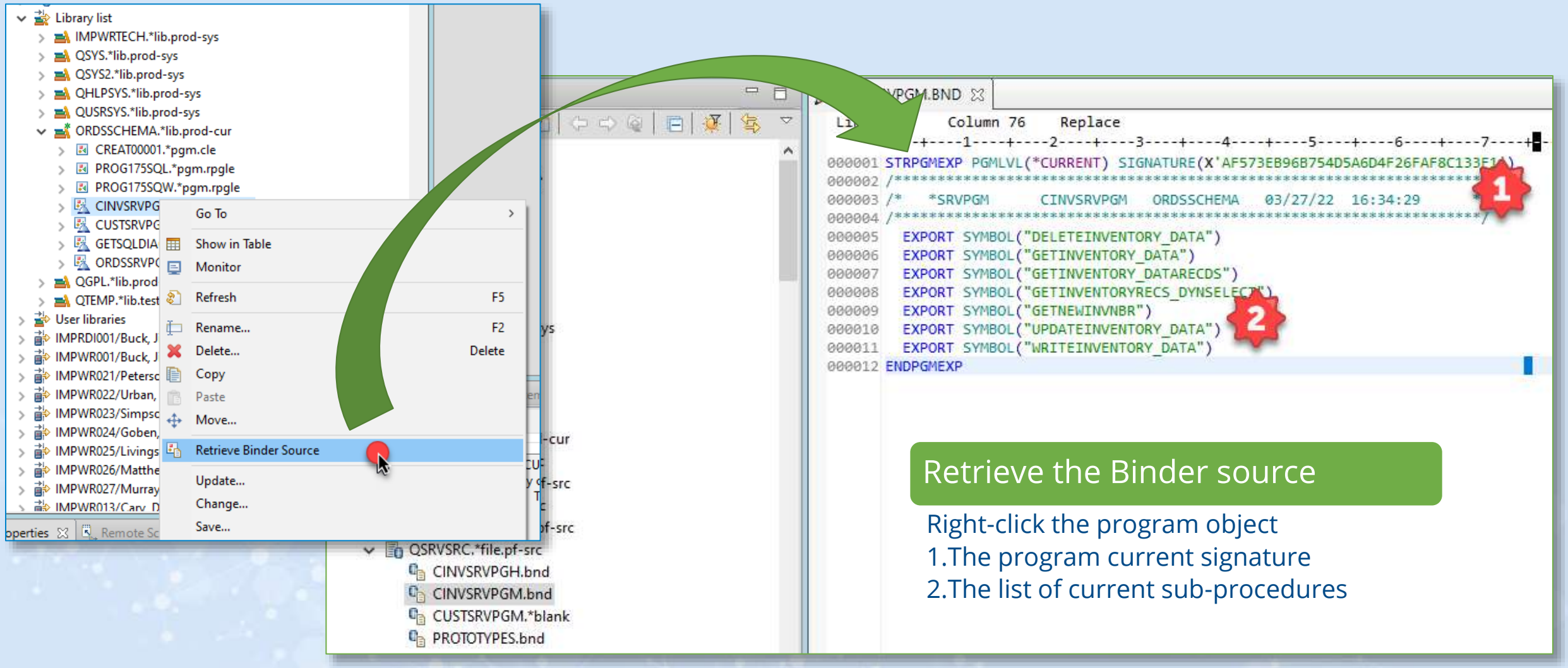
Binder source is used when creating a service program. A signature is created from the order the modules are listed and it shows the order modules are exported

- The file must contain:
  - ***Start Program Export (STRPGMEXP)*** command identifies the beginning of the list of exports from the service program.
  - ***Export Symbol (EXPORT)*** commands identify each symbol name (Procedure) available to be exported from the service program.
  - **End Program Export (ENDPGMEXP)** command identifies the end of the list of exports from the service program.

Important

Do not to change the order of exports in the binder source!

Retrieve Binder source

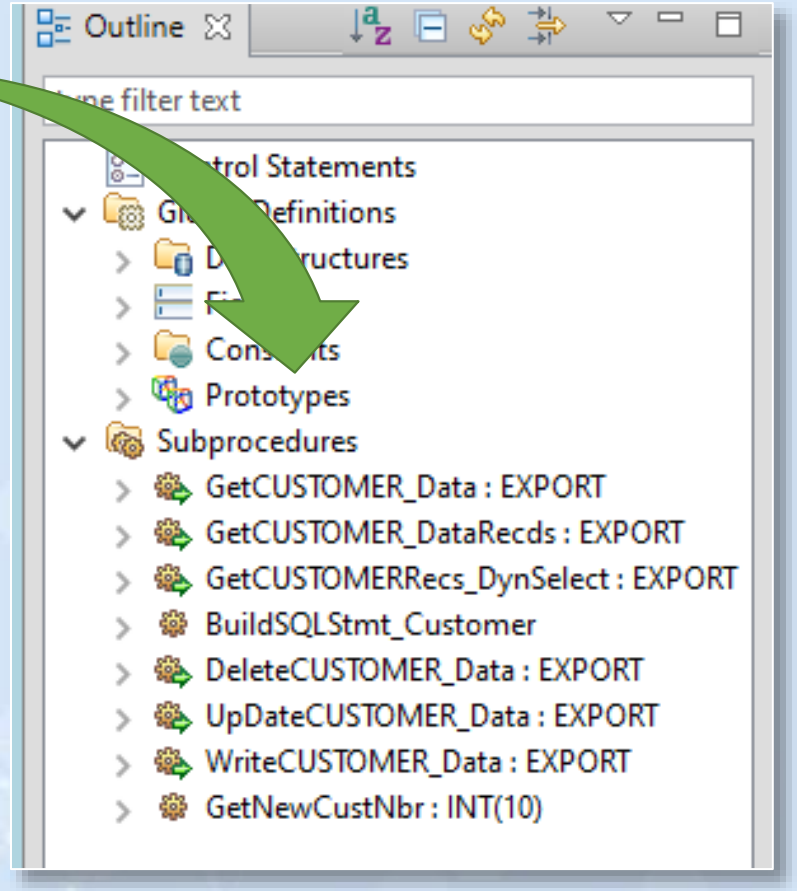


Retrieve the Binder source

Right-click the program object
1.The program current signature
2.The list of current sub-procedures

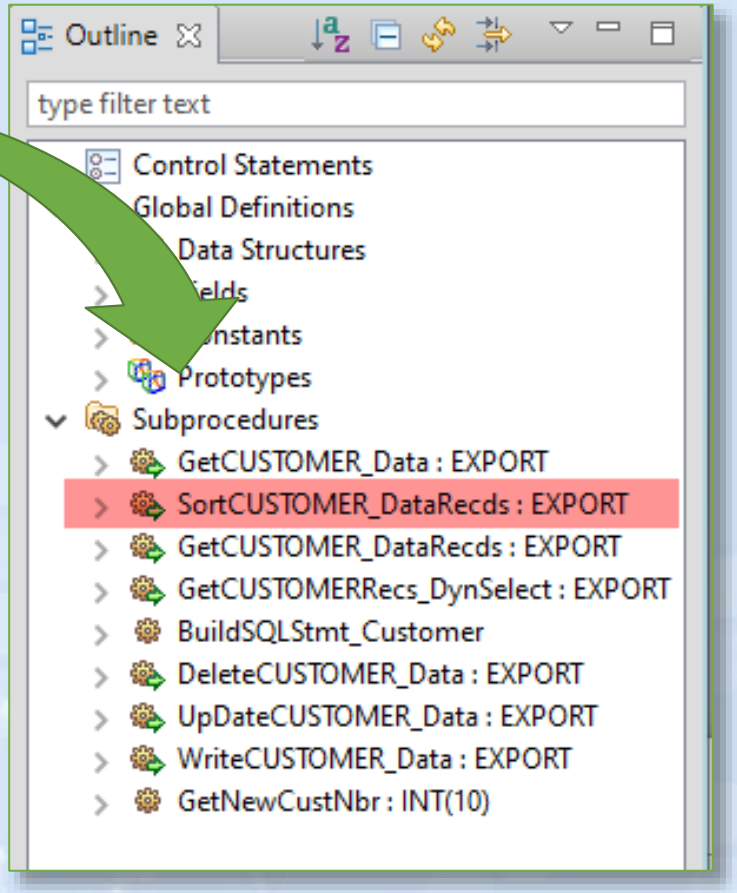
Original Service Program

```
Line 11      Column 43    Insert   6 changes
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+
000001
000014 STRPGMEXP PGMLVL(*CURRENT) SIGNATURE(X'6DA057FC96[illegible]A07C5244F5EFA13FD6')
000015 /*************************************************[illegible]*****************/
000016 /*   *SRVPGM      CUSTSRVPGM   ORDSSCHEMA   [illegible]3/22  16:03:39      */
000017 /*************************************************[illegible]*****************/
000018   EXPORT SYMBOL("GETCUSTOMER_DATA")
000019   EXPORT SYMBOL("GETCUSTOMER_DATARECDS")
000020   EXPORT SYMBOL("GETCUSTOMERRECS_DYNSELECT")
000021   EXPORT SYMBOL("DELETECUSTOMER_DATA")
000022   EXPORT SYMBOL("UPDATECUSTOMER_DATA")
000023   EXPORT SYMBOL("WRITECUSTOMER_DATA")
000024 ENDPGMEXP
```



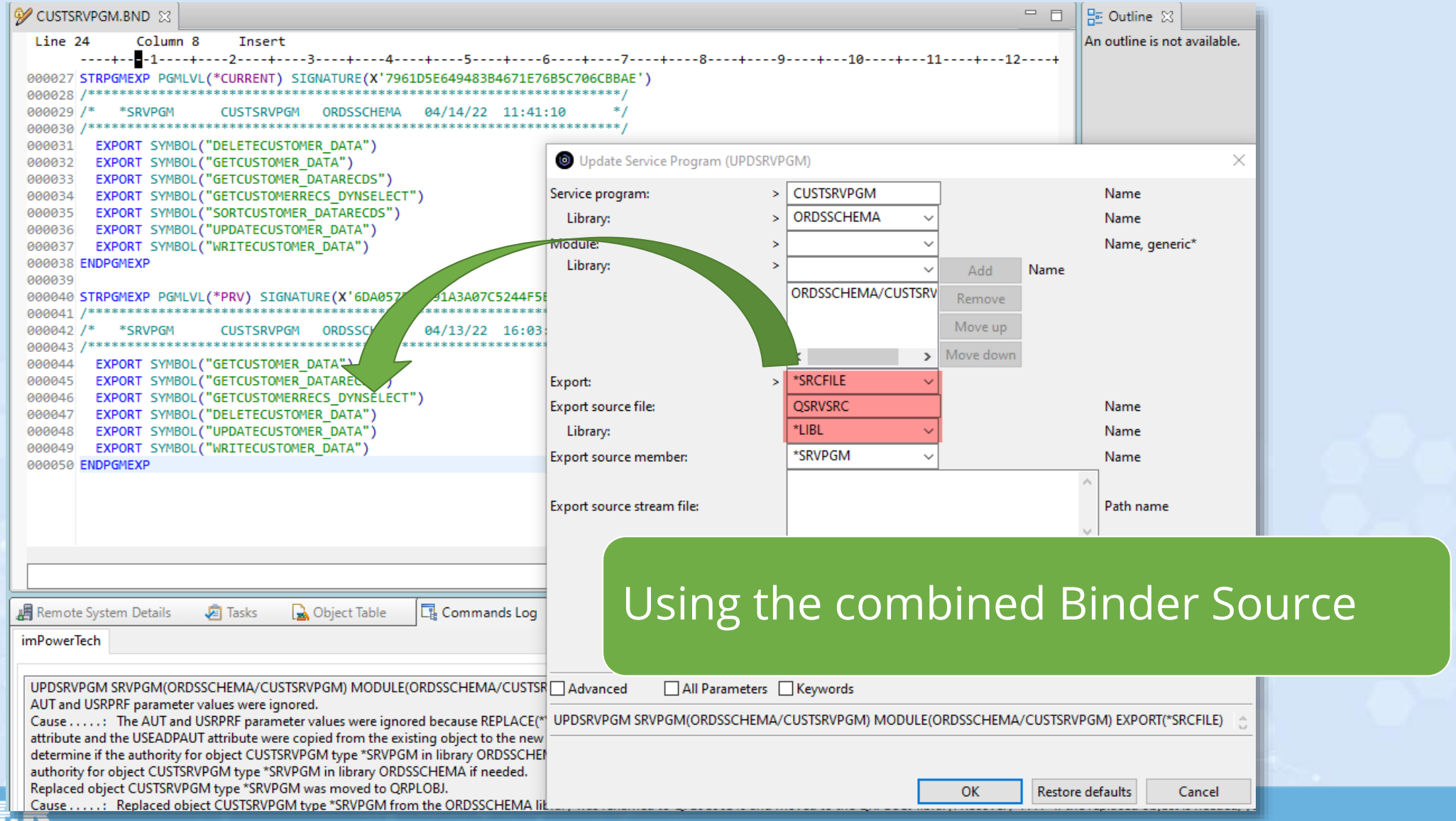
Changed Service Program

```
 Line 1       Column 1     Replace
      -+---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+--
000001 STRPGMEXP PGMLVL(*CURRENT) SIGNATURE(X'7961D5E649483B4[illegible]E76B5C706CBBAE')
000002 /*************************************************[illegible]*************/
000003 /*    *SRVPGM       CUSTSRVPGM    ORDSSCHEMA    [illegible]4/22  11:41:10       */
000004 /*************************************[illegible]*************************/
000005    EXPORT SYMBOL("DELETECUSTOMER_DATA")
000006    EXPORT SYMBOL("GETCUSTOMER_DATA")
000007    EXPORT SYMBOL("GETCUSTOMER_DATARECD[illegible]
000008    EXPORT SYMBOL("GETCUSTOMERRECS_DYNS[illegible]T")
000009    EXPORT SYMBOL("SORTCUSTOMER_DATARECDS")
000010    EXPORT SYMBOL("UPDATECUSTOMER_DATA")
000011    EXPORT SYMBOL("WRITECUSTOMER_DATA")
000012 ENDPGMEXP
```



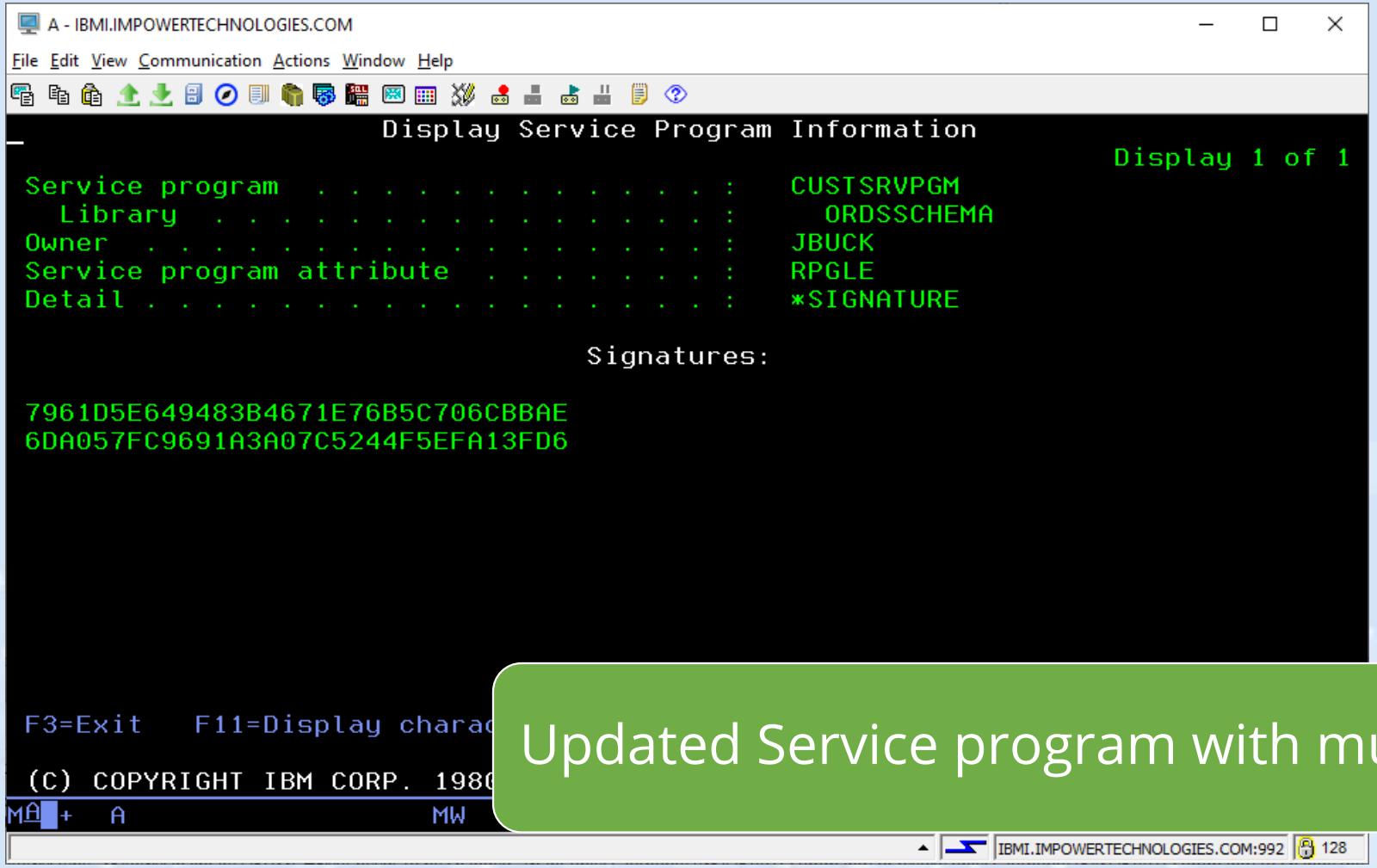
New Binder Source shows the new procedure

Updated Service Program



Using the combined Binder Source

Updated Service Program



Updated Service program with multiple signatures

Create the Application

Step 01

- Create the 5250 Display file PROG175D

Step 02

- Create The Module PROG175SQL

Step 03

- Create the Module CUSTSVRPGM
- Create the Service Progam CUSTSVRPGM

Step 04

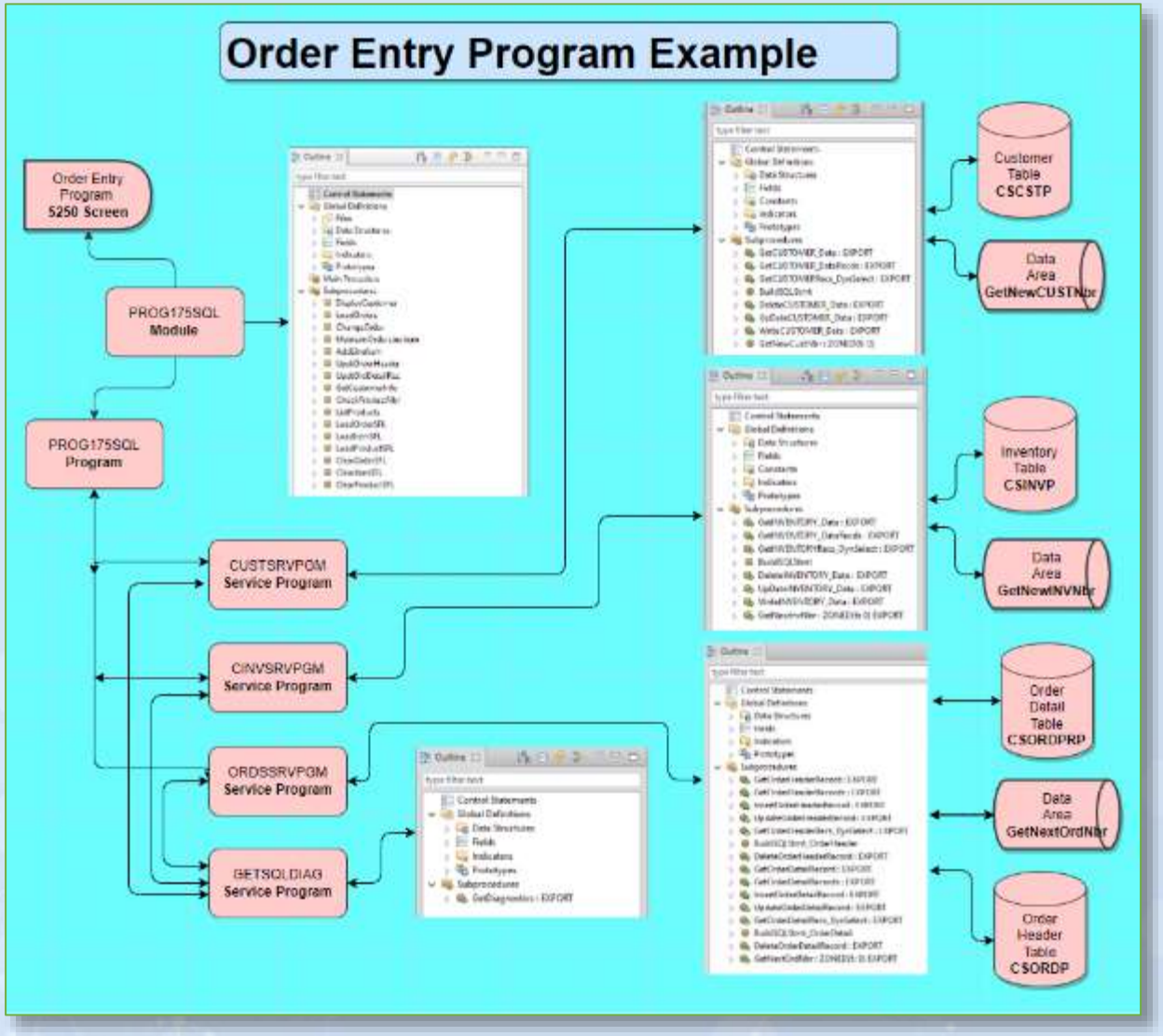
- Create The Module CINSVRPGM
- Create The Service Program CINSVRPGM

Step 05

- Create The Module ORDSSVRPGM
- Create The Service Program ORDSSVRPGM

Step 06

- Create the Executable PROG175SQL



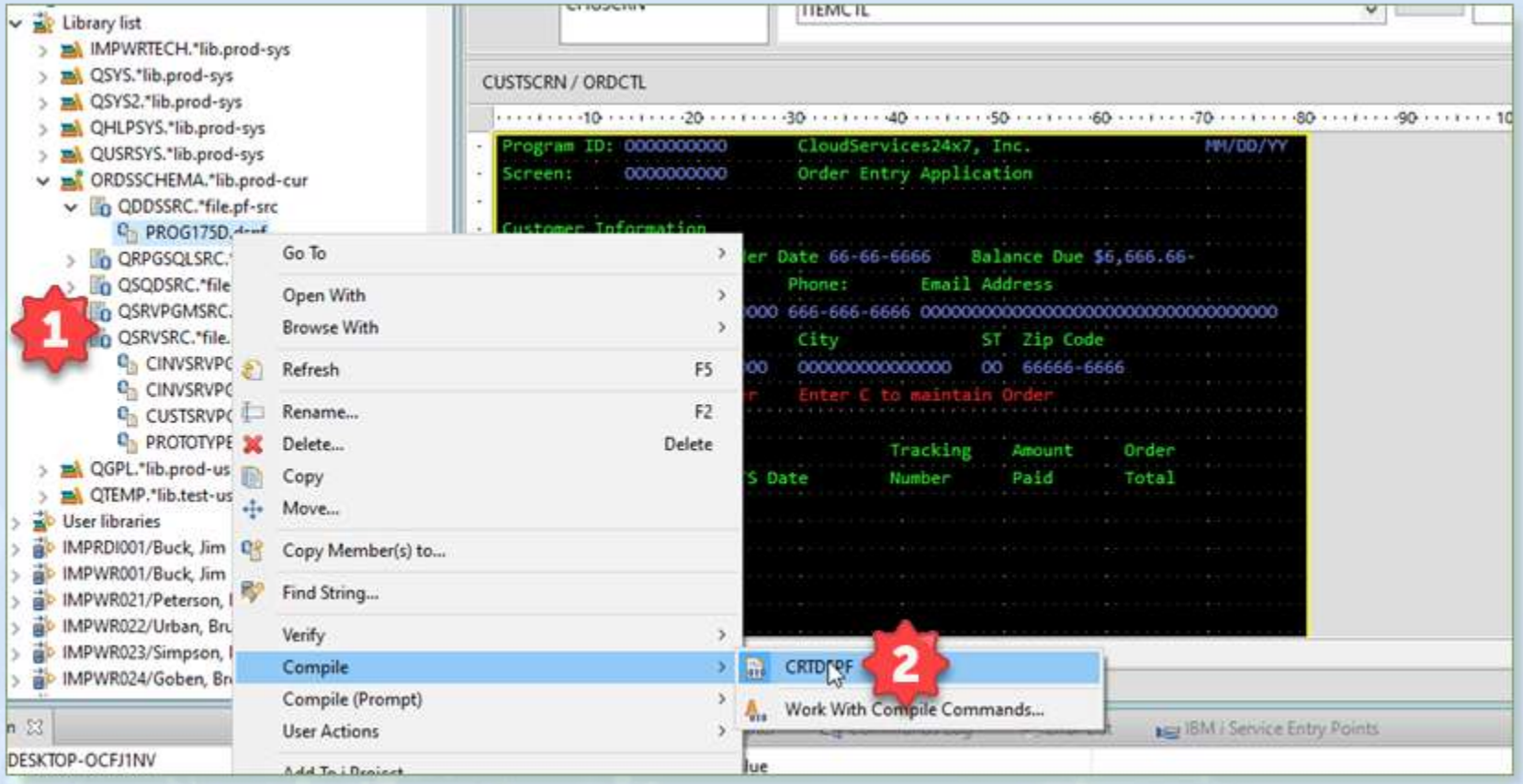
Create PROG175D Display File

Create the Display file

Right-click the program source member

1. Expand the Library>source file until you find the source member
2. Select the CRTDSPF

Always check that the compile worked

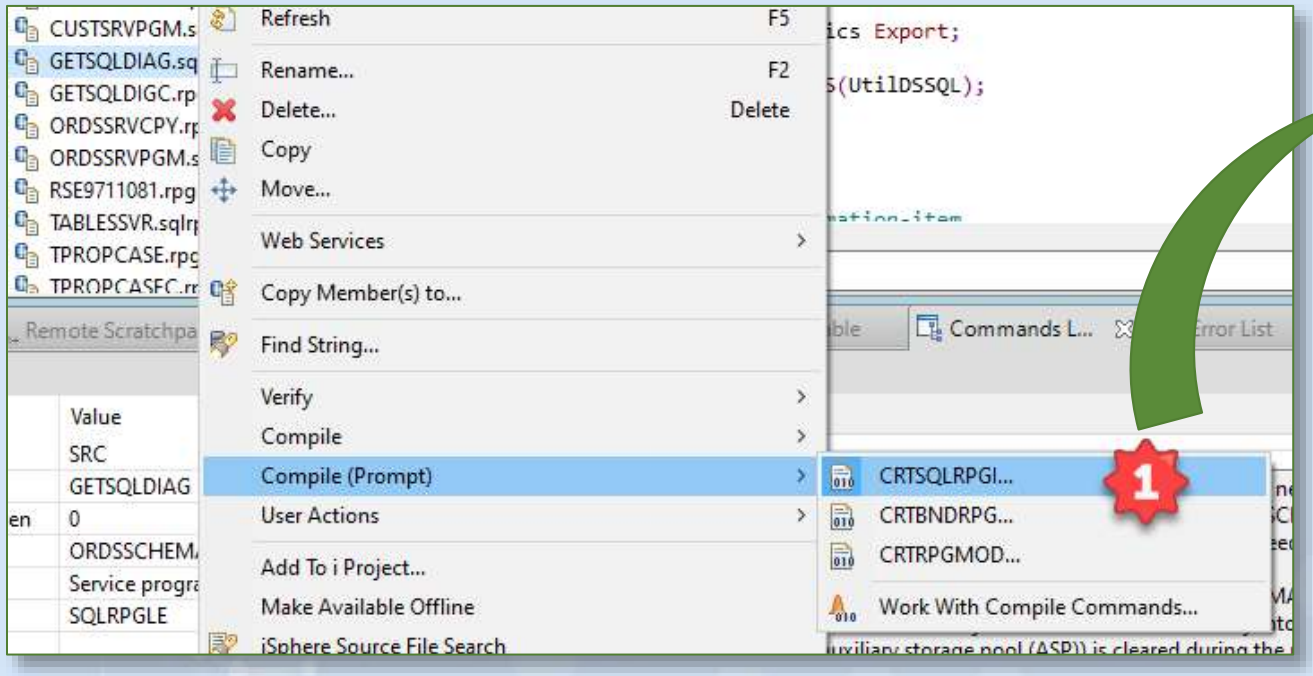


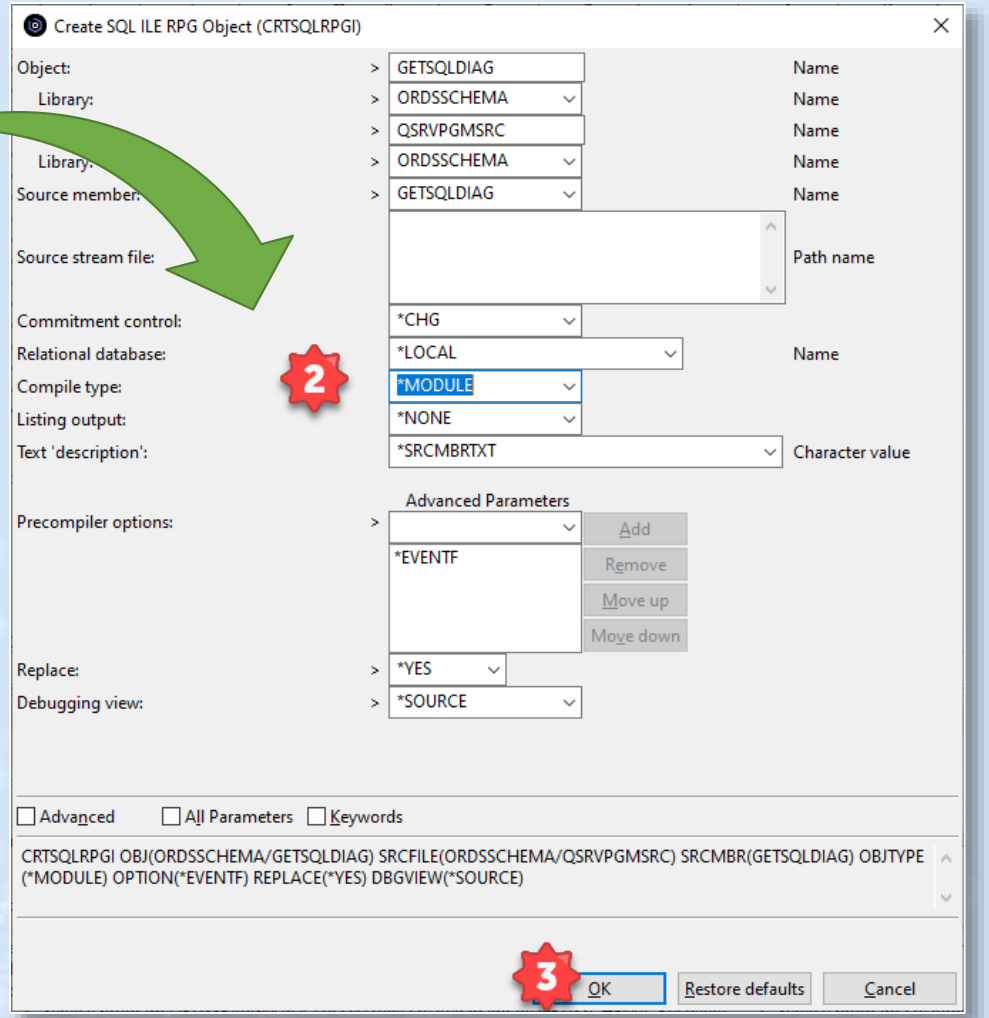
Activation Group information

Application ILE Information

- PROG175SQL – Will run in its own named **ORDERENTRY** activation group
- All Service programs will use *CALLER for the ACTGRP parameter
- Binder Directory name is **ORDERSBIND**

Create GETSQLDIAG Module

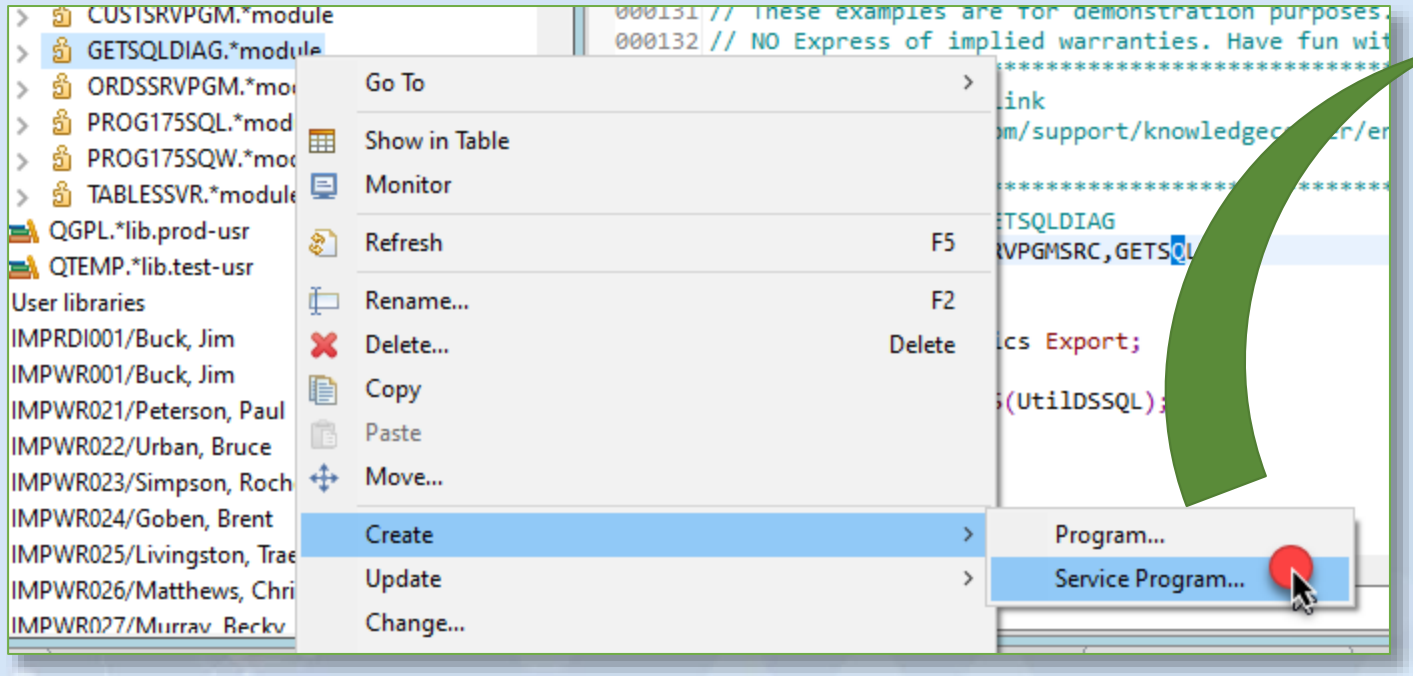


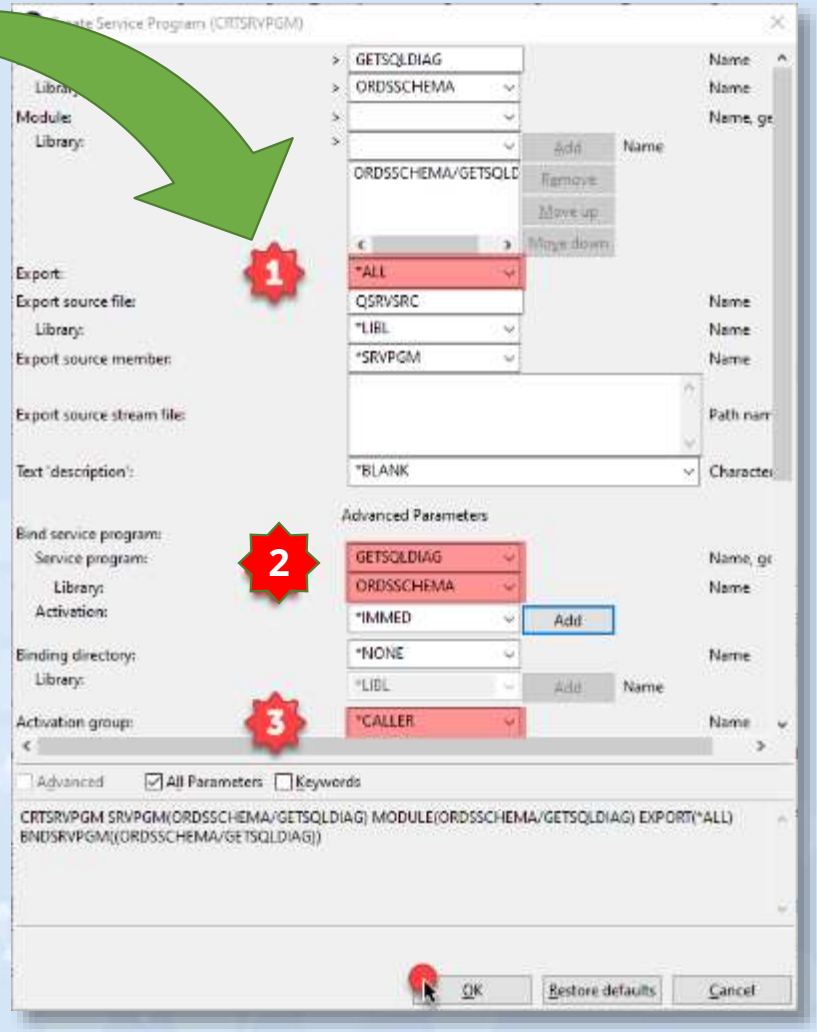


Create GETSQLDIAG Module

Right-click the program source member
1.Select the Compile (Prompt) > CRTSQLRPGI
2.Change Compile type to *Module
3.Click-on Ok
Always check that the compile worked under commands log view

Create GETSQLDIAG Service Program

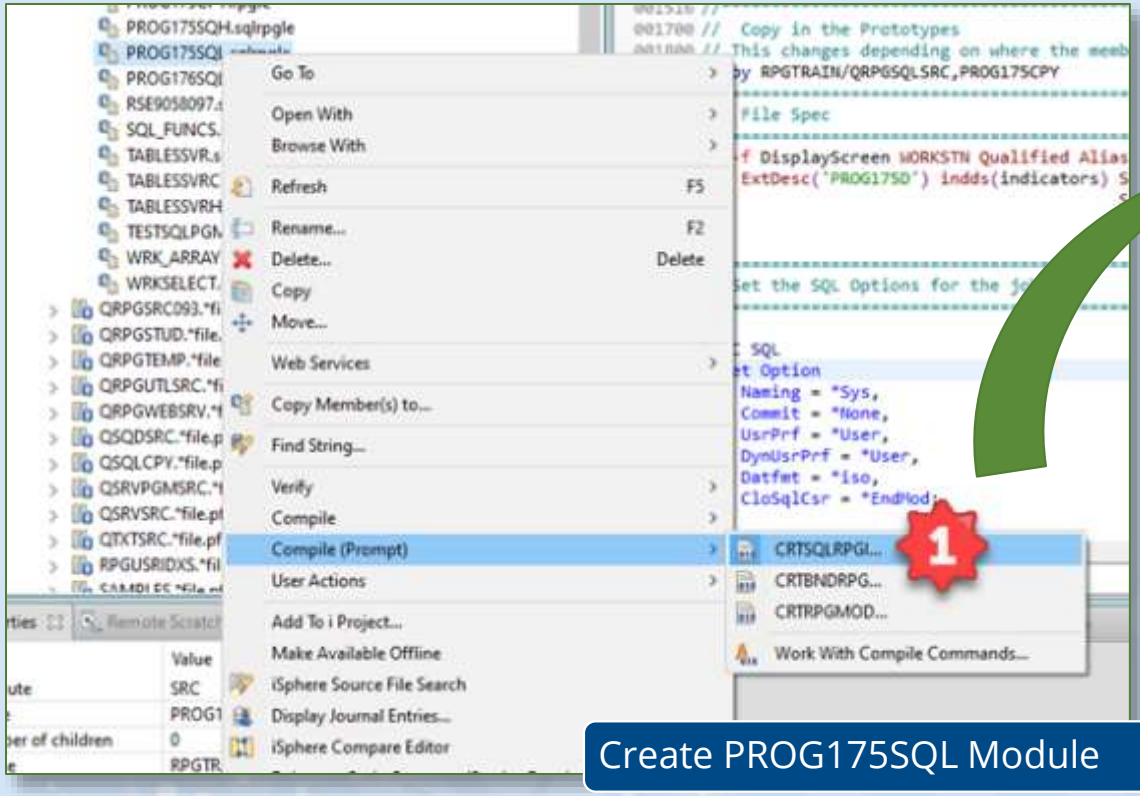


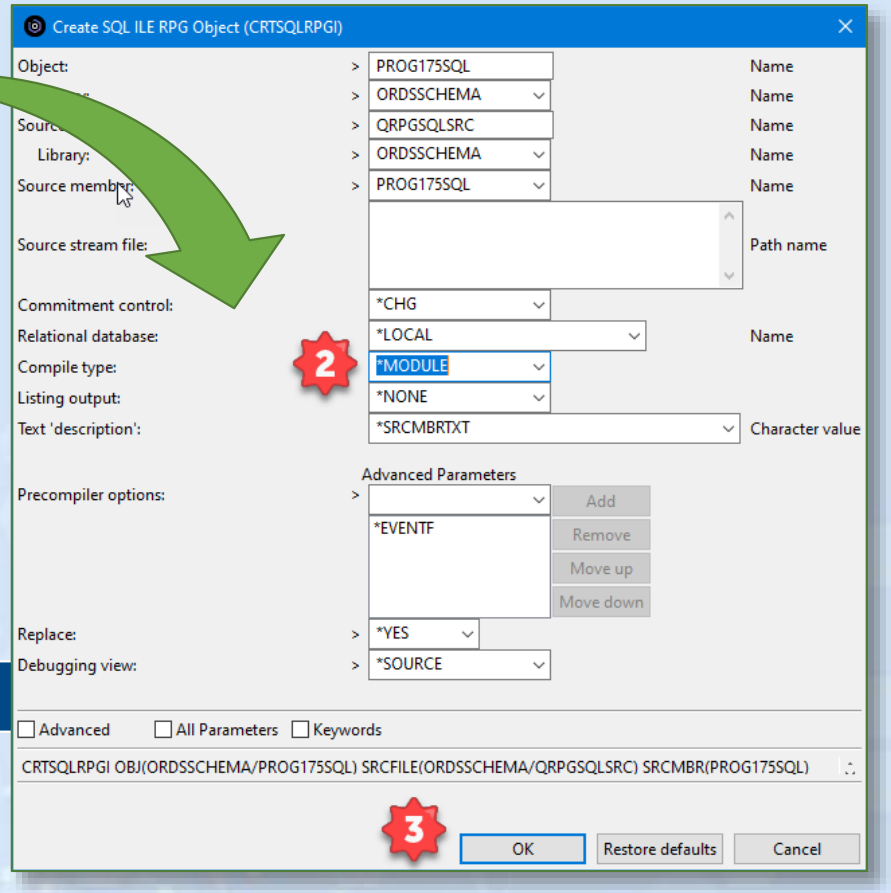


Create GETSQLDIAG Service Program

Right-click the program module THEN Select Create -> Service program
1.Change export to *ALL
2.Add the **YourLib**/GETSQLDIAG Service program
3.Use the *CALLER to the ACTGRP parameter
Click “Ok”

Create PROG175SQL Module

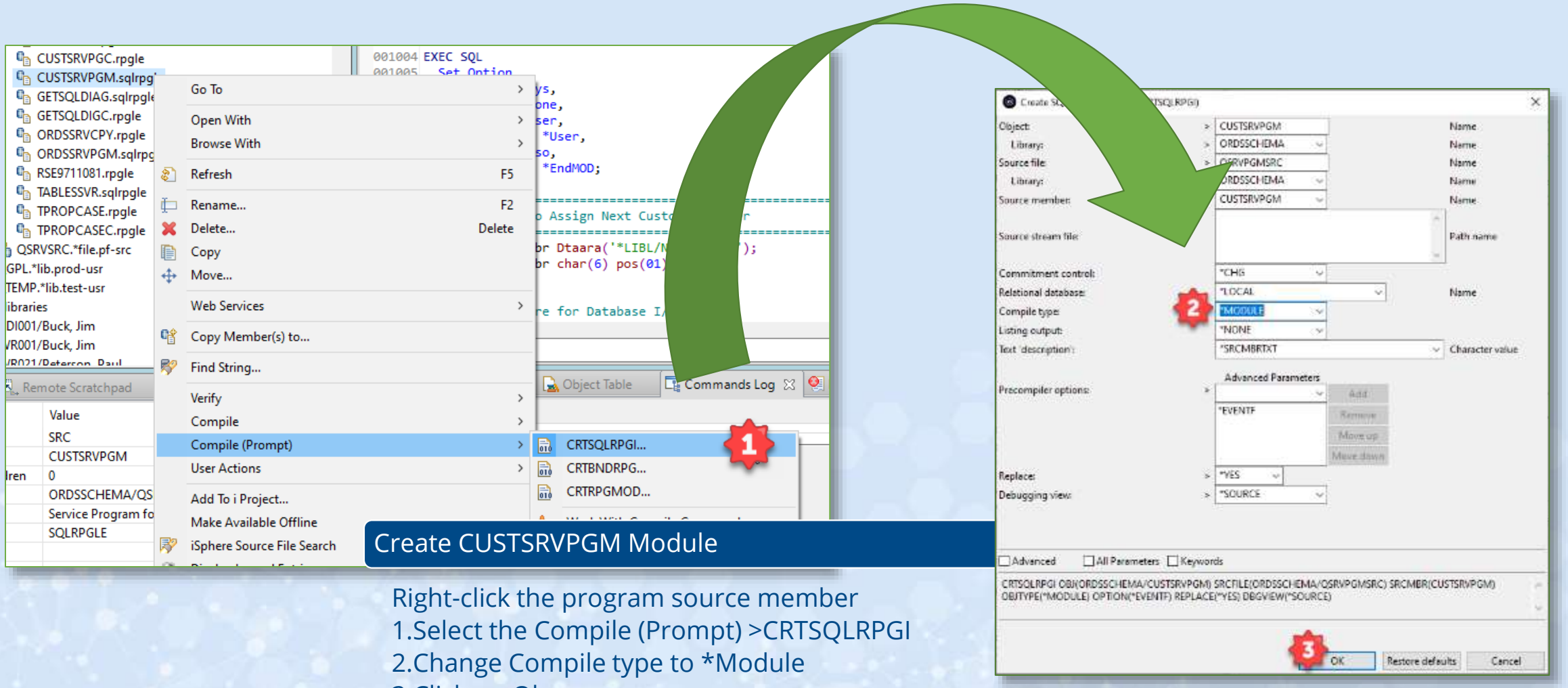




Create PROG175SQL Module

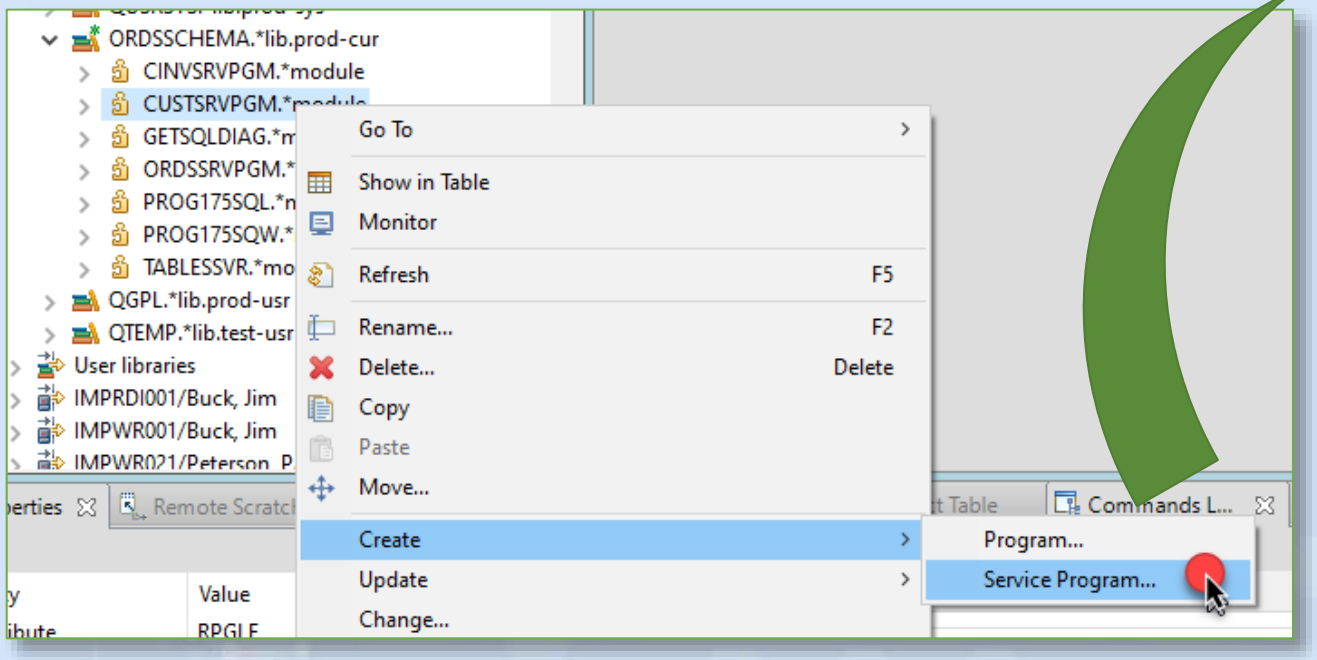
Right-click the program source member
1.Select the Compile (Prompt) >CRTSQLRPGI
2.Change Compile type to *Module
3.Click-on Ok
Always check that the compile worked under commands log view

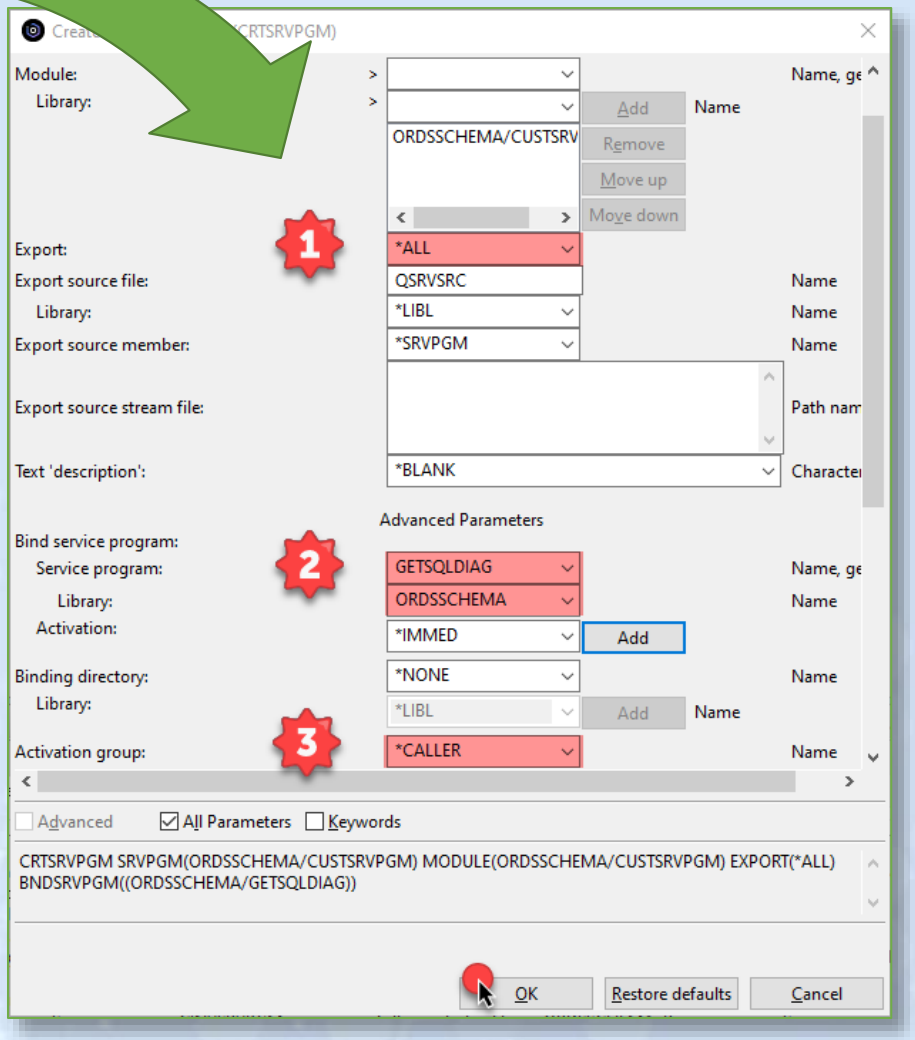
Create CUSTSRVPGM Module



Right-click the program source member
1.Select the Compile (Prompt) >CRTSQLRPGI
2.Change Compile type to *Module
3.Click-on Ok
Always check that the compile worked under commands log view

Create CUSTSRVPGM Service Program

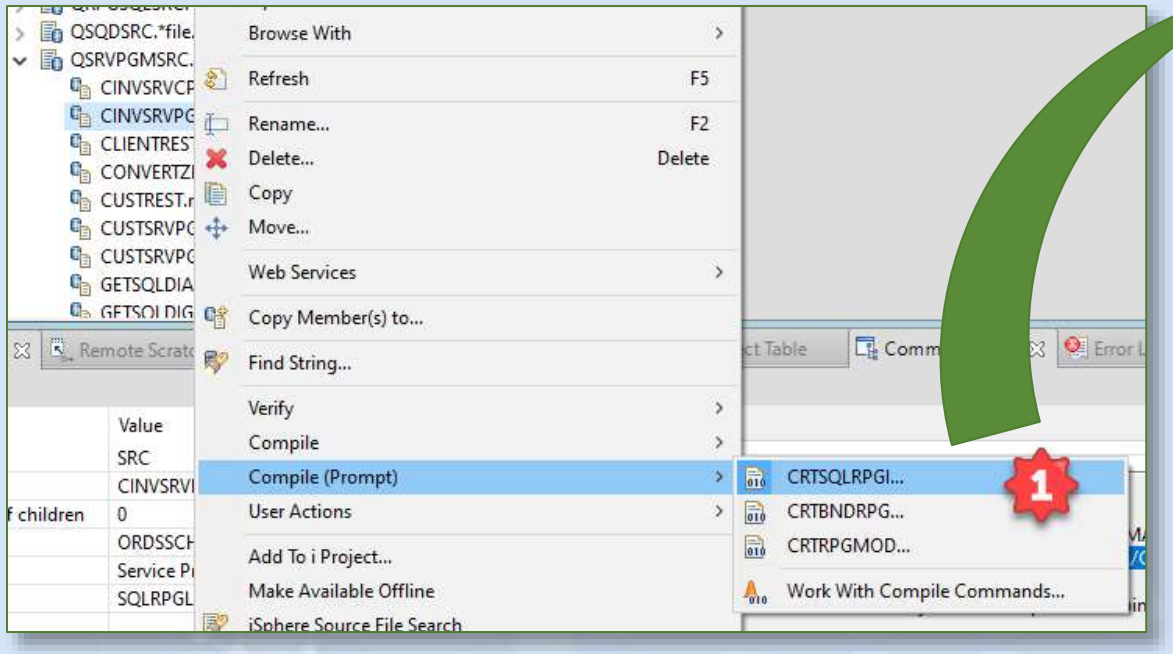


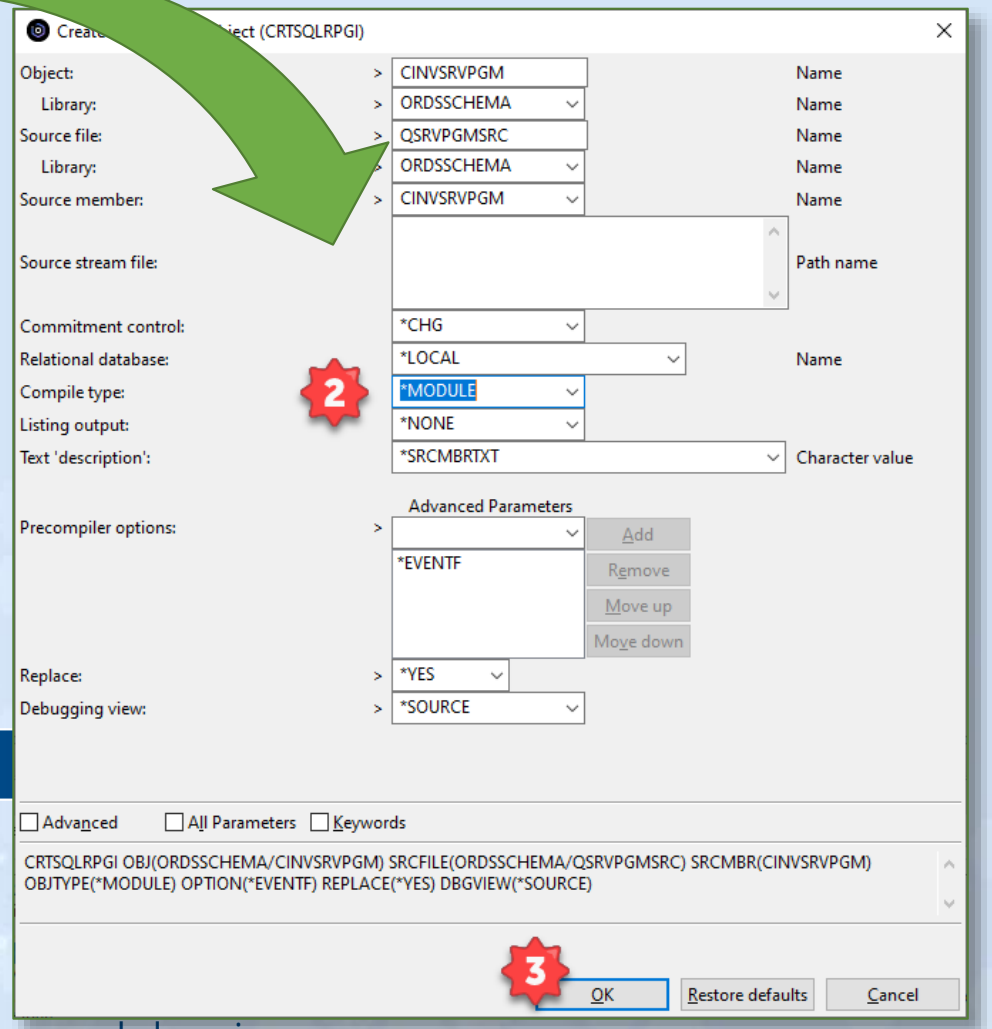


Create CUSTSRVPGM Service Program

Right-click the program module THEN Select Create -> Service program
1.Change export to *ALL
2.Add the **YourLib**/GETSQLDIAG Service program
3.Use the *CALLER to the ACTGRP parameter
Click “Ok”

Create CINVSRVPGM Module

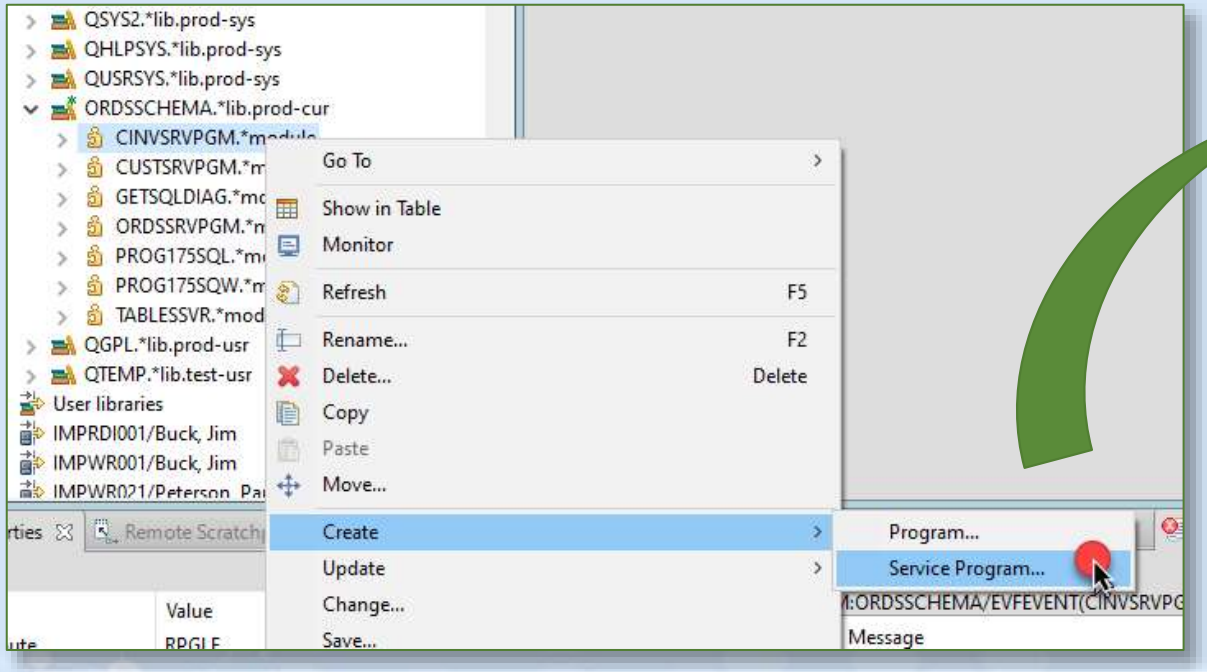


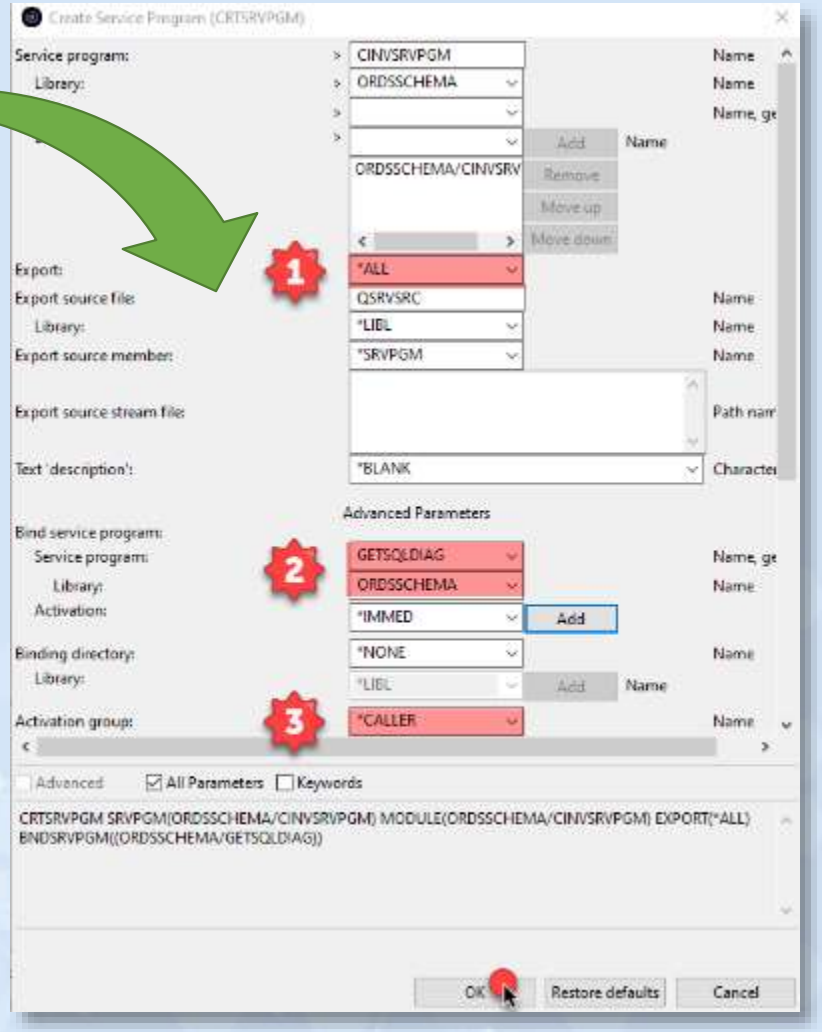


Create CINVSRVPGM Module

Right-click the program source member
1.Select the Compile (Prompt) > CRTSQLRPGI
2.Change Compile type to *MODULE
3.Click-on Ok
Always check that the compile worked under commands log view

Create CINVSRVPGM Service Program

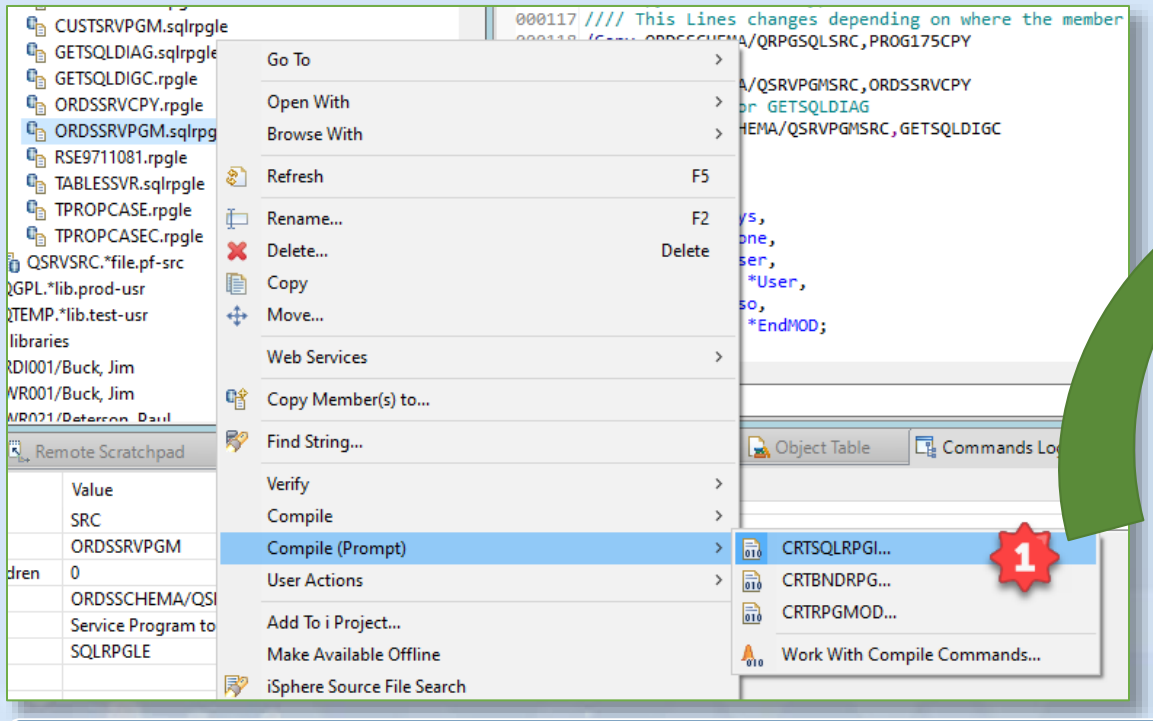


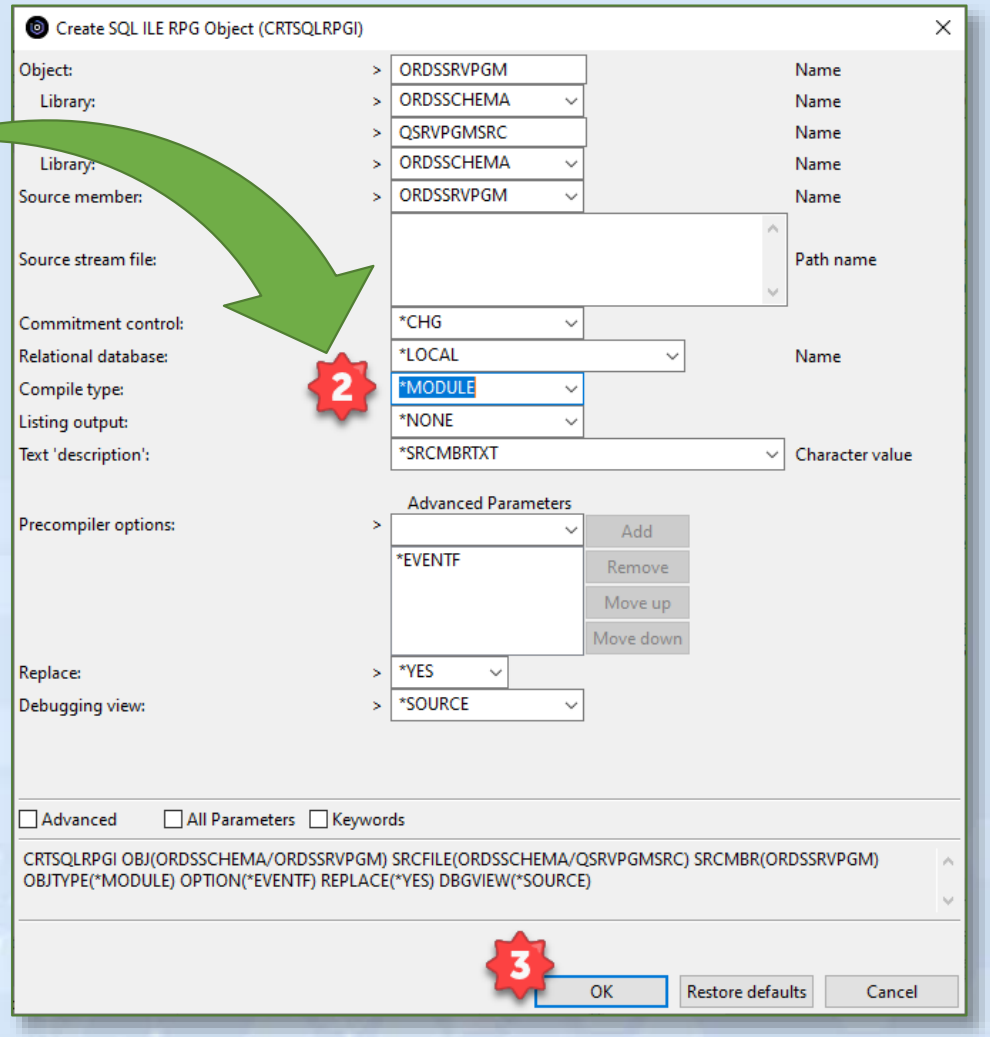


Create CINVSRVPGM Service Program

Right-click the program module THEN Select Create -> Service program
1.Change export to *ALL
2.Add the **YourLib**/GETSQLDIAG Service program
3.Use the *CALLER IN the ACTGRP parameter
Click “Ok”

Create ORDSSRVPGM Module

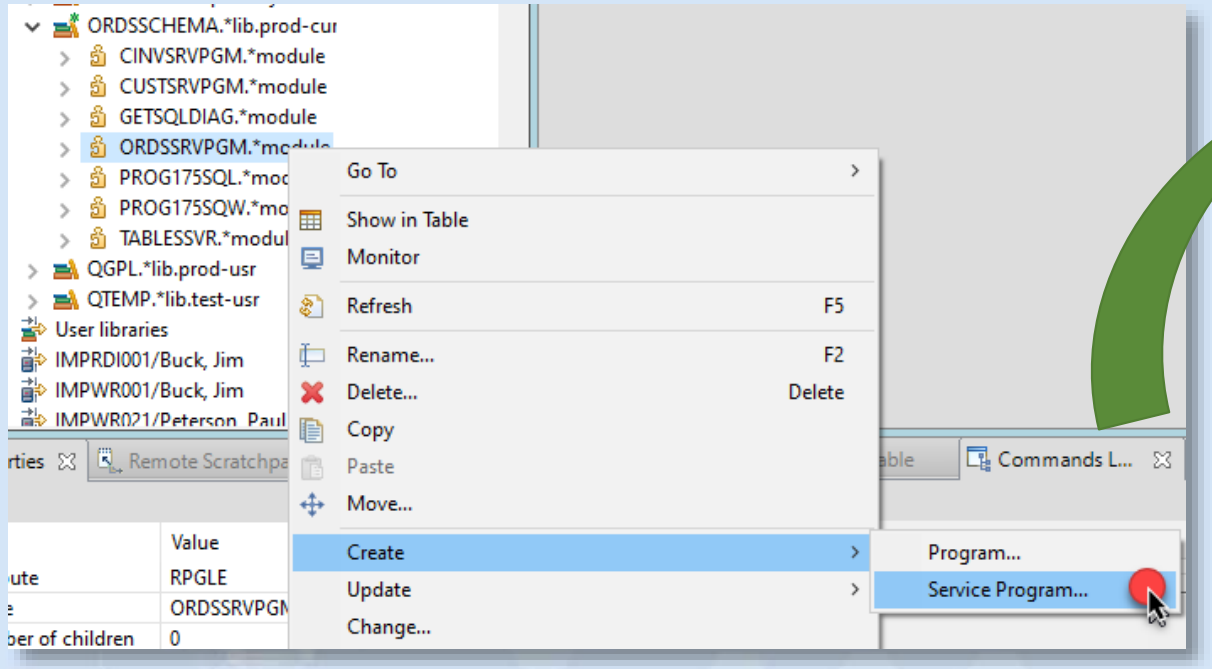


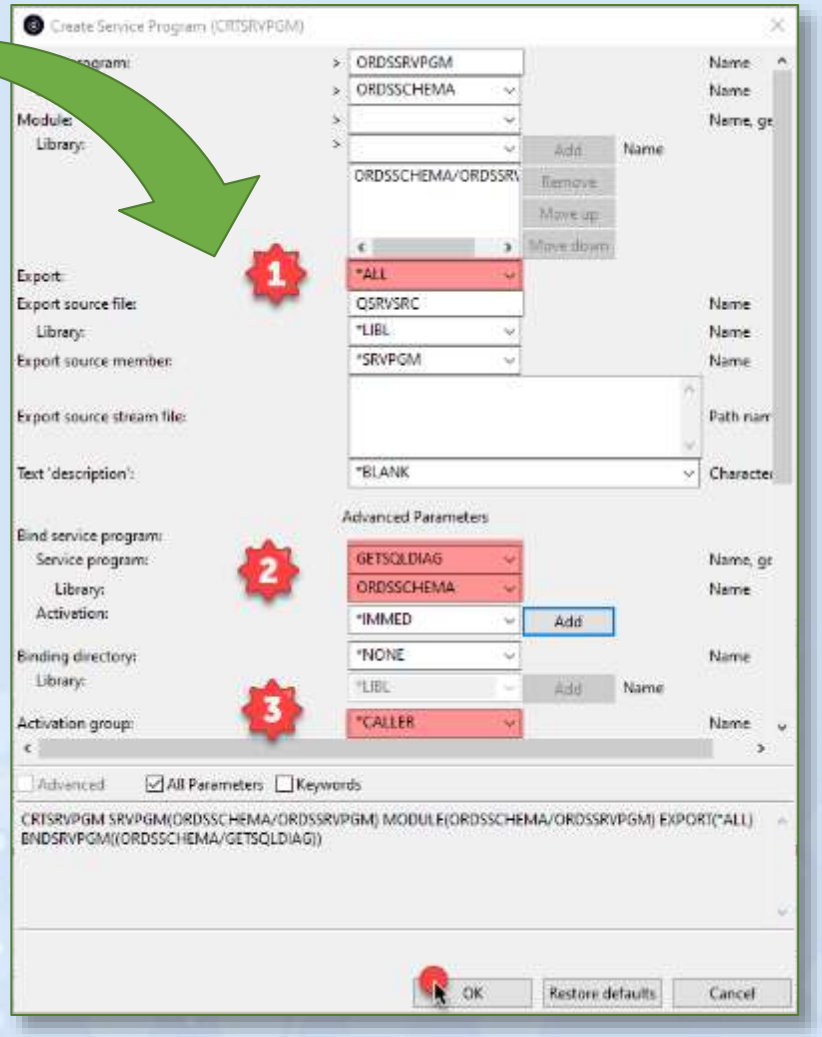


Create ORDSSRVPGM Module

Right-click the program source member
1.Select the Compile (Prompt) > CRTSQLRPGI
2.Change Compile type to *Module
3.Click-on Ok
Always check that the compile worked under commands log view

Create ORDSSRVPGM Service Program





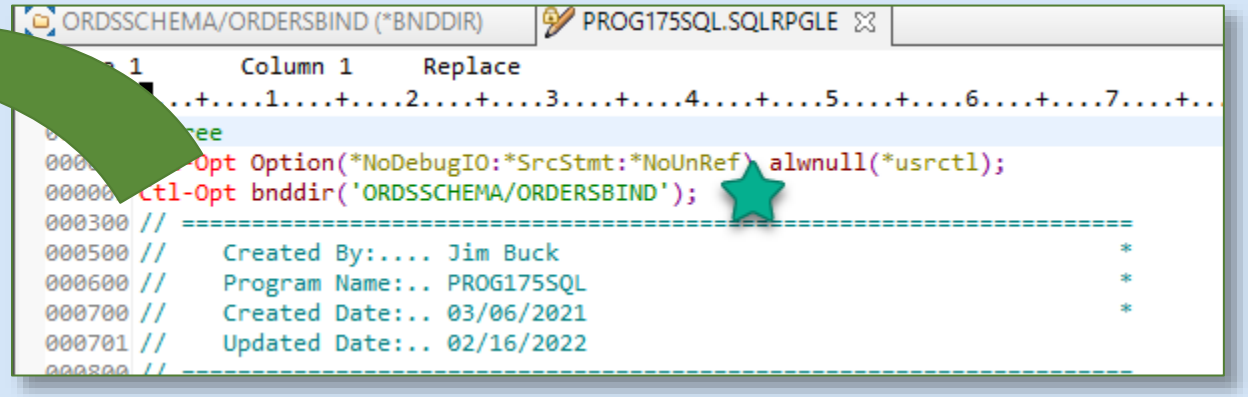
Create ORDSSRVPGM Service Program

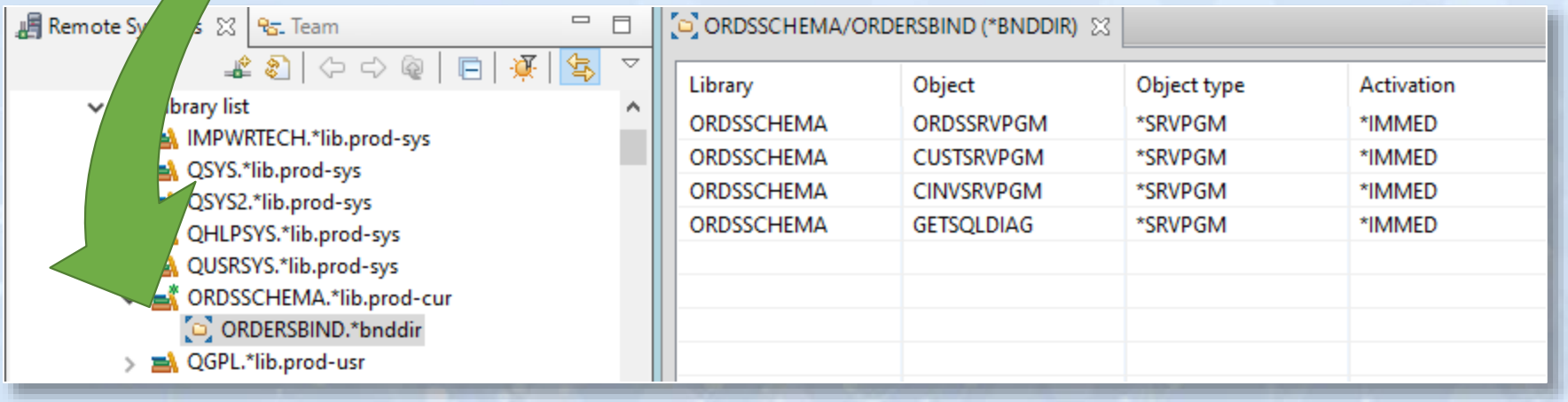
Right-click the program module THEN Select Create -> Service program
1.Change export to *ALL
2.Add the **YourLib**/GETSQLDIAG Service program
3.Use the *CALLER to the ACTGRP parameter
Click “Ok”

Create A RUNNABLE Application

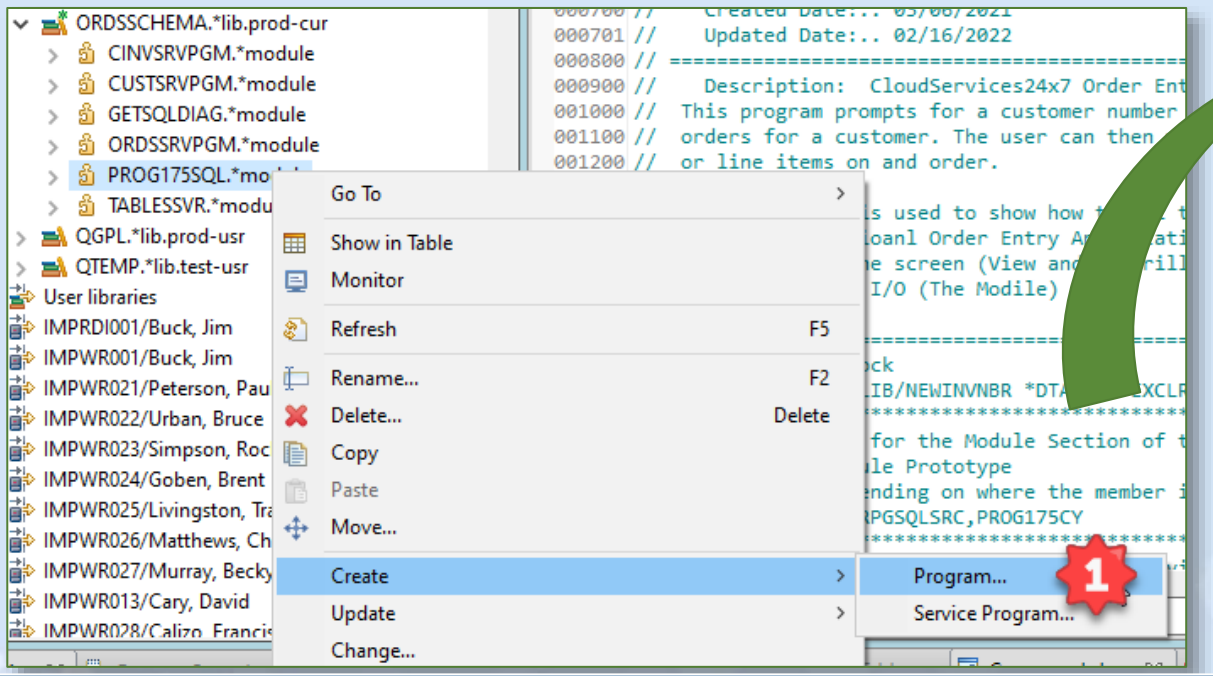
Use a Binding Directory

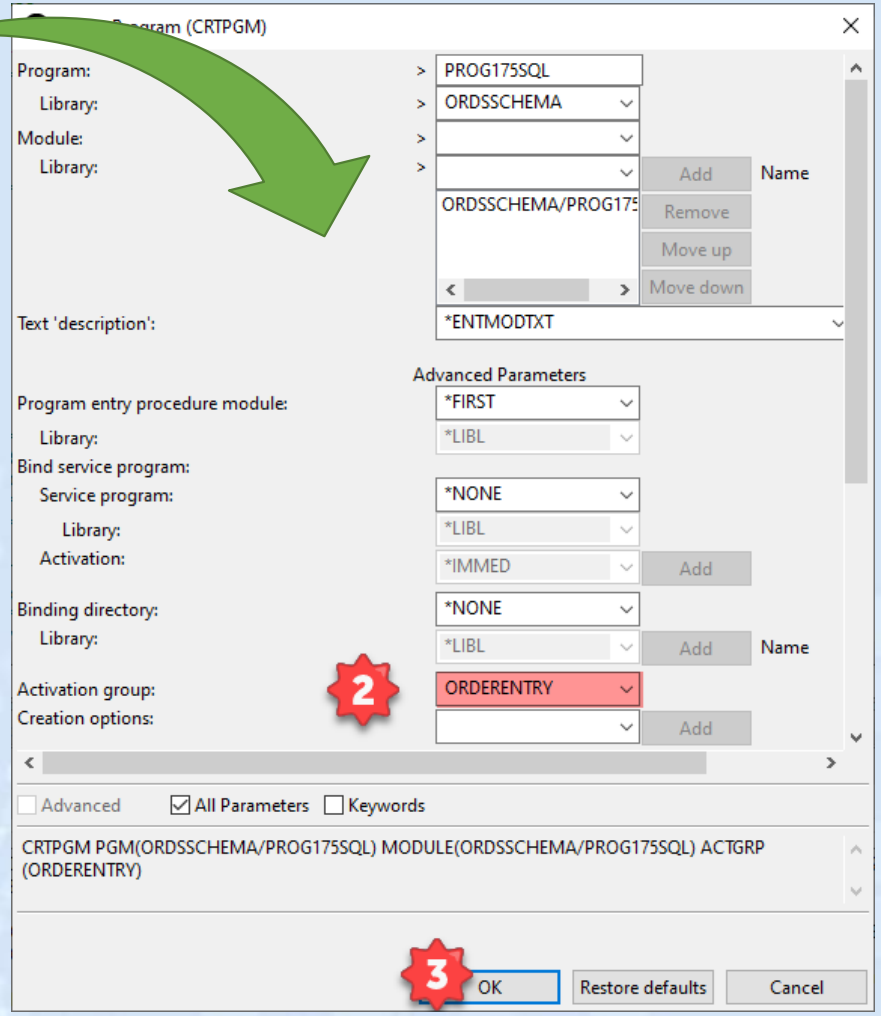
1. Expedites using Service Programs
2. Contains service programs with the required procedures for the Order Entry Application





Create A RUNNABLE Application





Create PROG175SQL Program

1. Right-click the program module THEN Select Create -> Program
2. Change ACTGRP to ORDERENTRY
3. Click “Ok”

Complete online Classes
Based on my textbooks
Includes numerous Videos,
Quizzes, Sample programs & and program assignments

imPOWER TECHNOLOGIES
Powered by Jim Buck

Jim Buck
Phone 262-705-2832
jbuck@impowertechnologies.com
Twitter - @jbuck_imPower
www.impowertechnologies.com

Any Questions or Comments?