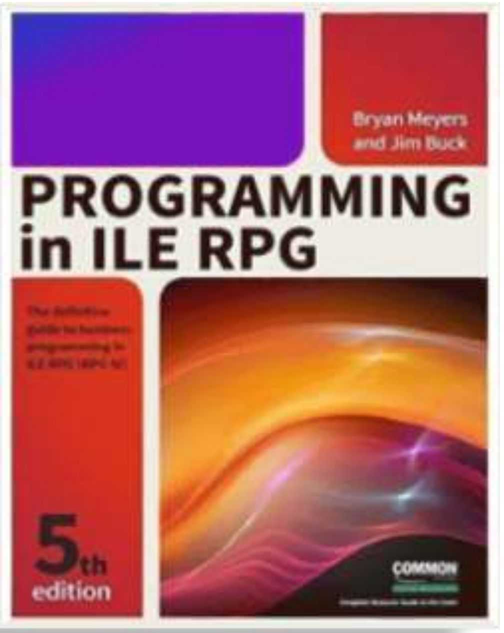
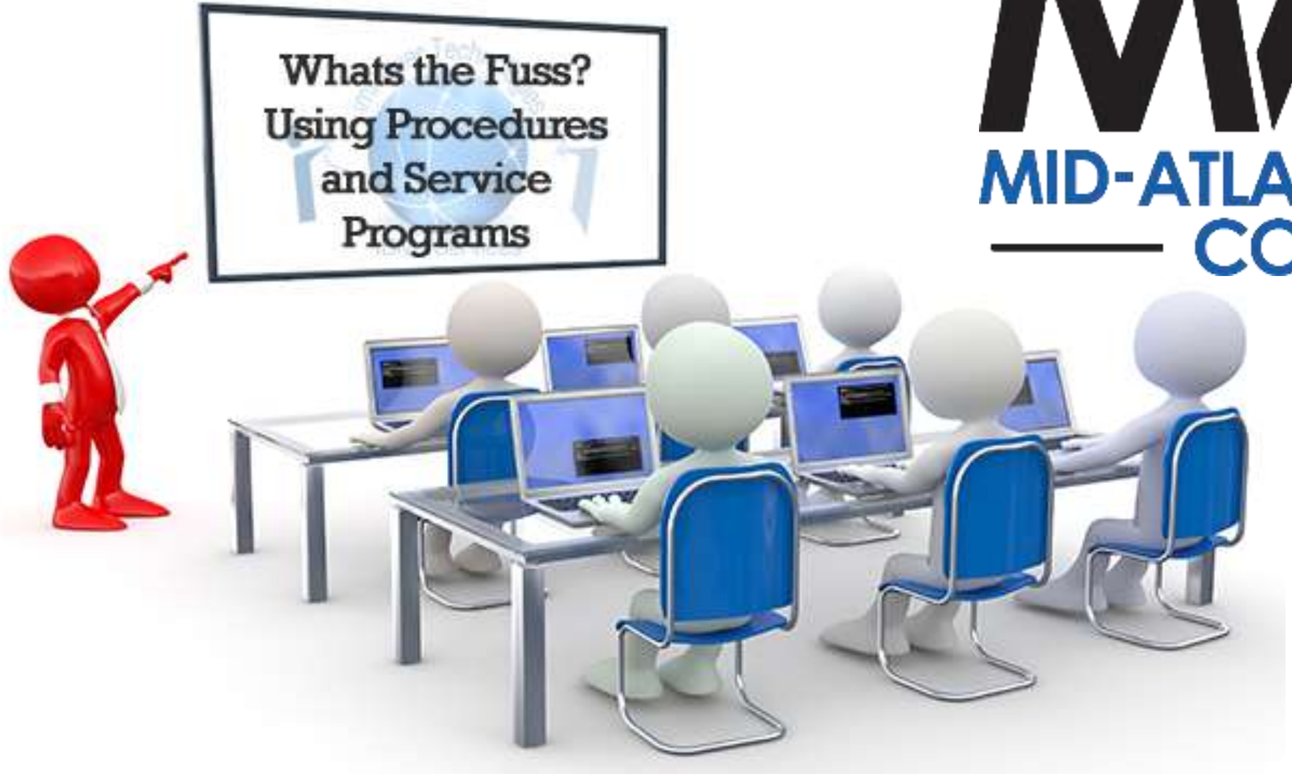




MAGiC
MID-ATLANTIC GROUP OF IBM i COLLABORATORS

Jim Buck
Phone 262-705-2832
Email – 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

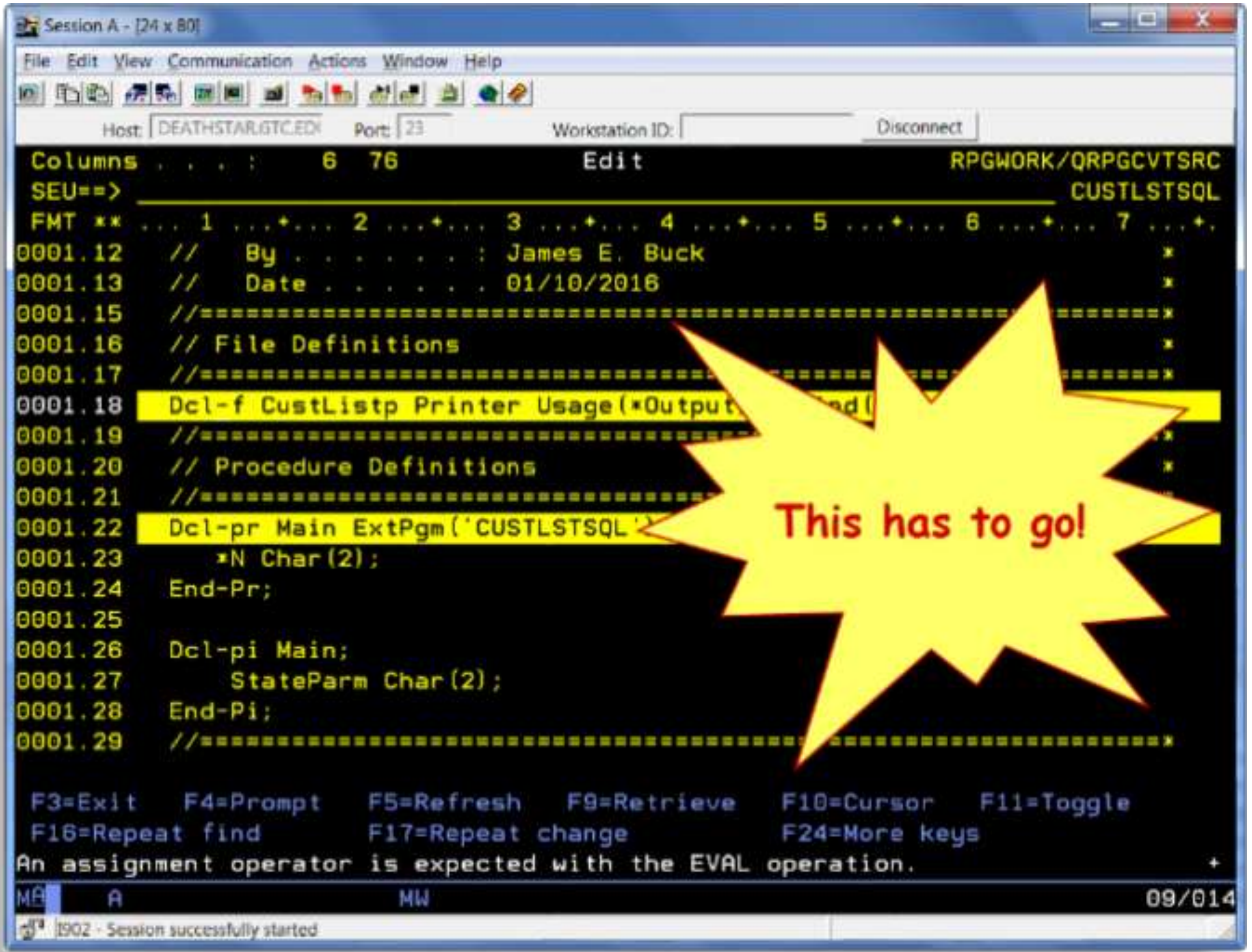
Onsite IBM i Classes:

- Topics include the latest RDi/RPG/SQL techniques
- 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

5250 & SEU – Doesn't work anymore!



```
 Columns . . . :    6  76            Edit                 RPGWORK/QRPGCVTSRC
 SEU==>                                                            CUSTLSTSQL
 FMT **  ... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+.
0001.12     //    By . . . . . . . : James E. Buck                          *
0001.13     //    Date . . . . . . . 01/10/2016                             *
0001.15     //===================================================================*
0001.16     // File Definitions                                              *
0001.17     //===================================================================*
0001.18     Dcl-f CustListp Printer Usage(*Output     nd(
0001.19     //===================================================================*
0001.20     // Procedure Definitions                                         *
0001.21     //===================================================================*
0001.22     Dcl-pr Main ExtPgm('CUSTLSTSQL')
0001.23        *N Char(2);
0001.24     End-Pr;
0001.25
0001.26     Dcl-pi Main;
0001.27        StateParm Char(2);
0001.28     End-Pi;
0001.29     //===================================================================*

 F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
 F16=Repeat find       F17=Repeat change          F24=More keys
An assignment operator is expected with the EVAL operation.                +
MA     A                   MW                                          09/014
```

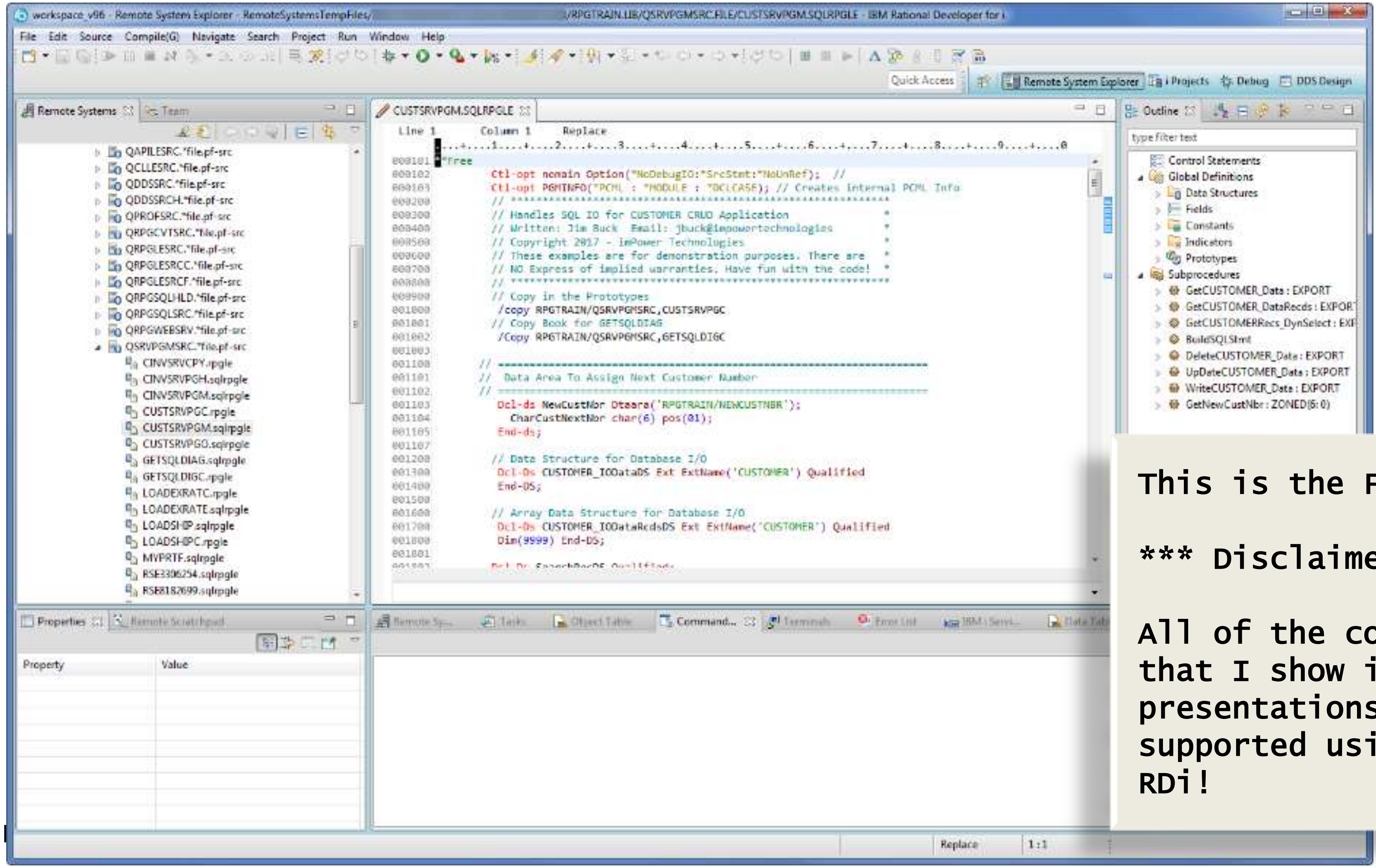
I902 - Session successfully started

SEU doesn't support the latest version of RPG.

Well I guess, you could turnoff Syntax Checking!

My students have a short introduction… in case of emergencies!

Rational Developer for i – 9.6.0.6



```
**Free
        Ctl-opt nomain Option(*NoDebugIO:*SrcStmt:*NoUnRef);  //
        Ctl-opt PGMINFO(*PCML : *MODULE : *DCLCASE); // Creates internal PCML Info
        // ***************************************************************
        // Handles SQL IO for CUSTOMER CRUD Application                 *
        // Written: Jim Buck  Email: jbuck@impowertechnologies          *
        // Copyright 2017 - imPower Technologies                        *
        // These examples are for demonstration purposes. There are     *
        // NO Express of implied warranties. Have fun with the code!    *
        // ***************************************************************
        // Copy in the Prototypes
         /copy RPGTRAIN/QSRVPGMSRC,CUSTSRVPGC
        // Copy Book for GETSQLDIAG
         /Copy RPGTRAIN/QSRVPGMSRC,GETSQLDIGC

      // ------------------------------------------------------------------
      //  Data Area To Assign Next Customer Number
      // ==================================================================
         Dcl-ds NewCustNbr Dtaara('RPGTRAIN/NEWCUSTNBR');
           CharCustNextNbr char(6) pos(01);
         End-ds;

        // Data Structure for Database I/O
         Dcl-Ds CUSTOMER_IODataDS Ext ExtName('CUSTOMER') Qualified
         End-DS;

        // Array Data Structure for Database I/O
         Dcl-Ds CUSTOMER_IODataRcdsDS Ext ExtName('CUSTOMER') Qualified
         Dim(9999) End-DS;

        Dcl-Ds SearchRecDS Qualified;
```

This is the FUTURE!

*** Disclaimer ***

All of the code that I show in my presentations is supported using RDi!

Modular Programming

Developing code in small, independent units offers several advantages

- Reusability
- Fewer errors
- Easier to test
- Changes are unlikely to cause unwanted side effects
- Easier team development

Callp (Call a Prototyped Procedure or Program) operation

- Passes control to separate *PGM object
- When called program returns, caller resumes

Prototyping the Call Interface

Variable's scope is limited to program in which it is defined

Callp operation passes parameters to communicate values between caller and called program

- Parameters are values that one program can pass to another
- Called program can then accept parameters and use those values to control its processing

Prototype definition defines call interface

- Name of program to call
- Number of parameters, data attributes, order

Compiler uses prototype to call program correctly and to ensure caller passes correct parameters

Prototyping the Call Interface

Prototype definition has two parts

- Prototype header
- Parameter descriptions

Dcl-pr (Declare Prototype) instruction signals beginning of prototyped call interface

- Coded in Declarations section
- End-pr instruction ends prototype

```
Dcl-pr Updcust Extpgm('AR002');
  Company  Char(5);
  Customer Zoned(7:0);
End-pr;
```

Prototyping the Call Interface

Subsequent lines describe parameters

- Similar to data structure subfields
- May list up to 255 parameters

Parameters need not be named

- Variable must be declared elsewhere
- May use *N placeholder instead
- If name matches RPG reserved word, Dcl-parm (Declare Parameter) instruction is required
  - Optional otherwise

```
Dcl-pr Updcust Extpgm('AR002');
  Company  Char(5);
  Customer Zoned(7:0);
End-pr;
```

```
Dcl-pr Updcust Extpgm('AR002');
  *N Char(5);
  *N Zoned(7:0);
End-pr;
```

```
Dcl-pr Updcust Extpgm('AR002');
  Dcl-parm Company  Char(5);
  Dcl-parm Customer Zoned(7:0);
End-pr;
```

Prototyping the Call Interface

If there are no parameters to pass, prototype may be coded on single line

```
Dcl-pr Sleep Extpgm End-pr;
```

Callp (Call Prototyped Procedure or Program)

Callp (Call Prototyped Procedure or Program) operation invokes associated *PGM object, then passes control to it

- Actually coding Callp is optional

```
Dcl-pr Updcust Extpgm('AR002');
  *N Char(5);
  *N Zoned(7:0);
End-pr;
…
Callp Updcust(Company:Custnbr);
```

Parameters in parentheses immediately follow prototype name

- If no parameters, code empty parentheses instead

```
Updcust(Company:Custnbr);
```

```
Dcl-pr Sleep Extpgm End-pr;
…
Sleep();
```

Procedure Interface – Called Program

Called RPG program should also use prototype to describe list of parameters that it is to receive from caller

- Main procedure prototype

Prototypes in each program must match each other

- Same number of parameters, data attributes
- Can Should use /COPY to keep them in sync!
- Names need not be same

Called program must include procedure interface definition

- In addition to prototype
- Defines variable names to hold received parameter values

Procedure Interface

Dcl-pi (Declare Procedure Interface) instruction defines main procedure interface

- Similar to prototype definition
- Must appear after main procedure prototype

Two parts

- Procedure interface header
- Parameter definitions to hold received parameters

```
Dcl-pr AR002 Extpgm;
  *N        Char(5);
  *N        Zoned(7:0);
End-pr;

Dcl-pi AR002;
  Company  Char(5);
  Customer Zoned(7:0);
End-pi;
```

Passing Parameters by Reference

Default parameter passing method

RPG passes parameter by passing address of memory location represented by variable

- Called program uses that storage address to retrieve and process the parameter value

Both programs share same storage

- If called program changes parameter value,
- Caller recognizes change when it regains control

Passing Parameters by Reference

```
// Caller
Dcl-pr Nextpgm Extpgm;
  *N Ind;
End-pr;

Dcl-s Okay Ind Inz(*Off);
…
             // Okay = *Off before call
Nextpgm(Okay);
             // Okay = *On after call
…
```

```
// Nextpgm
Dcl-pr Nextpgm Extpgm;
  *N Ind;
End-pr;

Dcl-pi Nextpgm;
  Flag *Ind;
End-pi;
…
Flag = *On;      // Change value of Flag
*Inlr = *On;
Return;
```

Passing Parameters by Read-only Reference

Alternative method of passing parameters

- **Const keyword** on prototype (and interface) specifies read-only referece

Several advantages over passing by reference

- System offers protection against called program changing parameter values
- Parameter values need not be represented by variable
- Parameter data types need not precisely match prototype

```
Dcl-pr Addcust Extpgm('AR001');
  *N Char(9)     Const;
  *N Zoned(7:0)  Const;
End-pr;
…
Addcust('BBY' : Lastcustomer + 1);
…
```

Passing Parameters by Read-only Reference

Callp can pass variables, literals, or expressions

Caller can first evaluate expression or literal, and then make temporary copy of value before invoking called program

Caller passes storage address of temporary copy

```
Dcl-pr Addcust Extpgm('AR001');
  *N Char(9)     Const;
  *N Zoned(7:0) Const;
End-pr;
…
Addcust('BBY' : Lastcustomer + 1);
…
```

Passing Parameters by Read-only Reference

```
Dcl-pr Addcust Extpgm('AR001');
  *N Char(9)     Const;
  *N Zoned(7:0)  Const;
End-pr;
…
Addcust('BBY' : Lastcustomer + 1);
…
```

```
// Called program (AR001)
…
Dcl-pr Addcust Extpgm('AR001');
  *N Char(9)     Const;
  *N Zoned(7:0)  Const;
End-pr;

Dcl-pi Addcust;
  Company  Char(5)     Const;
  Customer Zoned(7:0)  Const;
End-pi;
…
  // (Some processing goes here...
  // cannot change Company or Customer)
…
*Inlr = *On;
Return;
```

Choosing the Parameter Passing Method

If caller needs to access changes made by called program, pass by reference

- Or if large number of parameters must be passed
  - Improves the program “Call” performance

Use read-only reference as preferred method for passing parameters between programs

- Improves coding flexibility (variables, literals, or expressions)
- Protects integrity of caller’s data

Fitting the Pieces

Caller includes two items

- *Prototype definition* to describe parameters to pass
- Callp operation to execute call

Called program contains three items

- Main *procedure prototype definition* to describe parameters to receive
- Main *procedure interface* to accept parameters and place them into variables
- Return operation to return control to caller
  - Generally, will first set *Inlr *On

Dynamic Program Calls and Static Binding

*PGM to *PGM calls are dynamic calls

- Association between caller and called program is resolved at runtime
- Programs are never physically connected

Integrated Language Environment enables static calls

- Association between modules is resolved before runtime
  - When *PGM is created
- No need for resolution at runtime
- Procedures are key component of static binding

Introduction to Procedures

A **Procedure** is self-contained, identifiable collection of RPG statements within *PGM object

- Performs specific task and returns to caller
- Not a system object

Modern RPG Program is comprised **four sections**

- **Control options**
  - Provides default options for program
- **Declarations**
  - Identifies and defines files, variables, and other data items
- **Main procedure**
  - Processes, calculations, and procedures
- **Subprocedures**
  - Declarations and processes for optional distinct program functions that main procedure or other subprocedures can execute once or many times

Role of Procedures in a Program

Like a subroutine, but with more capabilities

Subprocedure can be created independently from program

- Code in separate source member and compile into separate module, then bind it when you create program

Subprocedure enables flexible variable scoping

- Global declarations make all variables equally accessible by entire program
  - Only option without subprocedures
- Local variables are recognized only within procedure in which they are defined
  - Data values are communicated between procedures by passing parameters

Subprocedure supports return value

- User-defined function

Subprocedures can be recursive

- Can call itself

Coding a Procedure

Dcl-proc (Declare Procedure) instruction begins procedure

- End-proc instruction ends it

Subprocedure code can include

- Procedure interface
- Declarations
- Processing statements (calculations)

```
// ---------------------------------------------
// Procedure Celsius =
                Converts Fahrenheit to Celsius
// ---------------------------------------------

   Dcl-proc Celsius;

// ------------------------- Procedure interface
   Dcl-pi *N  Zoned(5:2);
      Fahrenheit Zoned(5:2);
   End-pi;

// ----------------------------- Local variables
   Dcl-s Temperature  Zoned(5:2);

   Eval(H) Temperature = (5/9)
                       * (Fahrenheit - 32);
   Return Temperature;

   End-proc Celsius;
```

Coding a Procedure

Need not name subprocedure interface

- Use *N placeholder instead

Dcl-proc instruction includes data attributes of return value

- Can be value of any supported data type
- Can also be data structure (using Likeds) or array (using Dim)

```
// ----------------------------------------
// Procedure Celsius =
             Converts Fahrenheit to Celsius
// ----------------------------------------

   Dcl-proc Celsius;

// ------------------ Procedure interface
   Dcl-pi *N  Zoned(5:2);
      Fahrenheit Zoned(5:2);
   End-pi;

// ---------------------- Local variables
   Dcl-s Temperature  Zoned(5:2);

   Eval(H) Temperature = (5/9)
                       * (Fahrenheit - 32);
   Return Temperature;

   End-proc Celsius;
```

Coding a Procedure

Declarations within subprocedure have local scope

- Including variables in procedure interface
- Restricted to subprocedure
- No other procedure will recognize local variables

Return operation includes value to return to caller

- May be variable, literal, or expression

```
// ---------------------------------------------
// Procedure Celsius =
                Converts Fahrenheit to Celsius
// ---------------------------------------------

   Dcl-proc Celsius;

// ------------------------- Procedure interface
   Dcl-pi *N  Zoned(5:2);
      Fahrenheit Zoned(5:2);
   End-pi;

// ----------------------------- Local variables
   Dcl-s Temperature  Zoned(5:2);

   Eval(H) Temperature = (5/9)
                       * (Fahrenheit - 32);
   Return Temperature;

   End-proc Celsius;
```

Coding a Procedure

Subprocedure prototype is coded in global section of source

Prototype names procedure

- If procedure name is different from prototype name, use **Extproc** keyword

Prototype includes return value

```
// ---------------------- Global Area of Pgm
  Dcl-pr Celsius Zoned(5:2);
     *N Zoned(5:2);
  End-pr;
// -----------------------------------------
// Procedure Celsius =
        Converts Fahrenheit to Celsius
// -----------------------------------------

   Dcl-proc Celsius;

// ---------------------- Procedure interface
   Dcl-pi *N  Zoned(5:2);
      Fahrenheit Zoned(5:2);
   End-pi;

// -------------------------- Local variables
   Dcl-s Temperature  Zoned(5:2);

   Eval(H) Temperature = (5/9)
                     * (Fahrenheit - 32);
   Return Temperature;

   End-proc Celsius;
```

Coding a Procedure

Locally scoped variables are allocated in automatic storage

- Reinitialized each time procedure executes
- Do not retain value between iterations

Static keyword uses static storage instead

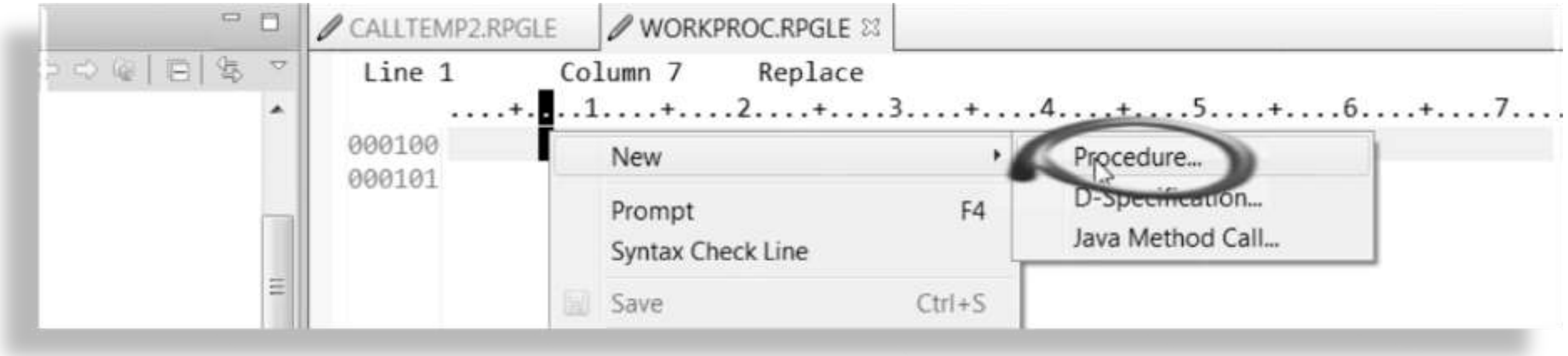
- Retains value

```
Dcl-s Counter Uns(5) Static;
```

RDi and creating PR and PI’s

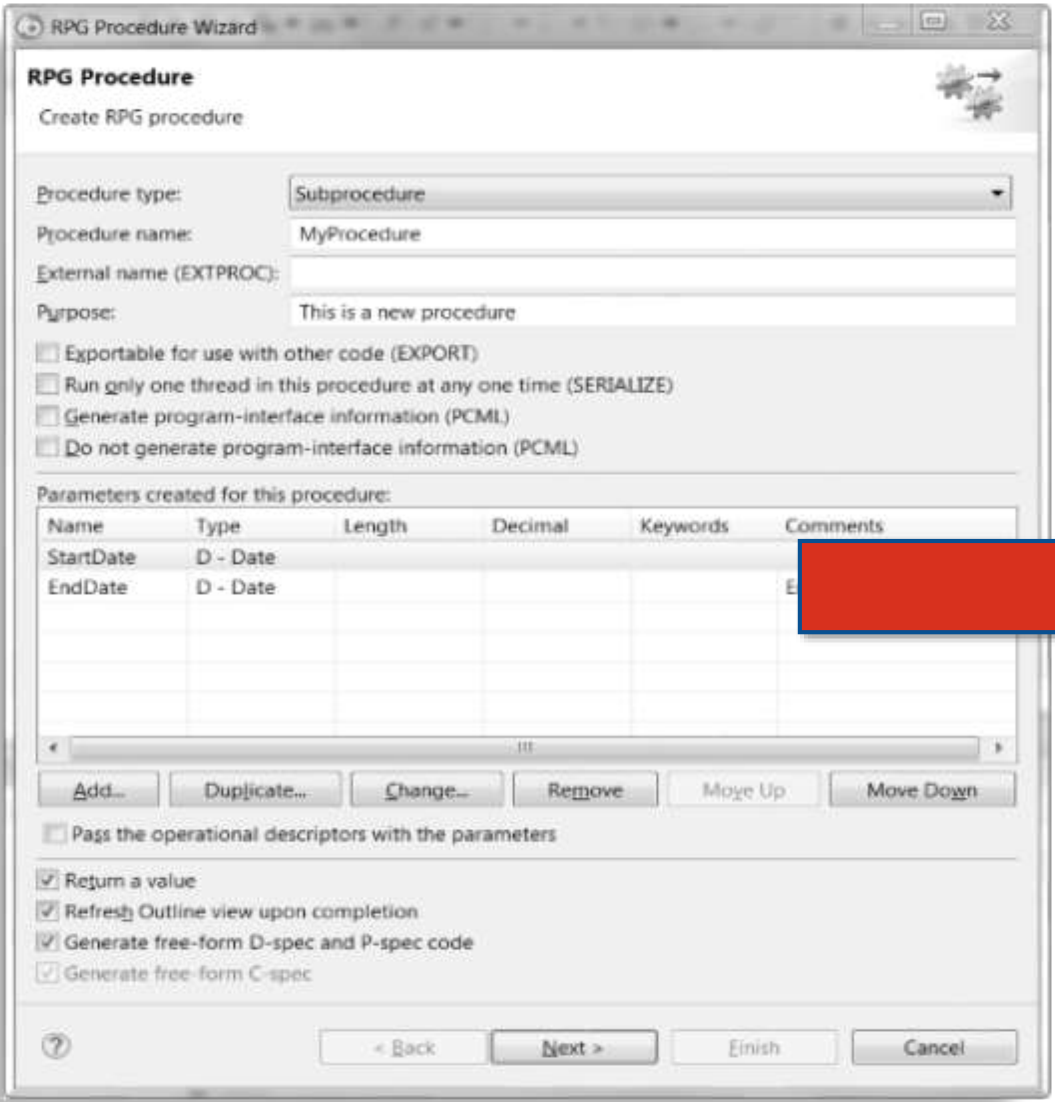
RDi has some great features to help you get started with Prototypes and Interfaces.

Most of the problems are caused by a “Piece(s)” missing.



```
Line 1        Column 7      Replace
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....
000100
000101
```

Creating PR and PI's



RPG Procedure Wizard

RPG Procedure

Create RPG procedure

Procedure type:	Subprocedure
Procedure name:	MyProcedure
External name (EXTPROC):	
Purpose:	This is a new procedure

- [ ] Exportable for use with other code (EXPORT)
- [ ] Run only one thread in this procedure at any one time (SERIALIZE)
- [ ] Generate program-interface information (PCML)
- [ ] Do not generate program-interface information (PCML)

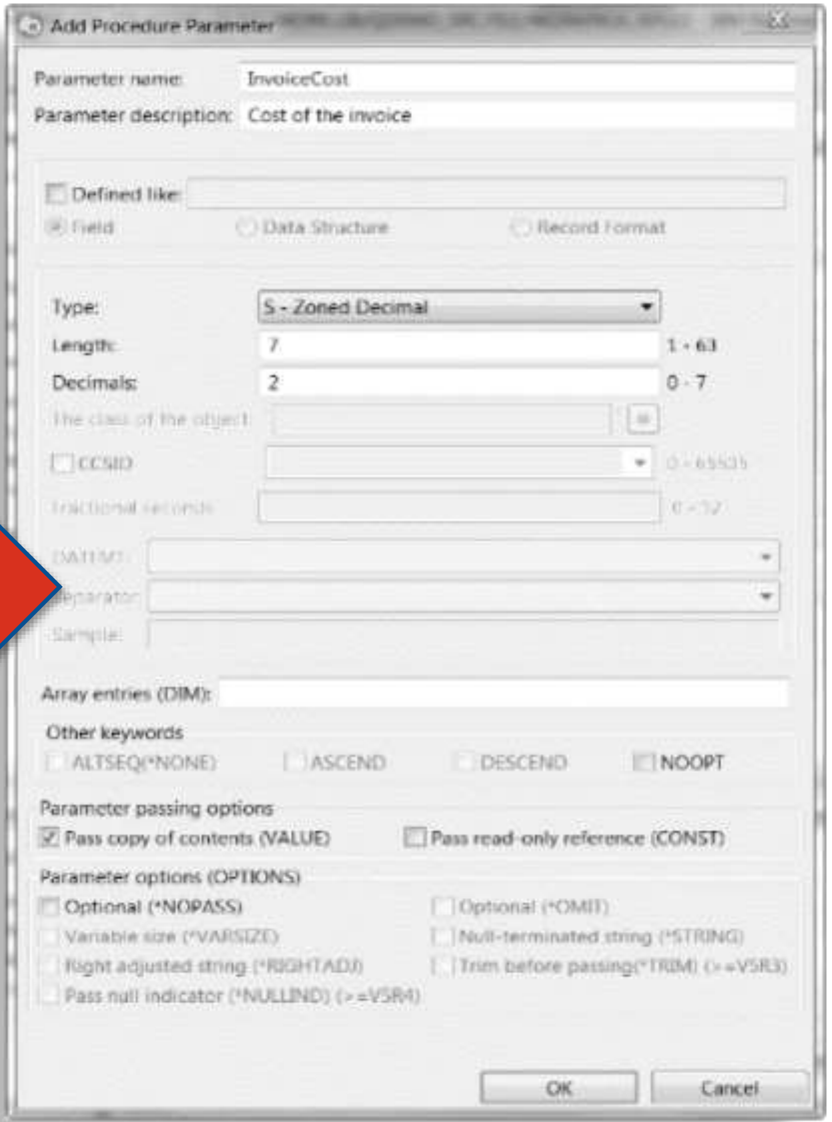
Parameters created for this procedure:

Name	Type	Length	Decimal	Keywords	Comments
StartDate	D - Date				
EndDate	D - Date				E

Add... | Duplicate... | Change... | Remove | Move Up | Move Down

- [ ] Pass the operational descriptors with the parameters
- [x] Return a value
- [x] Refresh Outline view upon completion
- [x] Generate free-form D-spec and P-spec code
- [x] Generate free-form C-spec

< Back | Next > | Finish | Cancel



Add Procedure Parameter

Parameter name:	InvoiceCost
Parameter description:	Cost of the invoice

- [ ] Defined like:

(●) Field ( ) Data Structure ( ) Record Format

Type:	S - Zoned Decimal	
Length:	7	1 - 63
Decimals:	2	0 - 7
The class of the object:		
CCSID		0 - 65535
Fractional seconds:		0 - 12
DATFMT:		
Separator:		
Sample:		

Array entries (DIM):

Other keywords

- [ ] ALTSEQ(*NONE)
- [ ] ASCEND
- [ ] DESCEND
- [ ] NOOPT

Parameter passing options

- [x] Pass copy of contents (VALUE)
- [ ] Pass read-only reference (CONST)

Parameter options (OPTIONS)

- [ ] Optional (*NOPASS)
- [ ] Optional (*OMIT)
- [ ] Variable size (*VARSIZE)
- [ ] Null-terminated string (*STRING)
- [ ] Right adjusted string (*RIGHTADJ)
- [ ] Trim before passing(*TRIM) (>=V5R3)
- [ ] Pass null indicator (*NULLIND) (>=V5R4)

OK | Cancel

Executing a Procedure

Callp operation calls procedure without return value

- Or if program ignores return value

Function-like call uses return value

```
Callp Updcust(Company:Custnbr);
```

```
Metrictemp = Celsius(Englishtemp);
```

```
If Celsius(Englishtemp) > 100;
…
Endif;
```

Cycle Main Programs

Most traditional RPG programs are cycle main programs

Cycle main program has main procedure implicitly specified

- Main procedure is not named
  - Location in the program code designates main procedure
- Includes main source section (the main procedure), followed by zero or more subprocedure sections
  - Main procedure is everything before **first Dcl-proc** instruction
  - May include local subprocedures coded following main procedure

Compiler automatically includes RPG cycle to provide program initialization, termination, file input and output (I/O)

Sample Cycle Main Program

```
// Control Options ==================================*
Ctl-Opt Option(*NoDebugIO) DftActGrp(*No);

Dcl-pr CallTemp ExtPgm;
   *N char(5);
End-pr;

Dcl-pi CallTemp;
   CharTemp char(5);
End-pi;

// ----------------------------------------- Prototypes
Dcl-pr Celsius Zoned(5:2);
   *N Zoned(5:2);
End-pr;
```

(Continued…)

```
// ------------------------------ Global variables
Dcl-s Message      Char(52);
Dcl-s Metrictemp   Zoned(5:2);
Dcl-s State        Varchar(8);
Dcl-s EnglishTemp  Zoned(5:2);
// ------------------------------- Main procedure

EnglishTemp = %Dec(CharTemp:5:2);
Metrictemp = Celsius(Englishtemp);

Select;
  When Metrictemp < 0;
    State = 'solid';
  When Metrictemp = 0;
    State = 'freezing';
  When Metrictemp = 100;
    State = 'boiling';
  When Metrictemp > 100;
    State = 'gaseous';
  Other;
    State = 'liquid';
Endsl;
```

(Continued…)

```
Message = 'At ' + %Char(Englishtemp) + ' degrees (' +
             %Char(Metrictemp) + ' Celsius), water is ' +
                State + '.';
Dsply Message '*REQUESTER';
*Inlr = *On;
Return;

// ---------------------------------------------------
// Procedure  Celsius  Converts Fahrenheit to Celsius
// ---------------------------------------------------
Dcl-proc Celsius;

// ------------------------------ Procedure interface
  Dcl-pi *N  Zoned(5:2);
     Fahrenheit Zoned(5:2);
  End-pi;

// ---------------------------------- Local variables
  Dcl-s Temperature  Zoned(5:2);

  Eval(H) Temperature = (5/9) * (Fahrenheit - 32);
  Return Temperature;

End-proc Celsius;
```

Linear Main Programs

Linear main program explicitly names main procedure

- Ctl-opt instruction uses Main keyword to name main procedure

Global section in linear main program does not contain any executable code

- Before first **Dcl-proc**
- Can include global declarations

Compiler will not embed RPG cycle into linear main program

Program implicitly initializes variables, locks data areas, and opens files when program starts

Program does not use *Inlr to trigger automatic shutdown

- Resources are not cleaned up or closed when program ends, unless program explicitly does so

Program is recursive

Sample Linear Main Program

```
Ctl-Opt Option(*NoDebugIO) DftActGrp(*No);
Ctl-opt Main(Driver);

// Control Options ==================================*
Dcl-pr Driver ExtPgm('CALLTEMP1');
   *N char(5);
End-pr;

// ----------------------------------------- Prototypes
Dcl-pr Celsius Zoned(5:2);
   *N Zoned(5:2);
End-pr;
// ----------------------------------------------------
// Main procedure
//
Dcl-proc Driver;
------------------------------ Procedure interface
Dcl-pi *N;
   CharTemp char(5);
End-pi;
```

(Continued…)

Sample Linear Main Program

```
// ---------------------------------- Local variables
Dcl-s Message    Char(52);
Dcl-s Metrictemp Zoned(5:2);
Dcl-s State      Varchar(8);
Dcl-s EnglishTemp Zoned(5:2);
// ------------------------------------- Main procedure

EnglishTemp = %Dec(charTemp:5:2);
Metrictemp = Celsius(Englishtemp);

Select;
When Metrictemp < 0;
  State = 'solid';
When Metrictemp = 0;
  State = 'freezing';
When Metrictemp = 100;
  State = 'boiling';
When Metrictemp > 100;
  State = 'gaseous';
Other;
  State = 'liquid';
Endsl;
```

(Continued…)

Sample Linear Main Program

```
Message = 'At ' + %Char(Englishtemp) + ' degrees (' +
              %Char(Metrictemp) + ' Celsius), water is ' +
                  State + '.';
Dsply Message '*REQUESTER';

Return;
End-proc Driver;

// ---------------------------------------------------
// Procedure Celsius = Converts Fahrenheit to Celsius
// ---------------------------------------------------
Dcl-proc Celsius;
// ------------------------------ Procedure interface
Dcl-pi *N  Zoned(5:2);
   Fahrenheit Zoned(5:2);
End-pi;

// ---------------------------------- Local variables
Dcl-s Temperature  Zoned(5:2);
Eval(H) Temperature = (5/9) * (Fahrenheit - 32);
Return Temperature;

End-proc Celsius;
```

Nomain Modules

Nomain modules consists of program segments without no main procedure

Source consists only of global declarations and subprocedures	Can combine with other modules to create program	One module in program must have main procedure

Includes Ctl-opt instruction with Nomain keyword

Enhance code reusability

Can help enforce business rules and practices by centralizing application functions

Eliminate redundant code

Improve maintainability, reliability

Nomain Modules

Dcl-proc “Export” keyword allows procedure to be called from outside module

- Other modules in program can execute procedure, even though they don’t contain the code for the procedure
- Without Export, procedure can only be executed from within module
- Most procedures in Nomain modules include Export keyword

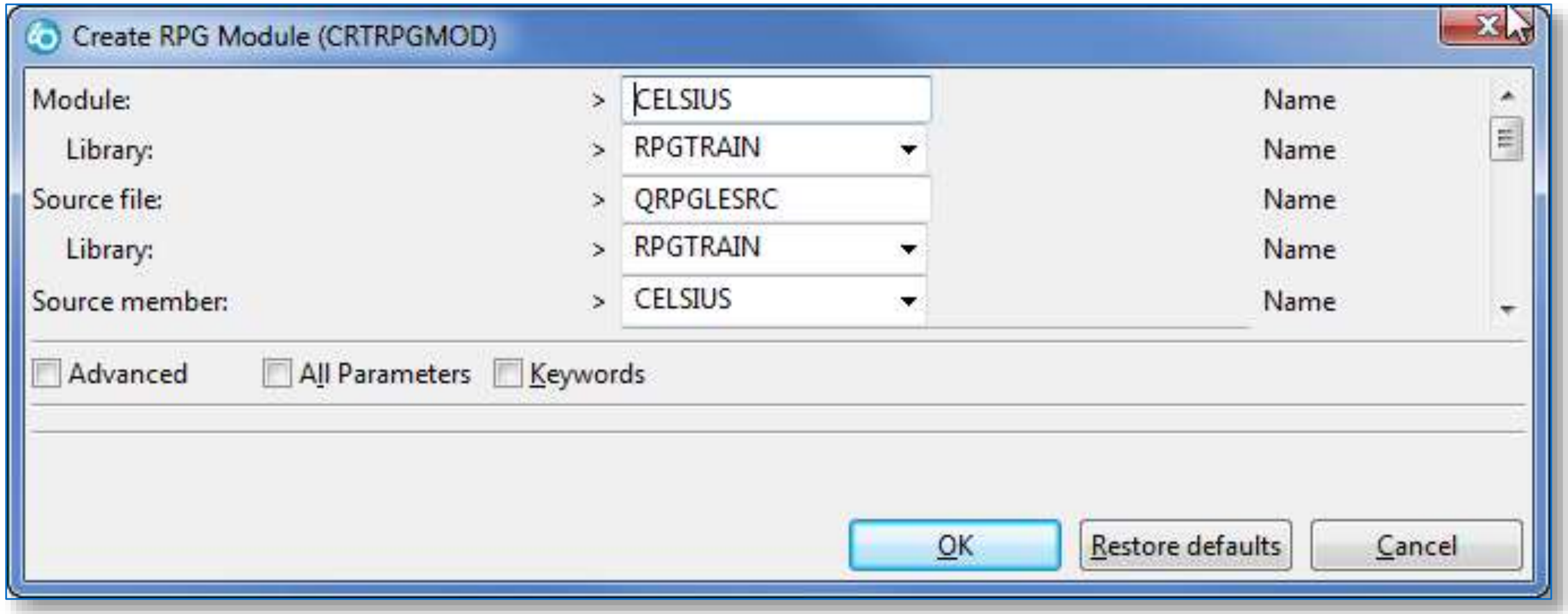
CELSIUS - Nomain Module

```
Ctl-opt Nomain;
// Copy Block – Replaces including the Prototype

   /Copy RPGTRAIN/QRPGLESRC,CALLTMPCPY

Dcl-proc Celsius Export;

// ------------------------------ Procedure interface
Dcl-pi *N  Zoned(5:2);
   Fahrenheit Zoned(5:2);
End-pi;

// ---------------------------------- Local variables
Dcl-s Temperature  Zoned(5:2);

Eval(H) Temperature = (5/9) * (Fahrenheit - 32);
Return Temperature;

End-proc Celsius;
```

Creating Modular Programs

CRTRPGMOD (Create RPG Module) command compiles source member, creates *MODULE object

- Module contains compiled, executable code
- Cannot run module by itself
- Module is interim building block for eventual program object


Create RPG Module (CRTRPGMOD)

Module:	>	CELSIUS	Name
Library:	>	RPGTRAIN	Name
Source file:	>	QRPGLESRC	Name
Library:	>	RPGTRAIN	Name
Source member:	>	CELSIUS	Name

Using /COPY, /INCLUDE

/Copy and /Include functions tell compiler to include source records from another source member (Copybook)

- Can reuse single copy of code without having to retype
- Useful for storing prototypes and other reusable code snippets

```
/Copy Mylib/Mysource,Prototypes
```

Copybook member

Copybook source file

```
/Include Mylib/Mysource,Prototypes
```

Using /COPY, /INCLUDE

Both directives use the same syntax and have the same purpose

- /copy is expanded by the SQL precompiler
- /include is ignored by the SQL precompiler

```
/Copy Mylib/Mysource,Prototypes
```

Copybook member

Copybook source file

```
/Include Mylib/Mysource,Prototypes
```

CALLTEMP2 - Linear Main Program

```
// Control Options ===================================*
      Ctl-opt Main(Driver) Option(*NoDebugIO);

// Copy Block
      /Copy RPGTRAIN/QRPGLESRC,CALLTMPCPY
// --------------------------------------------------
// Main procedure
// --------------------------------------------------
Dcl-proc Driver;
// ------------------------------ Procedure interface
Dcl-pi *N;
   CharTemp char(5);
End-pi;
// ----------------------------------- Global variables
   Dcl-s Message     Char(52);
   Dcl-s Metrictemp  Zoned(5:2);
   Dcl-s State       Varchar(8);
   Dcl-s EnglishTemp Zoned(5:2);
```

(Continued...)

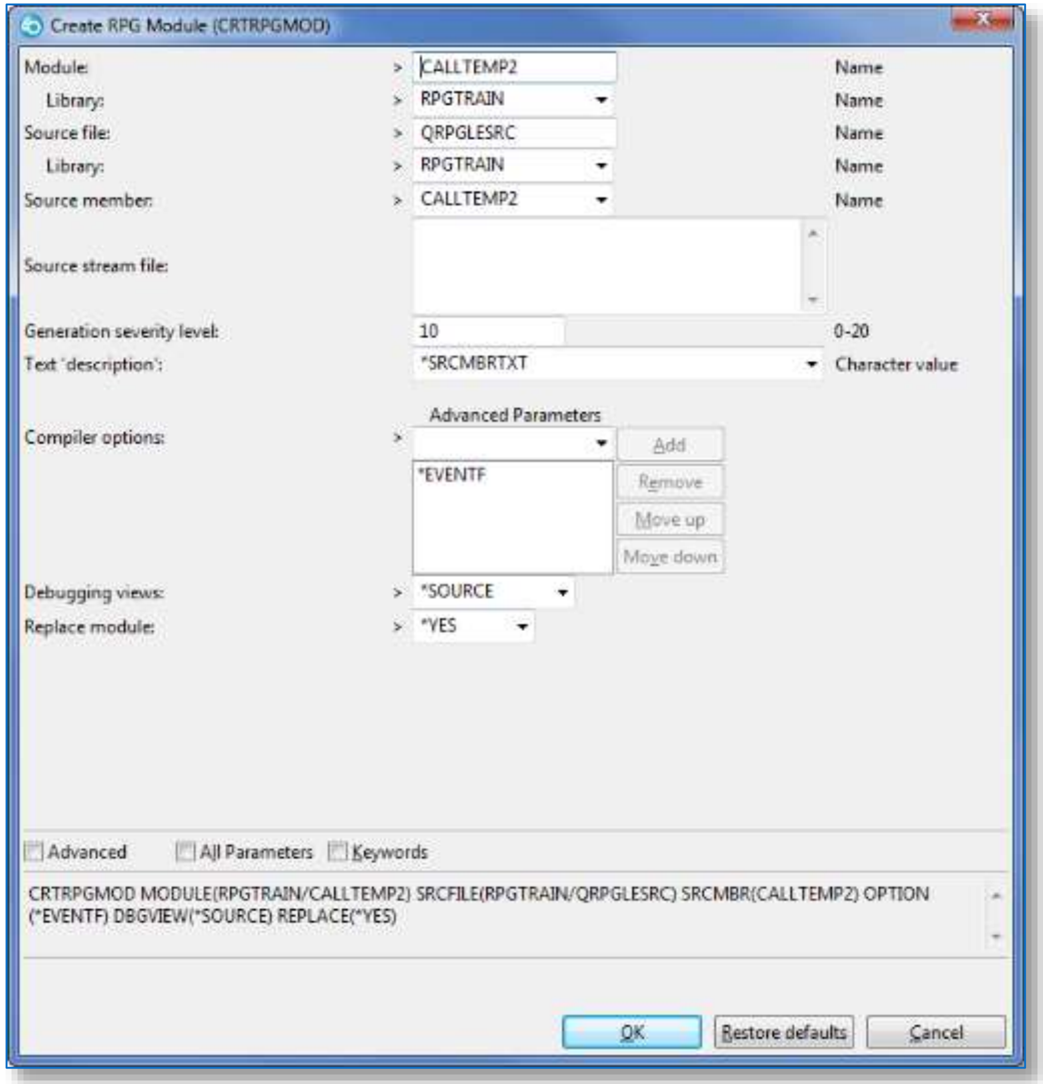
```
// ------------------------------------- Main procedure
    EnglishTemp = %Dec(CharTemp:5:2);
    Metrictemp = Celsius(Englishtemp);

    Select;
      When Metrictemp < 0;
          State = 'solid';
      When Metrictemp = 0;
          State = 'freezing';
      When Metrictemp = 100;
          State = 'boiling';
      When Metrictemp > 100;
           State = 'gaseous';
      Other;
           State = 'liquid';
      Endsl;

      Message = 'At ' + %Char(Englishtemp) + ' degrees (' +
           %Char(Metrictemp) + ' Celsius), water is ' +
                State + '.';
      Dsply Message '*REQUESTER';
      Return;
 End-proc Driver;
```

(Continued...)

Creating CALLTEMP2 - Module

CRTRPGMOD (Create RPG Module) command compiles source member, creates *MODULE object

- Module contains compiled, executable code
- Cannot run module by itself
- Module is interim building block for eventual program object



Create RPG Module (CRTRPGMOD)

Module:	>	CALLTEMP2	Name
Library:	>	RPGTRAIN	Name
Source file:	>	QRPGLESRC	Name
Library:	>	RPGTRAIN	Name
Source member:	>	CALLTEMP2	Name
Source stream file:			
Generation severity level:		10	0-20
Text 'description':		*SRCMBRTXT	Character value
Compiler options:	>	*EVENTF	
Debugging views:	>	*SOURCE	
Replace module:	>	*YES	

```
CRTRPGMOD MODULE(RPGTRAIN/CALLTEMP2) SRCFILE(RPGTRAIN/QRPGLESRC) SRCMBR(CALLTEMP2) OPTION
(*EVENTF) DBGVIEW(*SOURCE) REPLACE(*YES)
```

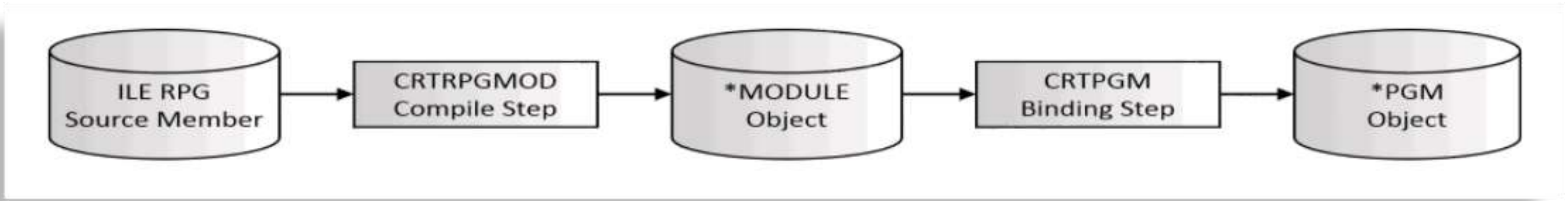
Creating Modular Programs

CRTRPGMOD (Create RPG Module) command compiles source member, creates *MODULE object

- Module contains compiled, executable code
- Cannot run module by itself
- Module is interim building block for eventual program object

CRTPGM (Create Program) command binds module to *PGM object

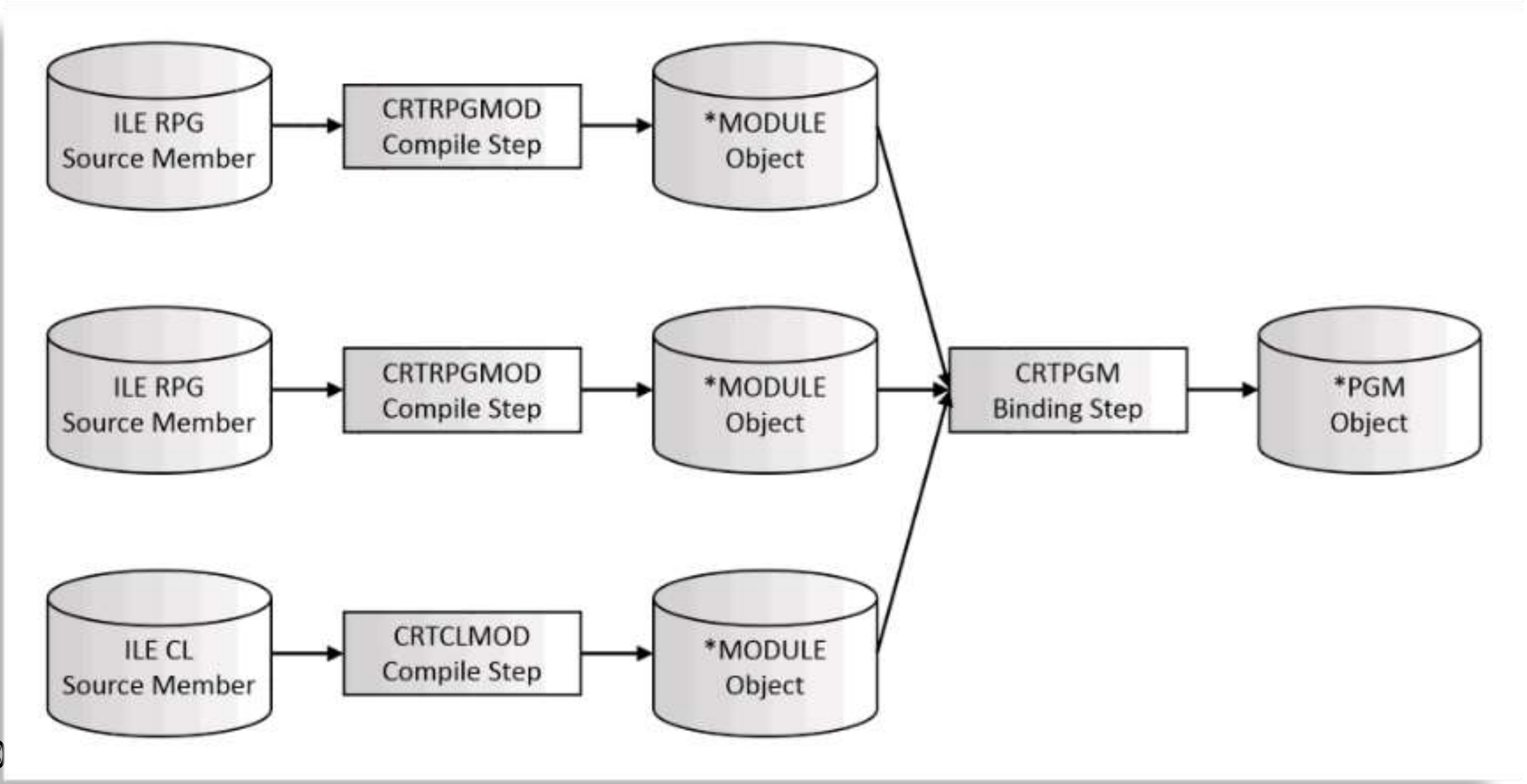
- Bind-by-copy
- Program is runnable, using CALL command



Creating Modular Programs

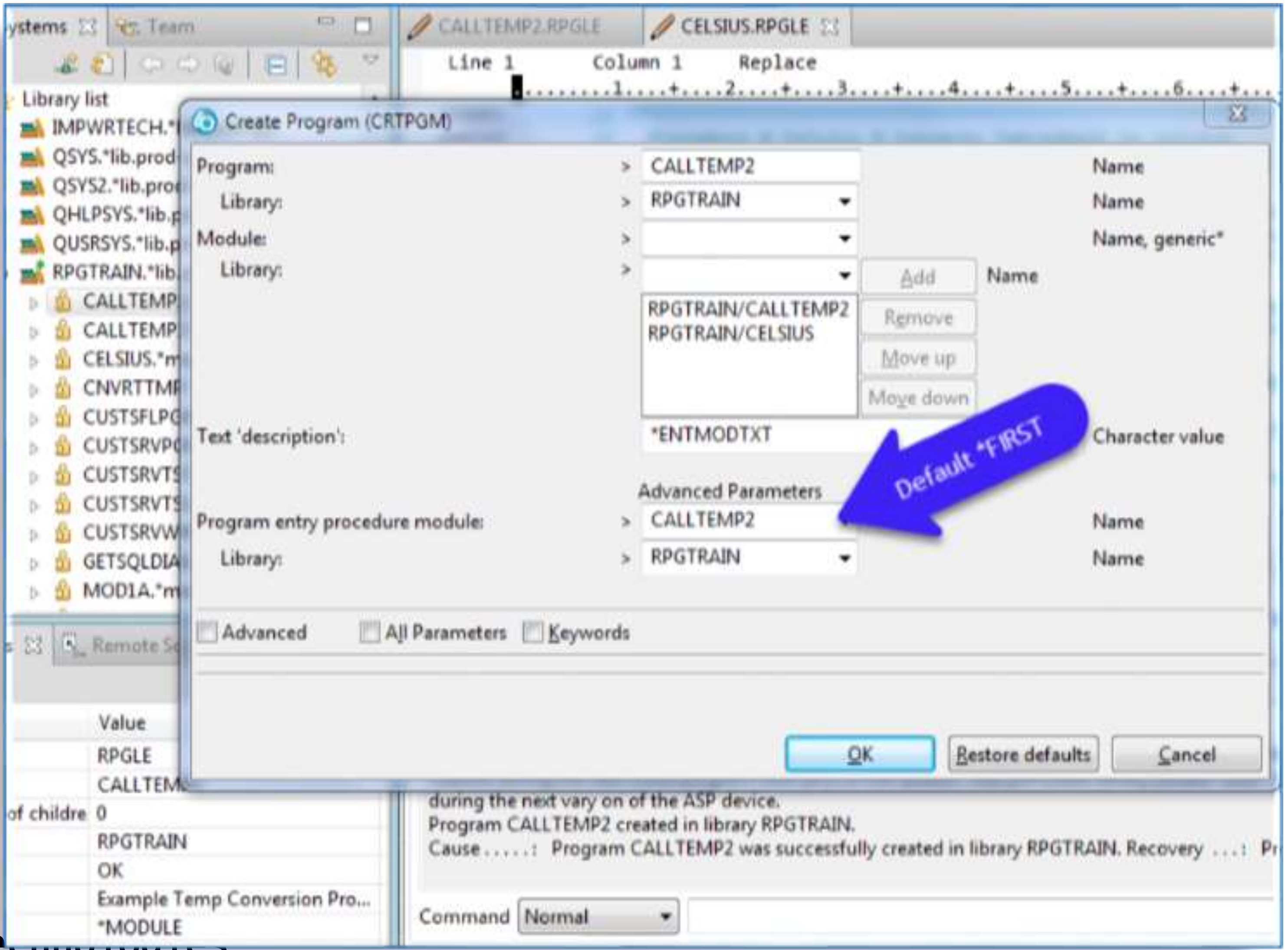
CRTPGM command can combine one or more modules during binding

- Modules may have been written using any ILE language



©

Creating the Executable Program



CALLTEMP2.RPGLE | CELSIUS.RPGLE

Line 1 Column 1 Replace

Create Program (CRTPGM)

Program:	>	CALLTEMP2	Name
Library:	>	RPGTRAIN	Name
Module:	>		Name, generic*
Library:	>		Name
		RPGTRAIN/CALLTEMP2 RPGTRAIN/CELSIUS	
Text 'description':		*ENTMODTXT	Character value
		Advanced Parameters	
Program entry procedure module:	>	CALLTEMP2	Name
Library:	>	RPGTRAIN	Name

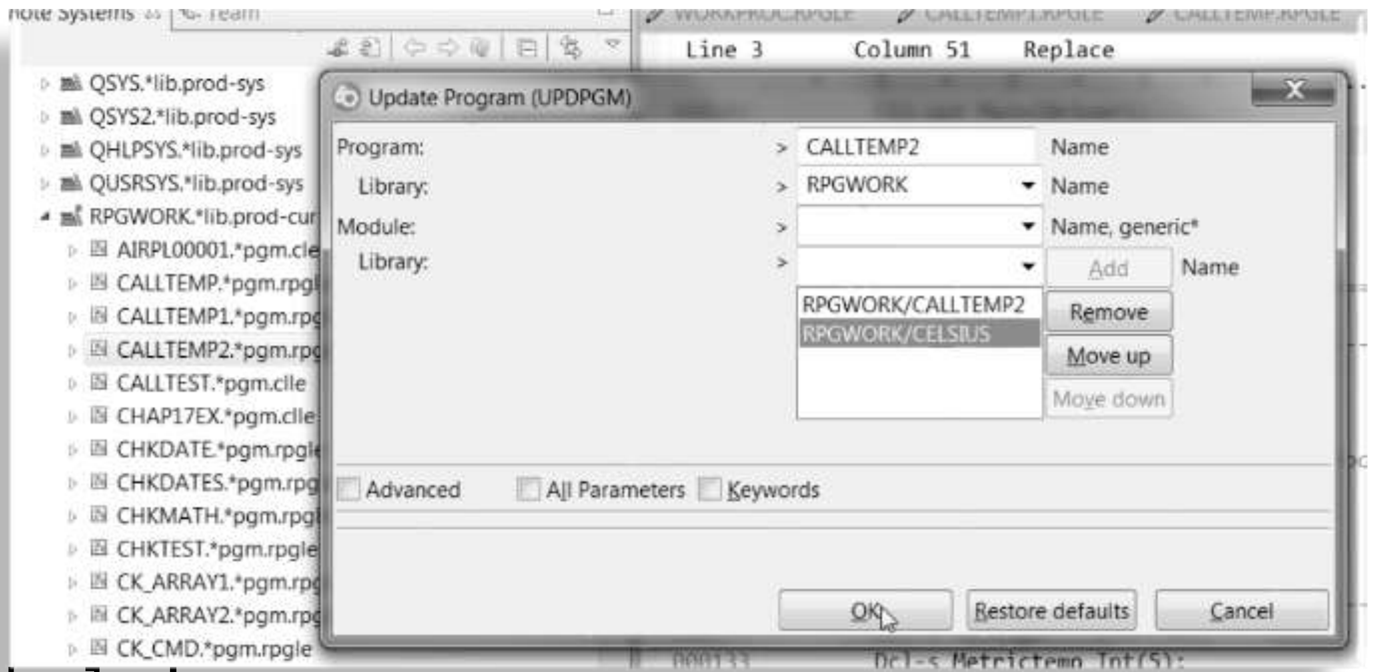
```
during the next vary on of the ASP device.
Program CALLTEMP2 created in library RPGTRAIN.
Cause . . . . . : Program CALLTEMP2 was successfully created in library RPGTRAIN. Recovery . . . : Pr
```

Maintaining Modular Programs

Program modification requires compile and binding steps

UPDPGM (Update Program) command performs abbreviated binding step

- Lists only module(s) to replace in original program
- Unchanged modules are unaffected



QSYS.*lib.prod-sys
QSYS2.*lib.prod-sys
QHLPSYS.*lib.prod-sys
QUSRSYS.*lib.prod-sys
RPGWORK.*lib.prod-cur
AIRPL00001.*pgm.cle
CALLTEMP.*pgm.rpgl
CALLTEMP1.*pgm.rpg
CALLTEMP2.*pgm.rpg
CALLTEST.*pgm.clle
CHAP17EX.*pgm.clle
CHKDATE.*pgm.rpgl
CHKDATES.*pgm.rpg
CHKMATH.*pgm.rpgl
CHKTEST.*pgm.rpgle
CK_ARRAY1.*pgm.rpg
CK_ARRAY2.*pgm.rpg
CK_CMD.*pgm.rpgle

Line 3 Column 51 Replace

Update Program (UPDPGM)

Program: CALLTEMP2 Name
Library: RPGWORK Name
Module: Name, generic*
Library: Name

RPGWORK/CALLTEMP2
RPGWORK/CELSIUS

Dcl-s Metrictemp Int(5);

Introduction to Service Programs

Service program (*SRVPGM) is code toolbox that many programs can use

- Binder need not physically copy subprocedure code into each client program
- Bind-by-reference

Service program does not have main procedure

- Any subprocedure can be entry point into service program
- Multiple entry point program

Introduction to Service Programs

Single entry point ILE program—or another service program—can invoke any exported procedure in service program

- Only one copy of actual subprocedure code exists
- Many other programs (clients) share

Combine some performance advantages of bind-by-copy static binding with modularity and flexibility benefits of dynamic program call

Compiling and Binding Service Programs

Service program procedures have no unique coding requirements

- Use Nomain modules
- Source can have global declarations section
  - Items declared in global section, before any subprocedures, are available to all procedures within module
  - Includes prototype for each procedure

Service program procedures usually use **“Export”** keyword to ensure availability to client programs

- May hide procedure inside module by omitting “Export” keyword

Compile source code with CRTRPGMOD command

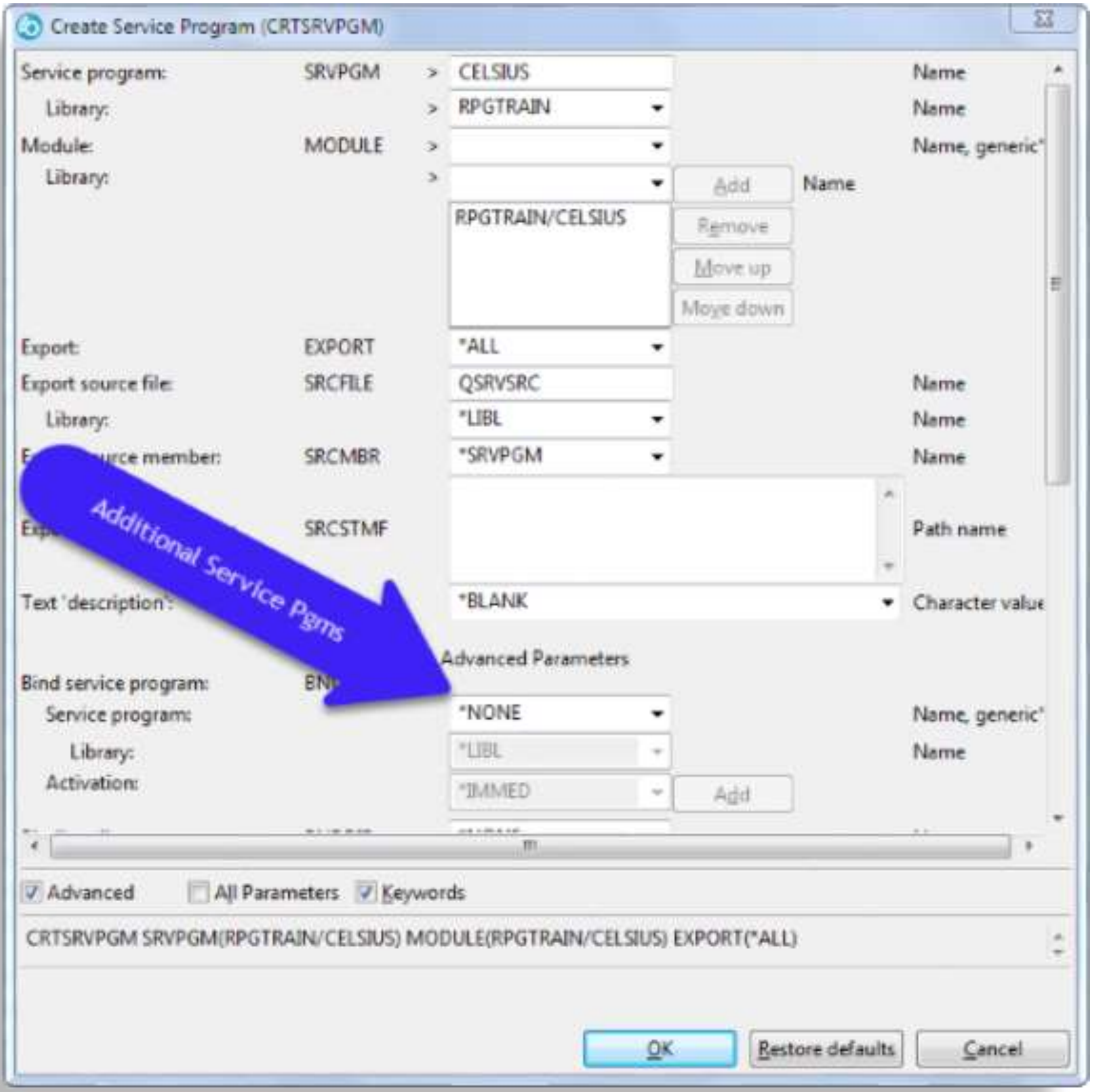
- Creates *Module object

Compiling and Binding Service Programs

We are using the *"Celsius"* Module that we created previously to create this service program.

CRTSRVPGM (Create Service Program) command binds multiple entry point service program

The actual CL command is shown in the window at right!


Create Service Program (CRTSRVPGM)

Service program:	SRVPGM	>	CELSIUS		Name
Library:		>	RPGTRAIN		Name
Module:	MODULE	>			Name, generic*
Library:		>		Add	Name
			RPGTRAIN/CELSIUS	Remove	
				Move up	
				Move down	
Export:	EXPORT		*ALL		
Export source file:	SRCFILE		QSRVSRC		Name
Library:			*LIBL		Name
Export source member:	SRCMBR		*SRVPGM		Name
Export source stream file:	SRCSTMF				Path name
Text 'description':			*BLANK		Character value
			Advanced Parameters		
Bind service program:	BNDSRVPGM				
Service program:			*NONE		Name, generic*
Library:			*LIBL		Name
Activation:			*IMMED	Add	

✓ Advanced ☐ All Parameters ✓ Keywords

```
CRTSRVPGM SRVPGM(RPGTRAIN/CELSIUS) MODULE(RPGTRAIN/CELSIUS) EXPORT(*ALL)
```

Compiling and Binding Service Programs

MODULE parameter lists modules to copy into *Srvpgm object

- No entry module

EXPORT allows you to reference a source member with a list of procedures to export. I have selected *ALL

BNDSRVPGM parameter lists other service programs to bind by reference to this service program

- If procedures refer to other procedures in other service programs

BNDDIR parameter supports binding directories

- Allows binder to find necessary modules or service programs not explicitly listed with the MODUE or BNDSRVPGM parameters

End result of successful binding step is *Srvpgm object

Deploying Service Programs in an Application

Single entry point program – or another service program – can invoke any service program procedure

- Caller does not call service program itself
- Caller calls procedure instead

```
Callp Updcust(Company:Custnbr);
```

```
Metrictemp = Celsius(Englishtemp);
```

```
If Celsius(Englishtemp) > 100;
…
Endif;
```

Deploying Service Programs in an Application

CRTPGM command decides whether to use bind-by-copy or bind-by-reference (service program)

- CRTSRVPGM command also uses same logic

Modules listed in MODULE parameter are bound by copy

Service programs listed in BNDSRVPGM parameter are bound by reference

```
CRTPGM  PGM(THISPGM)           +
        MODULE(THISPGM)        +
        ENTMOD(THISPGM)        +
        BNDSRVPGM(DATSRVPGM)   +
        BNDDIR(MYBNDDIR)
```

Bind-by-copy

Bind-by-reference

imPOWER TECHNOLOGIES
Powered by Jim Buck

Using RDi to create a Program

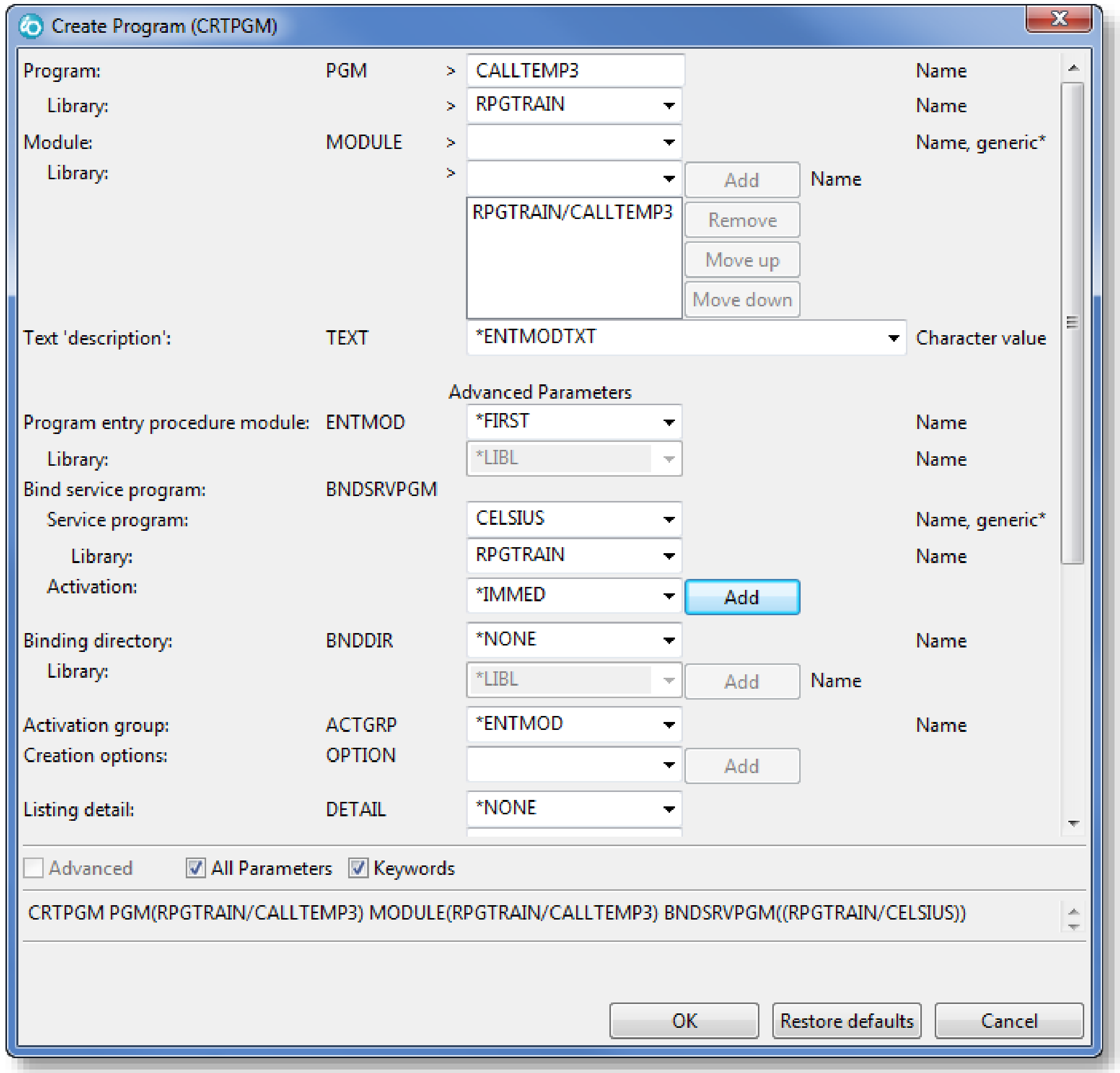
PGM - program name will be “CELSIUSPGM”

MODULE - “CALLTEMP2” Driver module, we created in a previous step

BNDSVRPGM - “MYSRVPGM” the service program we created previously

ACTGRP - *CALLER it will run in the same Activation group as the program that called “CELSIUSPGM”

We now have a functional (runnable) program


Create Program (CRTPGM)

Program:	PGM	CALLTEMP3	Name
Library:		RPGTRAIN	Name
Module:	MODULE		Name, generic*
Library:			Name
		RPGTRAIN/CALLTEMP3	
Text 'description':	TEXT	*ENTMODTXT	Character value
Advanced Parameters			
Program entry procedure module:	ENTMOD	*FIRST	Name
Library:		*LIBL	Name
Bind service program:	BNDSRVPGM		
Service program:		CELSIUS	Name, generic*
Library:		RPGTRAIN	Name
Activation:		*IMMED	
Binding directory:	BNDDIR	*NONE	Name
Library:		*LIBL	Name
Activation group:	ACTGRP	*ENTMOD	Name
Creation options:	OPTION		
Listing detail:	DETAIL	*NONE	

☐ Advanced ☑ All Parameters ☑ Keywords

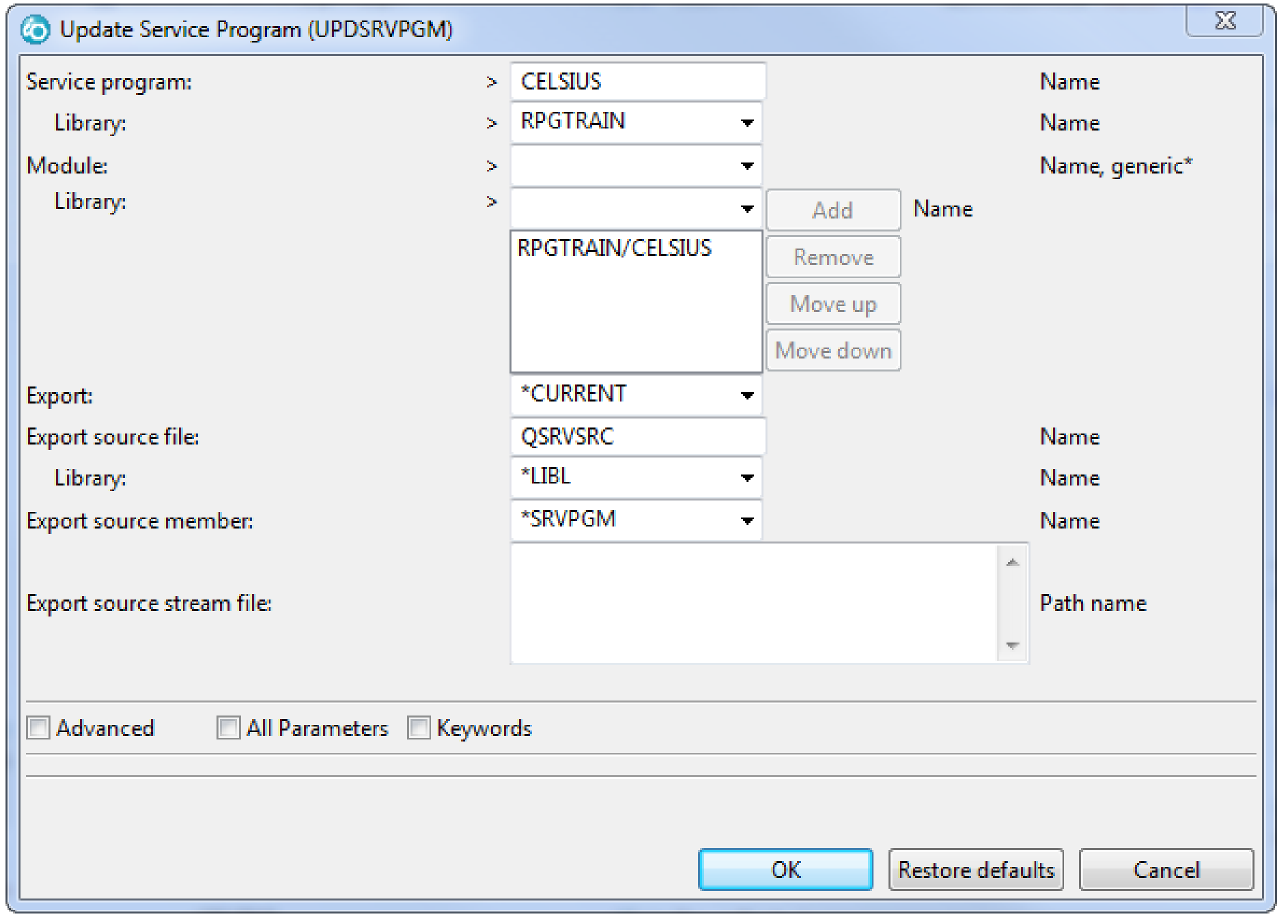
```
CRTPGM PGM(RPGTRAIN/CALLTEMP3) MODULE(RPGTRAIN/CALLTEMP3) BNDSRVPGM((RPGTRAIN/CELSIUS))
```

Maintaining Service Programs

Modification to service program procedure requires compile-then-bind process to apply changes

UPDSRVPGM (Update Service Program) command abbreviates binding step

- List only changed module(s)


Update Service Program (UPDSRVPGM)

Service program:	>	CELSIUS	Name
Library:	>	RPGTRAIN	Name
Module:	>		Name, generic*
Library:	>		Name
		RPGTRAIN/CELSIUS	
Export:		*CURRENT	
Export source file:		QSRVSRC	Name
Library:		*LIBL	Name
Export source member:		*SRVPGM	Name
Export source stream file:			Path name

Service Program Signatures

Signature is service program attribute identifying its external interface

- Usually system-generated string

When service program is bound to client, client makes note of service program signature

When client loads at runtime, system matches signatures

- Mismatch causes program load to fail

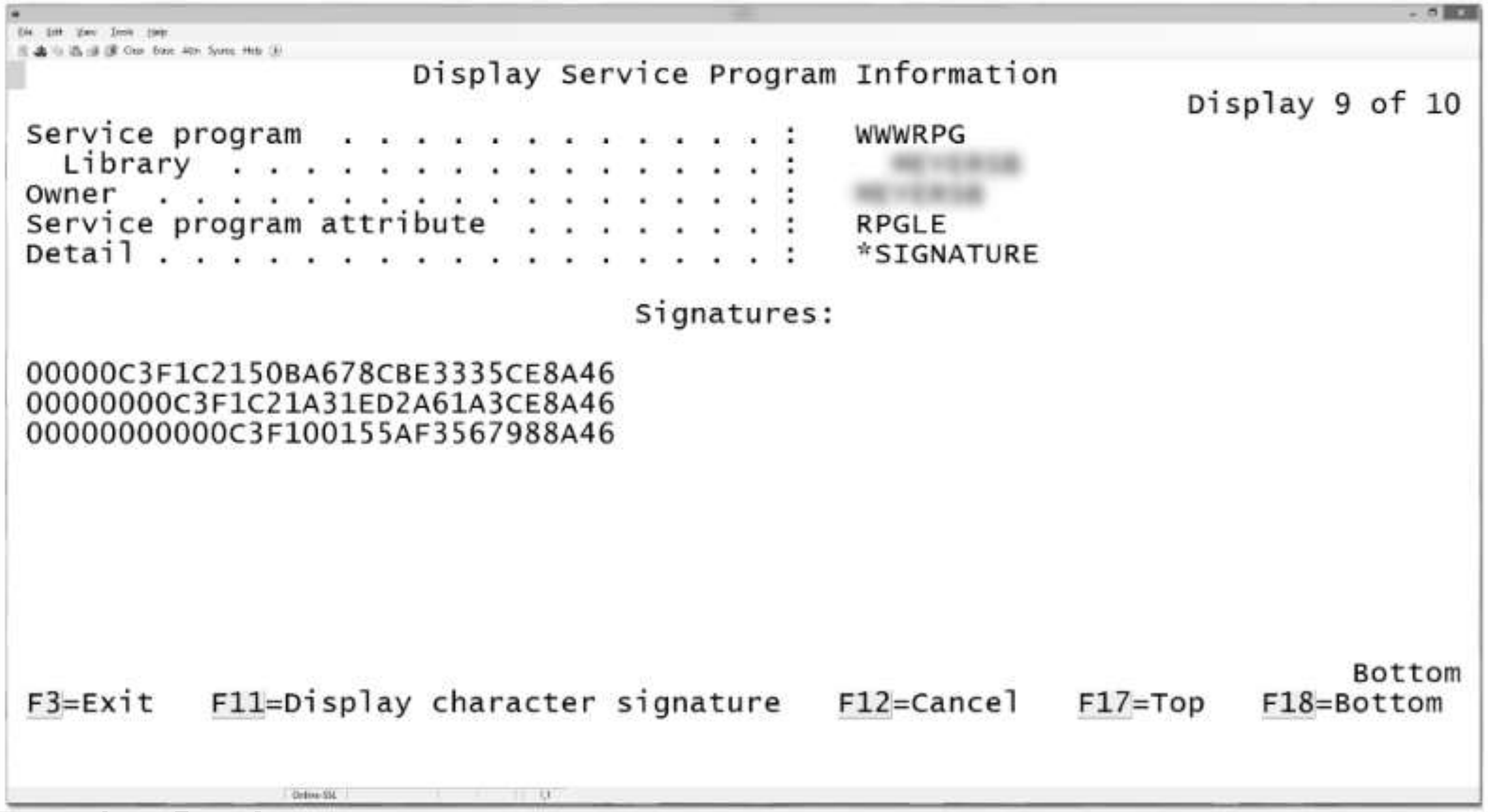
Service program has only one current signature, but can retain many previous signatures

- Binder uses current signature
- If clients are using previous signature, they can continue without recompiling or rebinding

Service Program Signatures

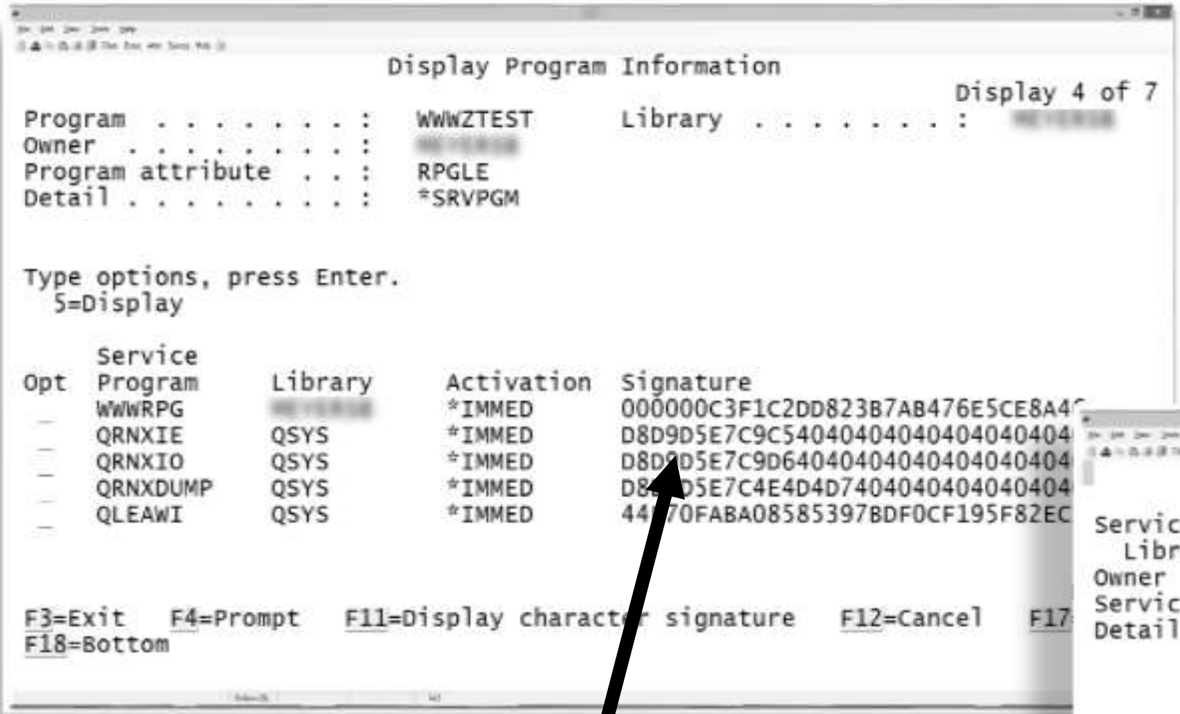
DSPSRVPGM command shows all valid signatures

- DETAIL(*SIGNATURE)

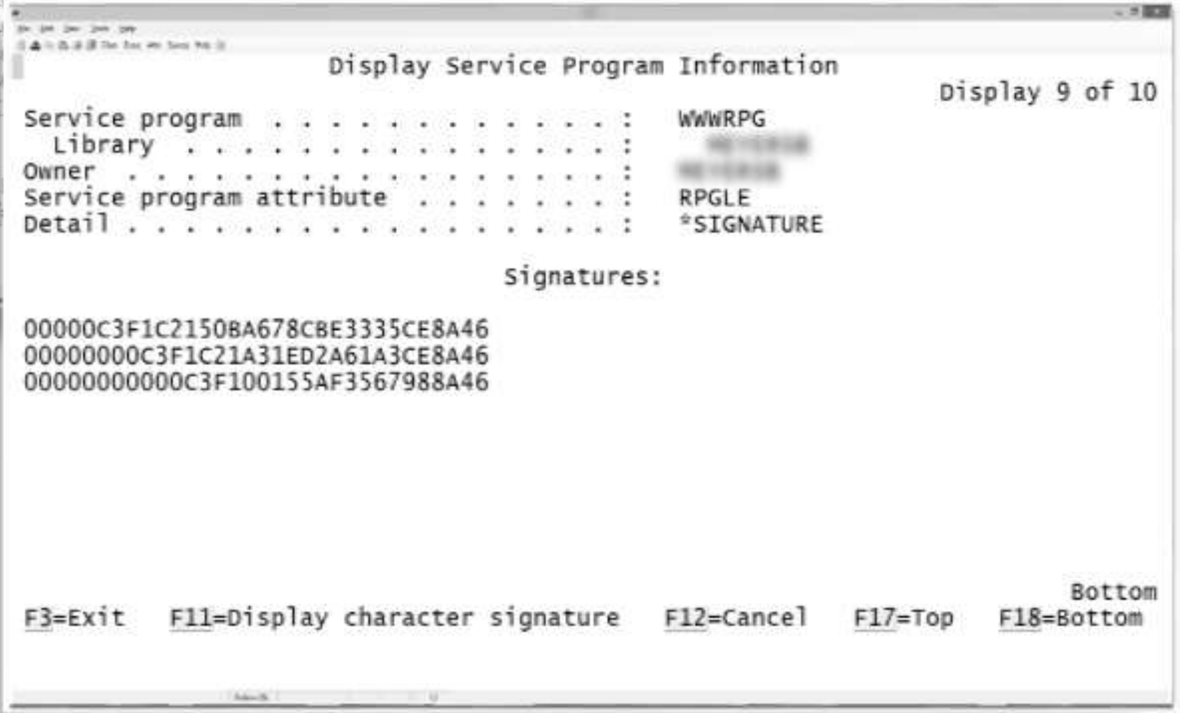


```
                    Display Service Program Information
                                                          Display 9 of 10
Service program . . . . . . . . . . . . :   WWWRPG
  Library . . . . . . . . . . . . . . . :     [redacted]
Owner . . . . . . . . . . . . . . . . . :   [redacted]
Service program attribute . . . . . . . :   RPGLE
Detail . . . . . . . . . . . . . . . . . :   *SIGNATURE

                              Signatures:

00000C3F1C2150BA678CBE3335CE8A46
00000000C3F1C21A31ED2A61A3CE8A46
00000000000C3F100155AF3567988A46




                                                                   Bottom
F3=Exit   F11=Display character signature   F12=Cancel   F17=Top   F18=Bottom
```

Service Program Signatures



```
                     Display Program Information
                                                         Display 4 of 7
Program . . . . . . . . :   WWWZTEST      Library . . . . . . . :   [redacted]
Owner . . . . . . . . . :   [redacted]
Program attribute . . . :   RPGLE
Detail . . . . . . . . . :   *SRVPGM


Type options, press Enter.
  5=Display

     Service
Opt  Program     Library      Activation  Signature
 _   WWWRPG      [redacted]   *IMMED      000000C3F1C2DD823B7AB476E5CE8A4
 _   QRNXIE      QSYS         *IMMED      D8D9D5E7C9C540404040404040404040404
 _   QRNXIO      QSYS         *IMMED      D8D9D5E7C9D640404040404040404040404
 _   QRNXDUMP    QSYS         *IMMED      D8[illegible]D5E7C4E4D4D7404040404040404
 _   QLEAWI      QSYS         *IMMED      44[illegible]70FABA08585397BDF0CF195F82EC


F3=Exit   F4=Prompt   F11=Display character signature   F12=Cancel   F17
F18=Bottom
```



```
                     Display Service Program Information
                                                         Display 9 of 10
Service program . . . . . . . . . . . . . :   WWWRPG
  Library . . . . . . . . . . . . . . . . :   [redacted]
Owner . . . . . . . . . . . . . . . . . . :   [redacted]
Service program attribute . . . . . . . . :   RPGLE
Detail . . . . . . . . . . . . . . . . . . :   *SIGNATURE

                           Signatures:

00000C3F1C2150BA678CBE3335CE8A46
00000000C3F1C21A31ED2A61A3CE8A46
00000000000C3F100155AF3567988A46




                                                                  Bottom
F3=Exit   F11=Display character signature   F12=Cancel   F17=Top   F18=Bottom
```

Must match

Using Binder Language

Binder language describes service program's signature

- Stored in source file

Three commands

- STRPGMEXP (Start Program Export List) begins signature
- EXPORT (Export a Program Symbol) names procedure(s)
- ENDPGMEXP (End Program Export List) ends signature

```
STRPGMEXP PGMLVL(*CURRENT)
  EXPORT SYMBOL(WWWDECODE)
  EXPORT SYMBOL(WWWDUMP)
  EXPORT SYMBOL(WWWECHO)
  EXPORT SYMBOL(WWWEXTRACT)
ENDPGMEXP
```

Using Binder Language

Refer to binder language source when creating service program

- CRTSRVPGM … EXPORT(*SRCFILE) SRCFILE(filename) SRCMBR(member)

```
CRTSRVPGM SRVPGM(WWWRPG)                                         +
          MODULE(WWWDECODE WWWDUMP WWWECHO WWWEXTRACT) +
          EXPORT(*SRCFILE)                                       +
          SRCFILE(QSRVSRC)                                       +
          SRCMBR(WWWRPG)
```

Using Binder Language

```
STRPGMEXP PGMLVL(*CURRENT)
  EXPORT SYMBOL(WWWDECODE)
  EXPORT SYMBOL(WWWDUMP)
  EXPORT SYMBOL(WWWECHO)
  EXPORT SYMBOL(WWWGETDOC)
  EXPORT SYMBOL(WWWREAD)
  EXPORT SYMBOL(WWWREPLACE)
ENDPGMEXP

STRPGMEXP PGMLVL(*PRV)
  EXPORT SYMBOL(WWWDECODE)
  EXPORT SYMBOL(WWWDUMP)
  EXPORT SYMBOL(WWWECHO)
  EXPORT SYMBOL(WWWGETDOC)
ENDPGMEXP

STRPGMEXP PGMLVL(*PRV)
  EXPORT SYMBOL(WWWDECODE)
  EXPORT SYMBOL(WWWDUMP)
  EXPORT SYMBOL(WWWECHO)
ENDPGMEXP
```

Binder language manages service program's current signature and any previous signatures

DO NOT REMOVE OR REARRANGE
entries in existing signatures

- Add new procedures to end of signature block
- Current signature block must contain same procedures in same order as previous signature(s)

Using Binder Language

May explicitly name signature

- Up to 16 characters

Service program needs only one named signature block

- Add new procedures to end of named signature
- Existing programs will still run without rebinding

```
STRPGMEXP PGMLVL(*CURRENT) SIGNATURE(WWWRPG)
  EXPORT SYMBOL(WWWDECODE)
  EXPORT SYMBOL(WWWDUMP)
  EXPORT SYMBOL(WWWECHO)
  EXPORT SYMBOL(WWWGETDOC)
  EXPORT SYMBOL(WWWREAD)
  EXPORT SYMBOL(WWWREPLACE)
ENDPGMEXP
```

Exporting Data Items

Export keyword can declare variable or data structure whose contents are available across modules

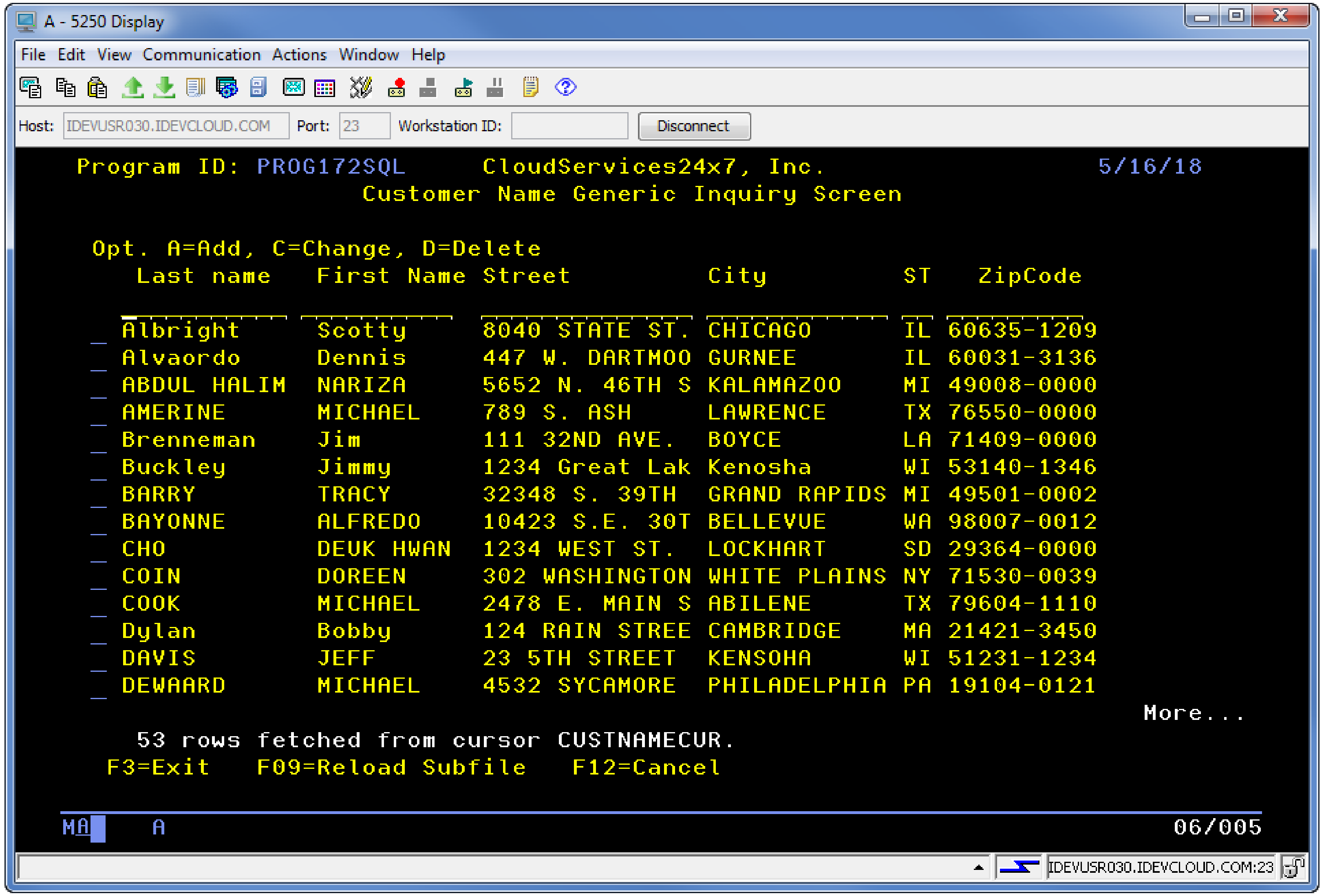
- Only one module actually allocates storage
  - Export keyword
- Other modules use data item allocated by exporting module
  - Import keyword

May be useful to export data items from service program and then import them into client

```
// Module: PRIMARY
Dcl-s Mydata Packed(9:2) Export Inz(200);
…
Proca();
```

```
// Module: PROCA
Dcl-s Mydata Packed(9:2) Import;
…
```

Subfile Application Example


A - 5250 Display
File Edit View Communication Actions Window Help
Host: IDEVUSR030.IDEVCLOUD.COM Port: 23 Workstation ID:
Disconnect

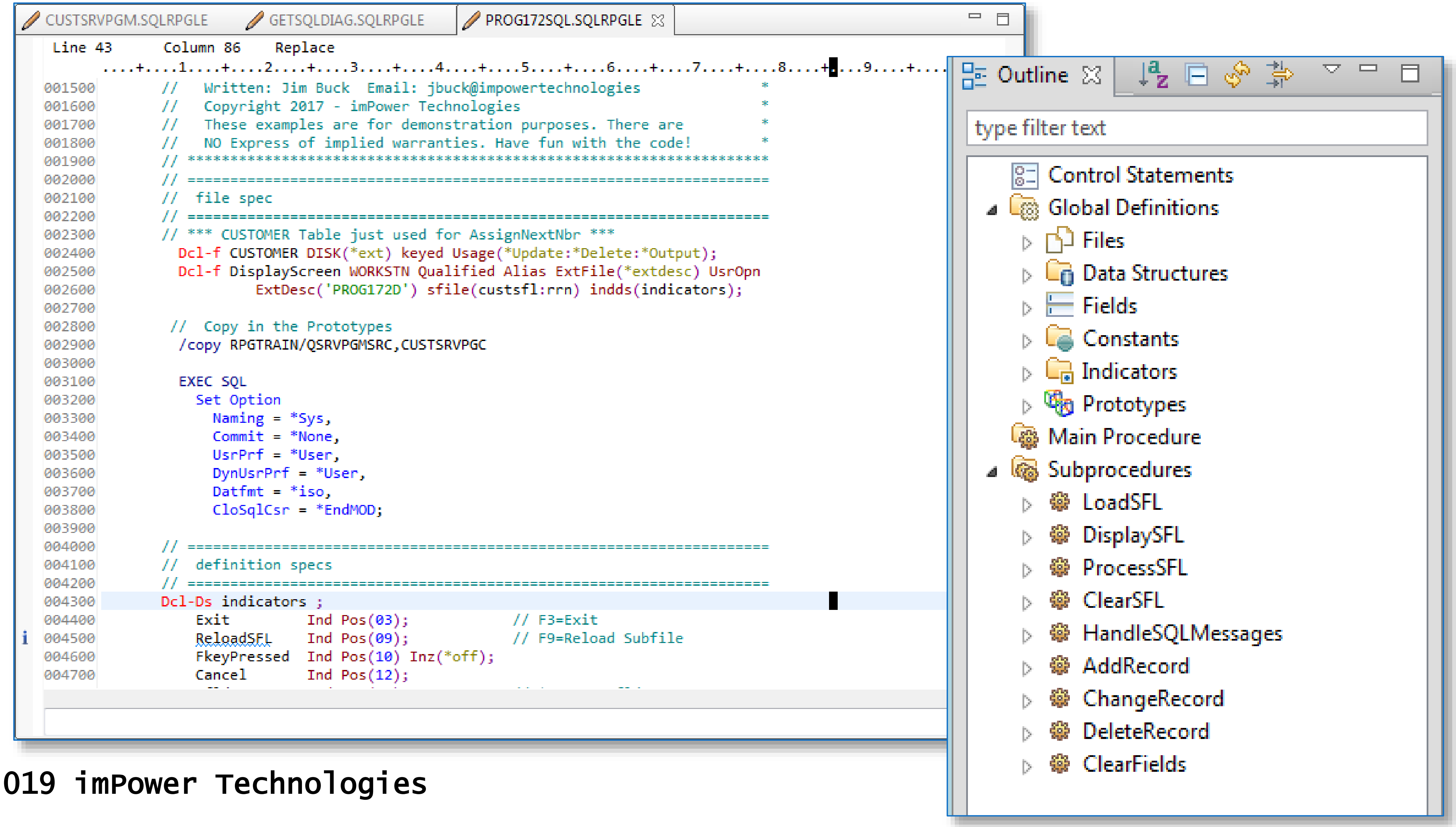
```
 Program ID: PROG172SQL     CloudServices24x7, Inc.                    5/16/18
                       Customer Name Generic Inquiry Screen

  Opt. A=Add, C=Change, D=Delete
     Last name    First Name Street         City         ST   ZipCode

 _  Albright     Scotty     8040 STATE ST. CHICAGO      IL 60635-1209
 _  Alvaordo     Dennis     447 W. DARTMOO GURNEE       IL 60031-3136
 _  ABDUL HALIM  NARIZA     5652 N. 46TH S KALAMAZOO    MI 49008-0000
 _  AMERINE      MICHAEL    789 S. ASH     LAWRENCE     TX 76550-0000
 _  Brenneman    Jim        111 32ND AVE.  BOYCE        LA 71409-0000
 _  Buckley      Jimmy      1234 Great Lak Kenosha      WI 53140-1346
 _  BARRY        TRACY      32348 S. 39TH  GRAND RAPIDS MI 49501-0002
 _  BAYONNE      ALFREDO    10423 S.E. 30T BELLEVUE     WA 98007-0012
 _  CHO          DEUK HWAN  1234 WEST ST.  LOCKHART     SD 29364-0000
 _  COIN         DOREEN     302 WASHINGTON WHITE PLAINS NY 71530-0039
 _  COOK         MICHAEL    2478 E. MAIN S ABILENE      TX 79604-1110
 _  Dylan        Bobby      124 RAIN STREE CAMBRIDGE    MA 21421-3450
 _  DAVIS        JEFF       23 5TH STREET  KENSOHA      WI 51231-1234
 _  DEWAARD      MICHAEL    4532 SYCAMORE  PHILADELPHIA PA 19104-0121
                                                                 More...
    53 rows fetched from cursor CUSTNAMECUR.
   F3=Exit   F09=Reload Subfile   F12=Cancel

MA     A                                                           06/005
```

Subfile Application – Putting it Together

Comprised of three programs + copybook

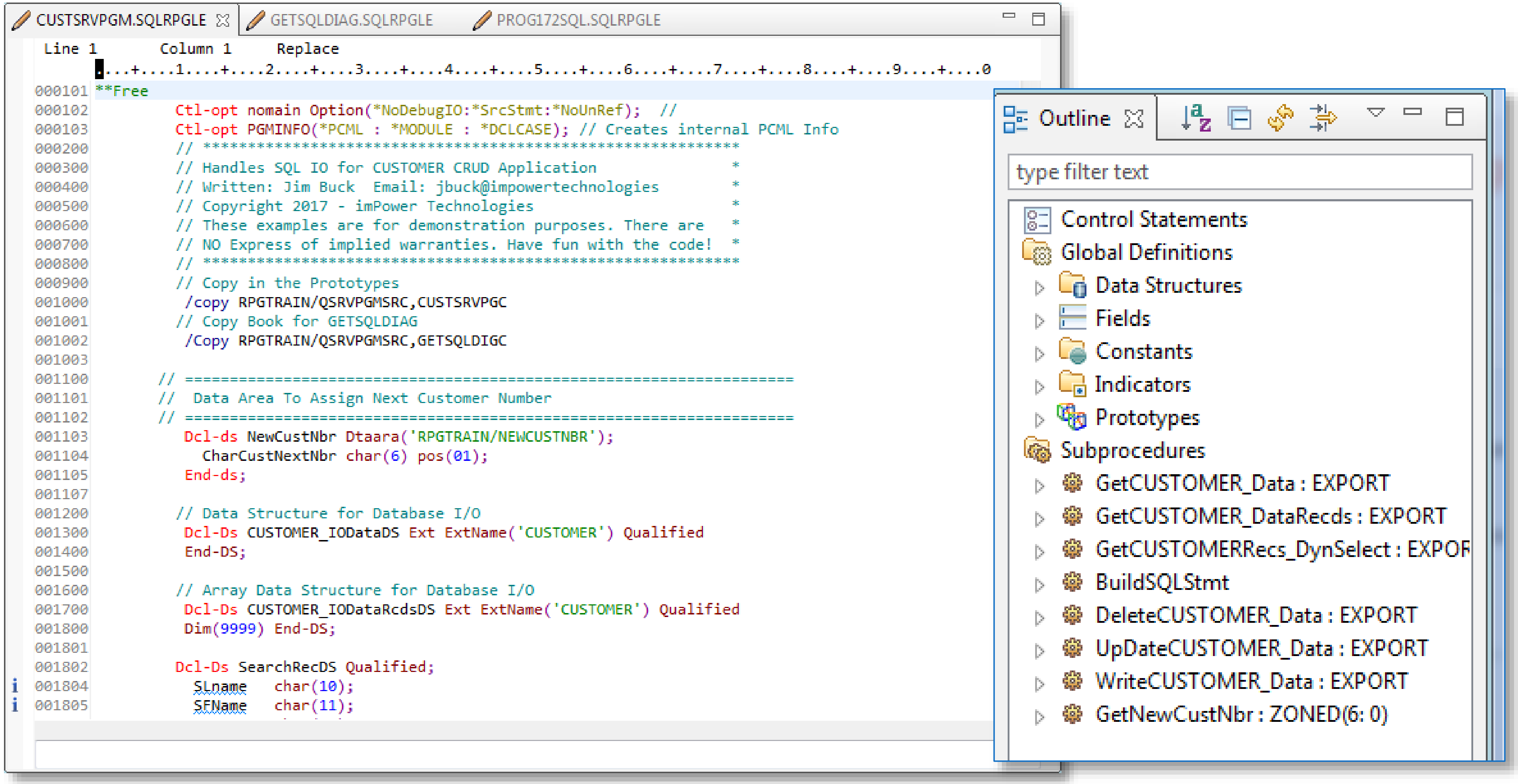
- **PROG172SQL** – Main Driver Program
  - Runs the 5250 screens
  - Handles the CREATE, READ, UPDATE and DELETE Logic
- **CUSTSRVPGM** – Service program that handles SQL I/O
  - SQL INSERT, UPDATE, SELECT and DELETE Code
  - Returns data Structures (Customer and SQL Status)
- **GETSQLDIAG** – Service program that:
  - Processes the GET DIAGNOSTICS command
  - Puts the results into a data structure
  - Returns this data structure to the calling program

PROG172SQL – Driver program



```
 Line 43      Column 86     Replace
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...
      //   Written: Jim Buck  Email: jbuck@impowertechnologies           *
      //   Copyright 2017 - imPower Technologies                         *
      //   These examples are for demonstration purposes. There are      *
      //   NO Express of implied warranties. Have fun with the code!     *
      // *******************************************************************
      // ===================================================================
      //  file spec
      // ===================================================================
      // *** CUSTOMER Table just used for AssignNextNbr ***
        Dcl-f CUSTOMER DISK(*ext) keyed Usage(*Update:*Delete:*Output);
        Dcl-f DisplayScreen WORKSTN Qualified Alias ExtFile(*extdesc) UsrOpn
                 ExtDesc('PROG172D') sfile(custsfl:rrn) indds(indicators);

       //  Copy in the Prototypes
        /copy RPGTRAIN/QSRVPGMSRC,CUSTSRVPGC

        EXEC SQL
          Set Option
            Naming = *Sys,
            Commit = *None,
            UsrPrf = *User,
            DynUsrPrf = *User,
            Datfmt = *iso,
            CloSqlCsr = *EndMOD;

      // ===================================================================
      //  definition specs
      // ===================================================================
      Dcl-Ds indicators ;
          Exit         Ind Pos(03);             // F3=Exit
          ReloadSFL    Ind Pos(09);             // F9=Reload Subfile
          FkeyPressed  Ind Pos(10) Inz(*off);
          Cancel       Ind Pos(12);
```

CUSTSRVPGM – SQL Database I/O



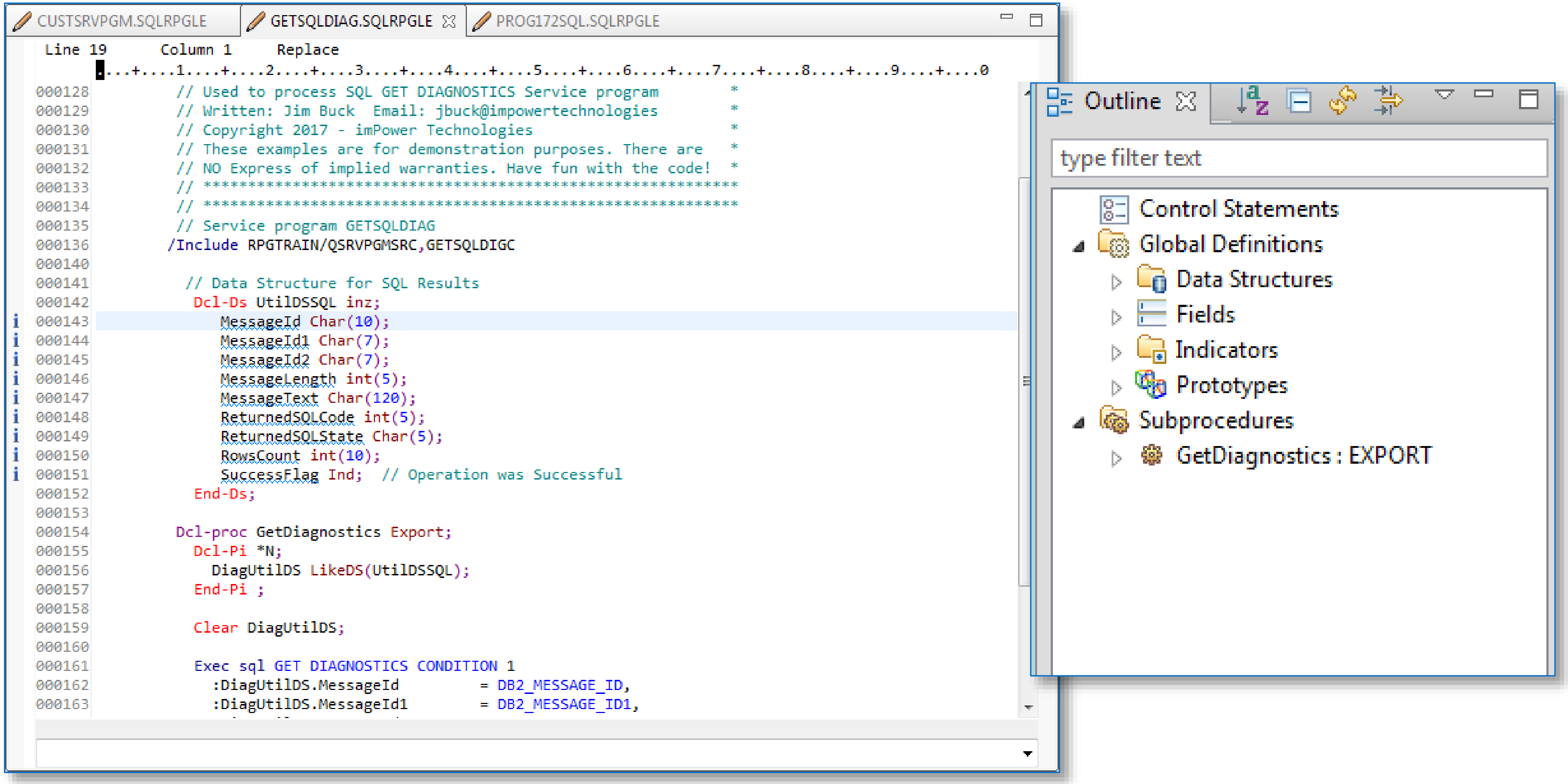
```
Line 1       Column 1     Replace
 ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0
**Free
         Ctl-opt nomain Option(*NoDebugIO:*SrcStmt:*NoUnRef);  //
         Ctl-opt PGMINFO(*PCML : *MODULE : *DCLCASE); // Creates internal PCML Info
         // ************************************************************
         // Handles SQL IO for CUSTOMER CRUD Application              *
         // Written: Jim Buck  Email: jbuck@impowertechnologies       *
         // Copyright 2017 - imPower Technologies                     *
         // These examples are for demonstration purposes. There are  *
         // NO Express of implied warranties. Have fun with the code! *
         // ************************************************************
         // Copy in the Prototypes
          /copy RPGTRAIN/QSRVPGMSRC,CUSTSRVPGC
         // Copy Book for GETSQLDIAG
          /Copy RPGTRAIN/QSRVPGMSRC,GETSQLDIGC

       // ======================================================================
       //  Data Area To Assign Next Customer Number
       // ======================================================================
          Dcl-ds NewCustNbr Dtaara('RPGTRAIN/NEWCUSTNBR');
            CharCustNextNbr char(6) pos(01);
          End-ds;

         // Data Structure for Database I/O
          Dcl-Ds CUSTOMER_IODataDS Ext ExtName('CUSTOMER') Qualified
          End-DS;

         // Array Data Structure for Database I/O
          Dcl-Ds CUSTOMER_IODataRcdsDS Ext ExtName('CUSTOMER') Qualified
          Dim(9999) End-DS;

         Dcl-Ds SearchRecDS Qualified;
           SLname    char(10);
           SFName    char(11);
```

Outline

type filter text

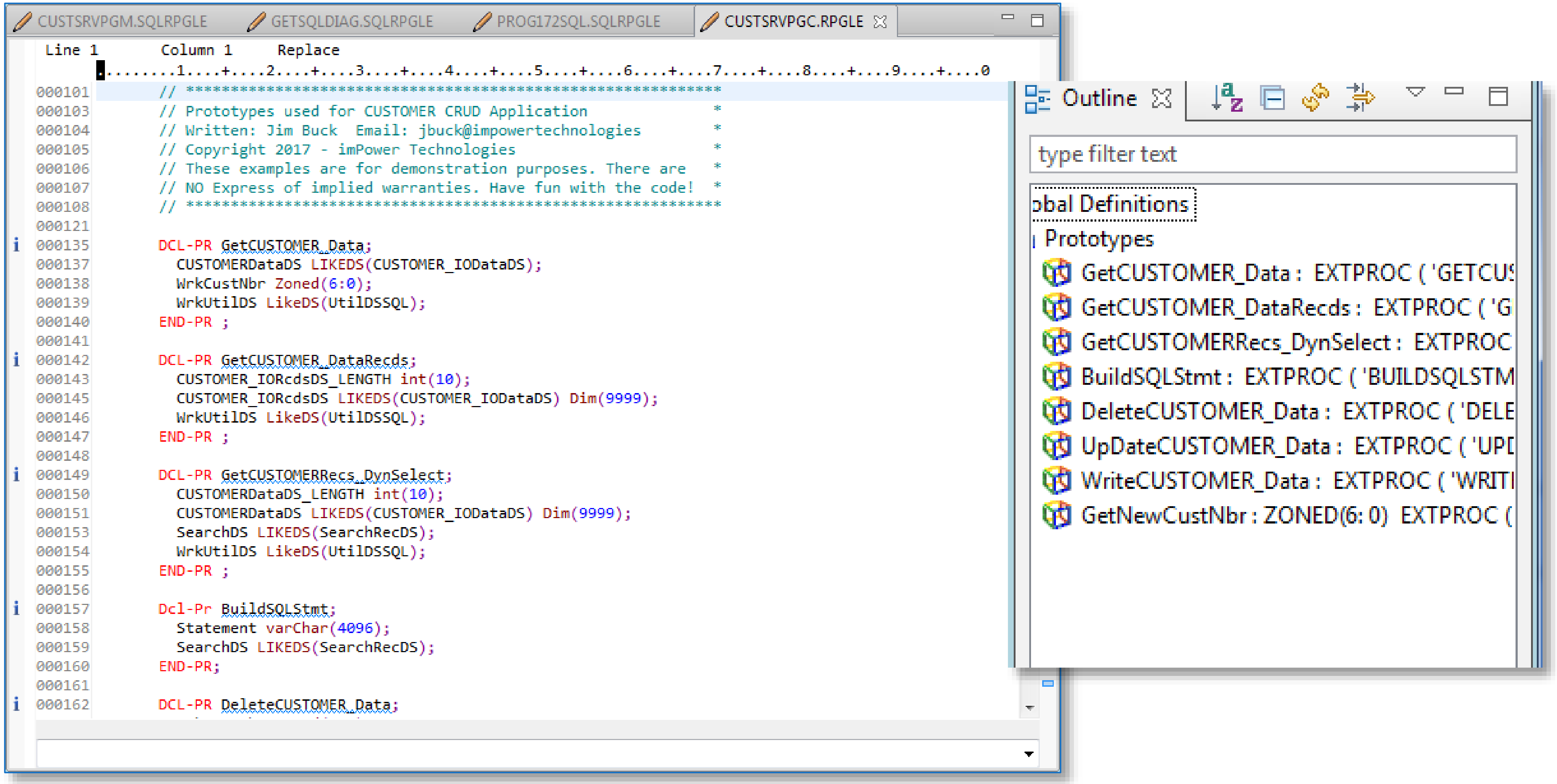
- Control Statements
- Global Definitions
  - Data Structures
  - Fields
  - Constants
  - Indicators
  - Prototypes
- Subprocedures
  - GetCUSTOMER_Data : EXPORT
  - GetCUSTOMER_DataRecds : EXPORT
  - GetCUSTOMERRecs_DynSelect : EXPOR
  - BuildSQLStmt
  - DeleteCUSTOMER_Data : EXPORT
  - UpDateCUSTOMER_Data : EXPORT
  - WriteCUSTOMER_Data : EXPORT
  - GetNewCustNbr : ZONED(6: 0)

GETSQLDIAG – SQL Database I/O



```
Line 19      Column 1      Replace
 ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0
        // Used to process SQL GET DIAGNOSTICS Service program       *
        // Written: Jim Buck  Email: jbuck@impowertechnologies       *
        // Copyright 2017 - imPower Technologies                     *
        // These examples are for demonstration purposes. There are  *
        // NO Express of implied warranties. Have fun with the code! *
        // ************************************************************
        // ************************************************************
        // Service program GETSQLDIAG
       /Include RPGTRAIN/QSRVPGMSRC,GETSQLDIGC

         // Data Structure for SQL Results
          Dcl-Ds UtilDSSQL inz;
             MessageId Char(10);
             MessageId1 Char(7);
             MessageId2 Char(7);
             MessageLength int(5);
             MessageText Char(120);
             ReturnedSQLCode int(5);
             ReturnedSQLState Char(5);
             RowsCount int(10);
             SuccessFlag Ind;  // Operation was Successful
          End-Ds;

        Dcl-proc GetDiagnostics Export;
          Dcl-Pi *N;
            DiagUtilDS LikeDS(UtilDSSQL);
          End-Pi ;

          Clear DiagUtilDS;

          Exec sql GET DIAGNOSTICS CONDITION 1
            :DiagUtilDS.MessageId         = DB2_MESSAGE_ID,
            :DiagUtilDS.MessageId1        = DB2_MESSAGE_ID1,
```

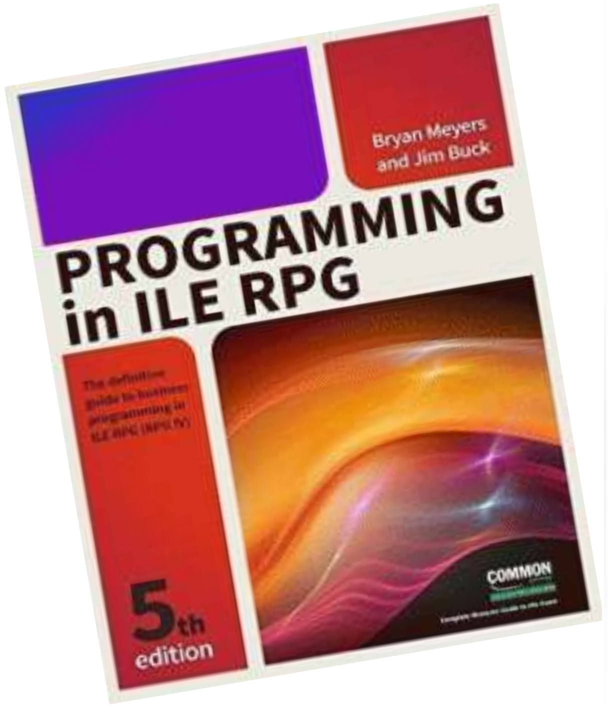
CUSTSRVCPY – Prototype Copybook



```
.........1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0
       // *************************************************************
       // Prototypes used for CUSTOMER CRUD Application               *
       // Written: Jim Buck  Email: jbuck@impowertechnologies         *
       // Copyright 2017 - imPower Technologies                       *
       // These examples are for demonstration purposes. There are    *
       // NO Express of implied warranties. Have fun with the code!   *
       // *************************************************************

       DCL-PR GetCUSTOMER_Data;
         CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS);
         WrkCustNbr Zoned(6:0);
         WrkUtilDS LikeDS(UtilDSSQL);
       END-PR ;

       DCL-PR GetCUSTOMER_DataRecds;
         CUSTOMER_IORcdsDS_LENGTH int(10);
         CUSTOMER_IORcdsDS LIKEDS(CUSTOMER_IODataDS) Dim(9999);
         WrkUtilDS LikeDS(UtilDSSQL);
       END-PR ;

       DCL-PR GetCUSTOMERRecs_DynSelect;
         CUSTOMERDataDS_LENGTH int(10);
         CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS) Dim(9999);
         SearchDS LIKEDS(SearchRecDS);
         WrkUtilDS LikeDS(UtilDSSQL);
       END-PR ;

       Dcl-Pr BuildSQLStmt;
         Statement varChar(4096);
         SearchDS LIKEDS(SearchRecDS);
       END-PR;

       DCL-PR DeleteCUSTOMER_Data;
```

```
obal Definitions
 Prototypes
   GetCUSTOMER_Data : EXTPROC ( 'GETCUS
   GetCUSTOMER_DataRecds : EXTPROC ( 'G
   GetCUSTOMERRecs_DynSelect : EXTPROC
   BuildSQLStmt : EXTPROC ( 'BUILDSQLSTM
   DeleteCUSTOMER_Data : EXTPROC ( 'DELE
   UpDateCUSTOMER_Data : EXTPROC ( 'UPD
   WriteCUSTOMER_Data : EXTPROC ( 'WRITI
   GetNewCustNbr : ZONED(6: 0) EXTPROC (
```

Questions or Comments?



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

MAGiC
MID-ATLANTIC GROUP OF IBM i COLLABORATORS