



*RPG IV Programming
Fundamentals Workshop for
IBM i*

(Course code AS06)

Student Notebook

ERC 7.0

Authorized

IBM | Training

Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AS/400®	AS/400e™	DB™
DB2®	Integrated Language Environment®	iSeries®
Iterations®	i5/OS™	Language Environment®
OS/400®	Power Systems™	Power Systems Software™
Power®	Rational®	Redbooks®
RPG/400®	System i®	WebSphere®
400®		

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

November 2012 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Contents

Trademarks	xi
Course description	xiii
Agenda	xvii
Unit 1. RPG IV introduction	1-1
Unit objectives	1-2
RPG IV's legacy	1-3
What is RPG IV?	1-4
Application development tools: Packaging	1-6
Development cycle	1-8
Describing a programming solution	1-10
RPG IV terminology	1-11
RPG IV specifications	1-12
Print program: RPG code	1-14
Print program: Report	1-15
Create RPG IV program	1-16
IBM i source physical file	1-18
The process of creating an RPG IV program	1-20
CRTBNDRPG compile options	1-21
Compiler listing	1-23
Compiler directives	1-24
Machine exercise: Coding and compiling RPG IV	1-28
Checkpoint (1 of 2)	1-29
Checkpoint (2 of 2)	1-30
Unit summary	1-31
Unit 2. Coding specifications for RPG IV	2-1
Unit objectives	2-3
Source specifications	2-4
Symbolic names	2-5
Field naming rules	2-7
RPG IV indicators	2-8
Machine exercise: Sequencing RPG IV specifications and compiling	2-9
H-specification	2-10
H-specification keywords	2-12
F-specifications	2-13
F-specification prompting	2-15
F-specification components	2-16
Device type and processing	2-17
Database processing options	2-18
H- and F-specifications: Employee print program	2-19
D-specifications	2-21

D-specification prompting	2-22
D-specification to define data	2-23
Defining a data structure and array	2-25
Where are externally described descriptions?	2-27
I-specification for input data	2-28
C-specifications	2-29
C-specification structure	2-30
Typical calculations	2-31
Common opcodes	2-32
Rules of free-format coding	2-33
Output specifications	2-35
Class exercise: Coding a simple listing	2-36
Machine exercise: Coding a report program and adding overflow	2-37
Checkpoint	2-38
Unit summary	2-39
 Unit 3. Data representation and definition.	 3-1
Unit objectives	3-2
RPG IV data types	3-3
Character data	3-4
Numeric data	3-6
Zoned decimal	3-7
Packed decimal	3-8
Integer/Unsigned	3-10
Named indicators	3-12
Other data types	3-13
Figurative constants	3-14
Data definition	3-15
Externally described files	3-16
Defining externally described data	3-17
Overriding externally defined data names	3-18
D-specification layout	3-20
Defining standalone fields	3-22
Defining named constants	3-24
LIKE	3-25
Machine exercise: Data definition	3-26
Checkpoint	3-27
Unit summary	3-28
 Unit 4. Manipulating data in calculations	 4-1
Unit objectives	4-3
Types of expressions	4-4
Assignment opcodes	4-5
Assignment of numeric variable	4-7
Assignment of character variable	4-8
Arithmetic operations	4-9
Arithmetic opcodes	4-10
Addition and subtraction	4-12

Multiplication	4-14
Numeric overflow	4-16
Division and remainders	4-18
Coding BiFs	4-20
Exponentiation	4-21
Machine exercise: Adding arithmetic function	4-22
Comparison operations	4-23
Class exercise	4-25
If/else/endIf	4-26
IF expressions	4-27
Using do while	4-28
String handling	4-29
String handling BiFs	4-30
Concatenation	4-31
Remove characters: %TRIM	4-32
Length and size of a string: %Len / %Size	4-33
Scan and replace for character string: %Scanrpl	4-34
Scan for character string: %Scan (1 of 2)	4-36
Scan for character string: %Scan (2 of 2)	4-37
Extract substring: %Subst	4-38
Replace string of characters: %Replace	4-39
Convert to character: %Char	4-41
Edit with %EditC / %EditW	4-42
Convert to numeric: %INT and %DEC	4-44
Convert string: %XLATE	4-45
Precedence rules	4-46
Precedence rules: Using parentheses	4-47
Precision: Expressions	4-48
Specifying precision rules	4-50
Machine exercise: Data manipulation	4-51
Checkpoint	4-52
Unit summary	4-53
 Unit 5. Printing from an RPG IV program.....	 5-1
Unit objectives	5-2
Printing process	5-3
Printing elements	5-5
Elements that control printing	5-7
Which OUTQ?	5-9
Using a printer file	5-10
Externally described printer files	5-13
Printer file characteristics	5-14
Printer file DDS	5-15
Printer DDS keywords	5-17
PRTF DDS	5-18
Output report: Example	5-20
PRTF: DDS example	5-21
PRTF RPG IV program: Itemlist	5-24

Edit codes for printed output	5-26
Edit examples: External print file	5-27
Creating, storing, and executing printer formats	5-28
Definition of reports	5-29
Machine exercise: Printing from an RPG IV program	5-30
Checkpoint	5-31
Unit summary	5-32
 Unit 6. Using the debugger	 6-1
Unit objectives	6-2
Source view debugger	6-3
Beginning a debug session	6-4
Compiling a program for debug	6-5
Using the ILE debugger	6-6
Example of source debugging	6-7
Using ILE debug commands	6-9
Using the debug EVAL command	6-10
Watching a variable	6-12
More debug commands	6-14
Source navigation	6-15
Debug views	6-16
Machine exercise: Debugging an RPG IV program	6-18
Source debugger summary	6-19
Features of integrated debug	6-21
Starting the debug server	6-22
Example of integrated debug	6-23
Tools	6-24
Debug and dump	6-25
Sample DUMP output	6-27
STRSRVJOB and TRCJOB	6-28
Using the joblog as a debugging tool (1 of 2)	6-30
Using the joblog as a debugging tool (2 of 2)	6-31
Checkpoint	6-32
Unit summary	6-33
 Unit 7. Structured programming and subroutines	 7-1
Unit objectives	7-2
Introducing structured programming	7-3
IF	7-4
IF and ELSE	7-6
Make your expressions clear	7-7
AND/OR in expressions	7-8
Nested structures groups	7-9
Nested IF/ELSE	7-10
SELECT/WHEN/OTHER	7-11
IF versus SELECT group	7-12
DoWhile and DoUntil	7-13
DoWhile	7-14

DoUntil	7-15
For/EndFor (1 of 2)	7-16
For/EndFor (2 of 2)	7-17
LEAVE and ITER	7-18
Subroutines	7-19
Coding subroutines	7-20
Unconditional execution of a subroutine	7-21
Conditional branching to a subroutine	7-22
Implicit execution of a subroutine	7-23
LEAVESR	7-24
Calls in RPG IV	7-25
The calling program	7-26
The called program: Return to caller	7-27
Setting *INLR	7-28
Machine exercise: Coding subroutines	7-29
Checkpoint	7-30
Unit summary	7-31
Unit 8. Accessing the DB2 database using RPG IV	8-1
Unit objectives	8-2
DB file: Physical	8-3
DB file: Logical	8-4
DB file descriptions	8-6
DDS limits	8-8
Physical file DDS keywords	8-9
Logical file DDS keywords	8-10
DDS field reference file	8-11
DDS: PF/LF	8-12
Creating DB files and entering data	8-13
Helpful commands	8-14
File operations	8-16
File-related BiFs and extenders	8-18
File open and close: Explicit or implicit?	8-20
Explicit open and close	8-21
Initial open at program start	8-22
Open data path	8-23
Data can be accessed	8-24
Sequential processing: READ	8-25
Sequential processing: READ example	8-26
Sequential processing: Record addition with write	8-27
Random processing: WRITE by RRN	8-28
Position file cursor using SETLL	8-29
Position file cursor using SETLL: Example	8-30
Random processing: CHAIN by key	8-32
Random processing using CHAIN by record key: Example	8-33
Random processing composite key (1 of 2)	8-35
Random processing composite key (2 of 2)	8-36
Random processing using CHAIN by RRN	8-37

Random processing using CHAIN by RRN: Example	8-38
Sequential processing: READP	8-39
Sequential processing: READE	8-40
Sequential processing: READE example	8-41
Positioning file cursor: SETGT	8-42
READP and SETGT	8-43
READPE	8-44
UPDATE	8-45
Database record locks	8-46
i file lock states	8-47
Record locking	8-48
Read update file without locking record	8-49
Releasing a locked record	8-50
Considerations when releasing locked record	8-51
DELETE	8-52
Deleted records	8-53
Machine exercise: Maintaining database files	8-54
Checkpoint	8-55
Unit summary	8-56
 Unit 9. Coding inquiry programs	 9-1
Unit objectives	9-2
Display file descriptions	9-3
Display record format	9-4
Display design (1 of 5)	9-5
Display design (2 of 5)	9-6
Display design (3 of 5)	9-7
Display design (4 of 5)	9-8
Display design (5 of 5)	9-9
Display attribute (DSPATR)	9-11
DSPATR and color displays	9-12
Enabling function keys	9-13
Function keys	9-14
Field-level keywords	9-15
Validity checking keywords	9-16
DDS message keywords	9-17
Message coding examples	9-19
Creating and using screen formats	9-20
Tools	9-21
RPG screen operations	9-22
Inquiry example: Display design	9-23
Inquiry example: Display file DDS	9-24
Inquiry example: Item inquiry program	9-26
Machine exercise: Coding an inquiry program	9-28
Checkpoint	9-29
Unit summary	9-30

Unit 10. What's next?	10-1
Unit objectives	10-2
What have you learned?	10-3
To do list	10-4
Topics list for next course	10-5
Useful references and web sites	10-6
Unit summary	10-7
Appendix A. RPG IV style guide	A-1
Appendix B. Checkpoint solutions	B-1
Bibliography	X-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AS/400®	AS/400e™	DB™
DB2®	Integrated Language Environment®	iSeries®
Iterations®	i5/OS™	Language Environment®
OS/400®	Power Systems™	Power Systems Software™
Power®	Rational®	Redbooks®
RPG/400®	System i®	WebSphere®
400®		

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Course description

RPG IV Programming Fundamentals Workshop for IBM i

Duration: 4 days

Purpose

This course teaches the basics of the IBM i RPG IV programming language. It is the first of three courses that should be attended in sequence.

This course is a comprehensive exposure to the basic features and functions of RPG IV for Version 7. It does not introduce either information processing or programming in general. Students who are new to programming should attend other courses that are offered by local technical colleges or self-study methods.

This course is designed to enable a trained programmer to develop and maintain simple RPG IV programs written using the latest features and techniques available in the Version 7 compiler.

Audience

This course is the first in a series of three courses designed for programmers who are new to RPG IV. Basic programming experience is required. The student should have attended at least an introductory course to programming such as one of those available at technical colleges. The student is not taught the concepts of programming.

Experienced programmers who are new to the Power System with IBM i should also attend this course. Examples of other programming languages are BASIC, COBOL or RPG II.

This course is *not* designed for RPG III programmers who want to learn RPG IV. The *Moving from RPG/400 to System i RPG IV* course (OE85/OE850) is designed to satisfy this need, when it is available in your geography. This course is too basic for experienced RPG III programmers.

If OE85/OE850 is not offered in your geography, you should attend this course instead.

Notes:

The term *RPG/400* refers to both System/38 RPG as well as IBM i (AS/400) RPG/400 (also known as RPG III).

This course focuses entirely on the features of the RPG IV Version 7 compiler and the techniques that this compiler provides. Features of V7 are discussed.

Previous techniques and the maintenance of programs written using legacy techniques, such as fixed format calculations, are not covered in the classroom. Some additional material and the bibliography will assist the new RPG IV programmer in the maintenance of legacy applications.

Prerequisites

Before attending this course, the students should be able to:

- Use a Windows-based PC
- Run PC applications using menus, icons, toolbars, and so forth
- Write a simple program in another high-level language (for example, CL, COBOL, BASIC or RPG II)
- Use basic IBM i tools, including:
 - CL commands
 - Online Help
 - WRKSPLF and related commands to manage output
 - WRKJOB, DSPMSG, DSPJOB commands and so forth to perform basic problem determination
- Use and display IBM i print queues
- Use the Program Development Manager/source entry utility or the RSE/LPEX Editor to create and maintain DDS
- Create and maintain physical and logical files

Students must have attended these courses:

- *Introduction to IBM i for New Users* (OE98/OE980)
- *System i Application Programming Facilities Workshop* (OL49/OL490)

Attendance at *IBM i RPG Development with Rational Developer for Power Systems Software V8* (RN500) is strongly recommended. Experience with Printer and Display files prior to attending this course is beneficial as well.

Objectives

After completing this course, you should be able to:

- Write RPG IV version 7 programs to produce reports
- Write simple RPG IV version 7 inquiry programs that interact with displays
- Review compilation listing, find and correct compilation errors
- Maintain existing applications written in the RPG IV (version 7) language
- Use the debugger tool to determine the cause of incorrect results
- Use many popular RPG IV built-in functions

Contents

1. RPG IV introduction
2. Coding specifications for RPG IV
3. Data representation and definition
4. Manipulating data in calculations
5. Printing from an RPG IV program
6. Using the debugger
7. Structured programming and subroutines
8. Accessing the DB2 database using RPG IV
9. Coding inquiry programs
10. What's next?

Curriculum relationship

This course is the first of three courses designed to develop new RPG IV programmers. It is part of the IBM i programming curriculum to support the RPG IV Version 7 compiler. Below are others that you should attend to enhance your skills in developing applications using the RPG IV language after you have completed this course. Please contact your local IBM Education Center regarding the availability of these classes in your area:

- AS07/AS070 *RPG IV Programming Intermediate Workshop for IBM i*
- AS10/AS100 *RPG IV Advanced Programming Workshop for IBM i*

Agenda

Day 1

- Welcome and administration
- Unit 1: RPG IV introduction
- Lab 1: Coding and compiling RPG IV
- Unit 2: Coding specifications for RPG IV
- Lab 2: Sequencing RPG IV specifications and compiling
- Lab 3: Coding a report program

Day 2

- Lab 4: Adding overflow
- Unit 3: Data representation and definition
- Lab 5: Data definition
- Unit 4: Manipulating data in calculations
- Lab 6: Adding arithmetic function
- Lab 7: Data manipulation

Day 3

- Unit 5: Printing from an RPG IV program
- Lab 8: Printing from an RPG IV program
- Unit 6: Using the debugger
- Lab 9: Debugging an RPG IV program
- Unit 7: Structured programming and subroutines

Day 4

- Unit 7: Structured programming and subroutines (cont.)
- Lab 10: Coding subroutines
- Unit 8: Accessing the DB2 database using RPG IV
- Lab 11: Maintaining database files
- Unit 9: Coding inquiry programs
- Lab 12: Coding an inquiry program
- Unit 10: What's next?

Unit 1. RPG IV introduction

What this unit is about

This unit introduces RPG IV programming terms and concepts. It overviews RPG IV and the basics of the language. It also discusses the steps involved when coding RPG IV programs. A sample program is presented in this unit.

What you should be able to do

After completing this unit, you should be able to:

- Explain the general purpose of each RPG IV specification type
- Describe the steps to code and test an RPG IV program
- List several unique features of the RPG language and discuss RPG's strengths and weaknesses
- Browse an RPG IV compiler listing

How you will check your progress

Accountability:

- Checkpoint questions
- Machine exercise
- Given a simple RPG IV program, the class enters the source member using either the RSE/LPEX or SEU editor and compiles the program. The students are exposed to the editor and a compilation listing.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Explain the general purpose of each RPG IV specification type
- Describe the steps to code and test an RPG IV program
- List several unique features of the RPG language and discuss RPG's strengths and weaknesses
- Browse an RPG IV compiler listing

© Copyright IBM Corporation 2012

Figure 1-1. Unit objectives

AS067.0

Notes:

RPG IV's legacy

IBM i

- Most popular high-level language on i (System i)
- Column-oriented coding
- Fixed format calculations (logic)
- Logic cycle
- Number indicators
- LR (last record)

© Copyright IBM Corporation 2012

Figure 1-2. RPG IV's legacy

AS067.0

Notes:

Report Program Generator (RPG) IV (fourth generation) is by far the most commonly used programming language on the AS/400, AS/400e, and i systems.

You probably have heard many things about the language. Many of the facts that you have heard or read do not apply to the latest version of the RPG IV compiler, version 7.

Version 7 is the one that we use in this class.

What is RPG IV?

IBM i

- Available beginning with OS/400 Version 3 Release 1
- Component of IBM Rational Development Studio for i program product (5770-WDS)
- Expanded or eliminated language limits
- Fulfilled RPG programmers' requirements at that time:
 - Longer field names
 - Free form expressions
 - Date and time calculations support
- Positioned for future growth
- ILE:
 - Part of OS/400 beginning with version 2 release 3
 - Provides toolset to support and encourage modular code development

© Copyright IBM Corporation 2012

Figure 1-3. What is RPG IV?

AS067.0

Notes:

RPG IV is named ILE RPG. Let us break ILE RPG up into the two components that they really are. ILE is the development environment and RPG IV is the programming language.

The Integrated Language Environment (ILE) provides the facilities that enable RPG IV programs to be written in a very modular fashion without the same level of overhead required by RPG III. Modules of code can be written in a generic, reusable manner so that they can be shared by many application programs. ILE helps improve application execution efficiency and programmer productivity.

RPG IV is the name that IBM has given the latest version of the RPG language. It is packaged as part of the *ILE RPG for i* compiler. This package of software includes RPG IV and other support, such as support for RPG/36. We might use the term *RPG/400* in this class. We try to differentiate between RPG IV features and features that were included in the previous version of the language (known as RPG III). All previous versions of RPG on the i that were packaged as part of RPG/400 are now a part of the ILE RPG for i package.

The term **ILE RPG** can be confusing. The RPG IV compiler, officially named IBM Integrated Language Environment RPG IV for i is one component of the WebSphere Development Studio for i (Version 7), Program Number 5770-WDS.

Integrated Language Environment (ILE) RPG for i is the IBM implementation of RPG on the i and AS/400e system. Also included in 5770-WDS are:

- RPG/400
- System/36-Compatible RPG II
- System/38-Compatible RPG III
- ILE RPG IV Previous compiler

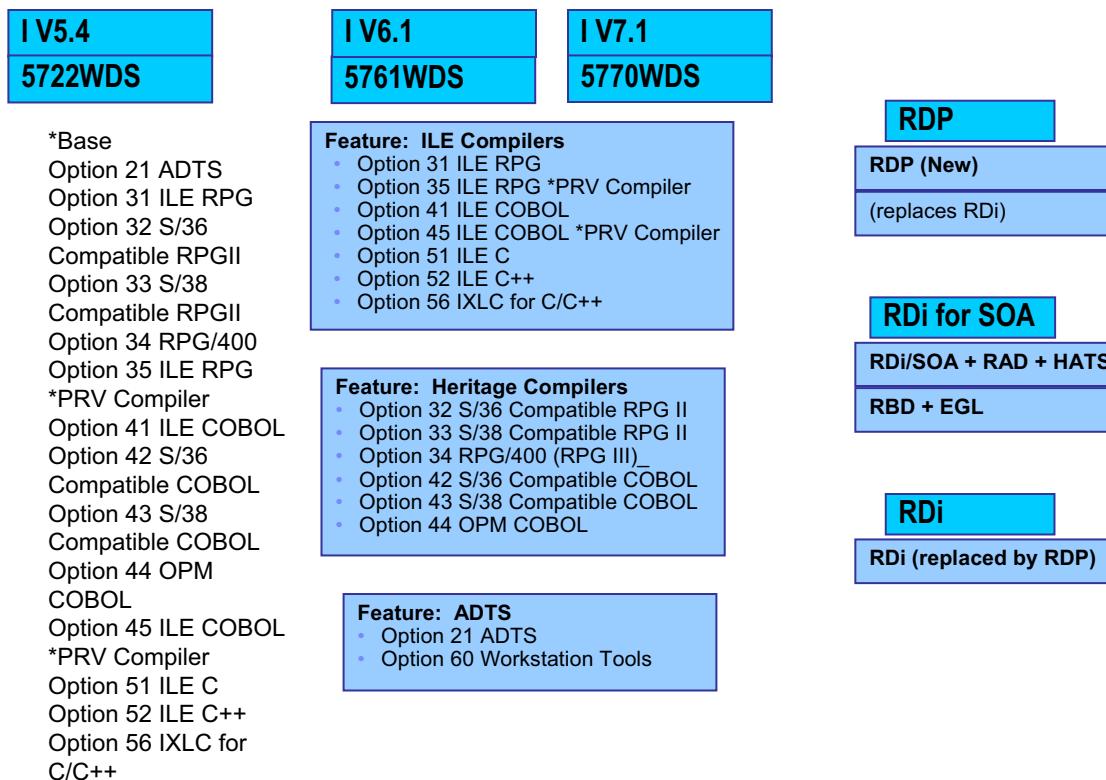
The Rational Development Studio for i packages the ILE RPG IV for i compiler component together with other development components. Users can select to install those components that best fits their application development needs.

You do *not* have to use the ILE features of binding, subprocedures, and so forth in order to write RPG IV programs that work and perform the tasks you require.

The RPG/400 component above is the RPG III for i compiler. This class discusses the development of RPG IV programs only.

Application development tools: Packaging

IBM i



© Copyright IBM Corporation 2012

Figure 1-4. Application development tools: Packaging

AS067.0

Notes:

There is one application development product available for i programmers. The product is called the WebSphere Development Studio for i (5770-WDS) which includes:

i compilers:

- ILE RPG
- ILE COBOL
- ILE C
- ILE C++

You install only the compilers and features that you need for your installation. The difference is that now all compilers and tools are included as a single packaged offering.

Application Development ToolSet:

- Source entry utility (SEU)
- Screen design aid (SDA)
- Report layout utility (RLU)
- Data file utility (DFU)

- Character generator utility (CGU)
- Advanced printer function (APF)
- Programming Development Manager (PDM)
- Interactive source debugger (ISDB)
- File compare and merge utility (FCMU)

The WebSphere Development Studio (WDS) also includes limited licenses to use the workstation-based toolset named Rational Development for Power Systems (RDP). You now have a choice of Rational Developer for Power Systems (RDP) or Rational Developer for System i (RDi). WDS at IBM i releases 5.4, 6.1 and 7.1 all cooperatively work with the client PC programs RDP and RDi.

RDP packages the following:

- LPEX Editor
- Remote System Explorer
- Debugger
- i Project Organizer
- Screen Designer
- Report Designer

RDi includes the following:

- LPEX Editor
- Remote System Explorer
- Debugger

Rational Developer for i for SOA Construction is additional PC-licensed code that, when purchased, adds the following features:

- Rational Business Developer
- Host Access Transformation Services (HATS) Toolkit
- Rational Application Developer (RAD)
- Rational Business Developer (RBD)
- Enterprise Generation Language (EGL)

RDi for SOA can be used to modernize existing RPG and COBOL business applications to the web, mobile devices or into web services.

Starting with IBM i 6.1, WDS was packaged into three separately priced optional features: ILE compilers, heritage compilers, and ADTS (legacy development tools). Modernized shops now have the option of ordering the appropriate i optional features as well as the appropriate PC client software. Now shops can choose RDP or RDi to use with WDS and RDi SOA (at additional cost).

Entitlement on 5761-WDS and 5770-WDS ILE Compilers, Heritage Compilers, and ADTS features are based on a maximum number of users per processor tier.

Development cycle

IBM i

- Define the problem
- Design the solution
- Write the program
- Test and debug the program
- Document the program
- Maintain the program

© Copyright IBM Corporation 2012

Figure 1-5. Development cycle

AS067.0

Notes:

Developers develop a solution to solve a business problem. This visual describes a simplistic methodology for developing a solution and maintaining that solution over time.

1. The program development cycle begins with understanding the problem.
2. When you understand the business problem, you can design a solution to the problem.
3. Your design must then be expressed in RPG IV code. It is good practice to use formal design tools to express the solution prior to attempting to write your RPG IV program. Flow charts, Warnier-Orr diagrams or pseudocode are all valid choices for documenting the logic of your program. Here is where most of the heavy thinking takes place. Spend the time here to develop a correct, well-structured design that addresses the problem. Time spent here could well result in time saved fixing problems later.

Writing the program involves translating your design into corresponding RPG IV specifications. Inexperienced developers find this difficult, at first, because they are unfamiliar with language rules and syntax. This part of the process becomes more

mechanical and requires less thought with time. Most of your thinking is in the design stage...not the writing.

Typing the program involves using an editor to input program statements into the computer. Again, the more you use the editor, the more familiar this tool becomes.

4. Testing the program is next. What are you looking for during this test? First, look for any syntax errors. Syntax errors are errors in the use of the language. These errors are often discovered by the editor as you key the source statements. Those that happen to get past the editor are surely flagged by the compiler during compile attempts. Next, look for logic errors. Logic errors are errors of design. The programmed solution does not work as intended or does not perform the required functions. It is the programmer's responsibility to verify the accuracy of output from the program. To debug your programs is to correct the errors found.
5. Documenting the program can take many forms. Internal documentation in the form of comments in the code can be very helpful to other programmers responsible for later maintenance of this program. Program flowcharts, application manuals and operator instructions could also provide valuable information for later maintenance.
6. When the program has been designed, written, tested, and debugged, it is then ready to be used. Actual use of the code in live situations often leads to the need for fixes or corrections. Maintenance is a fact of life; so remember to document your programs well to make the maintenance effort as painless as possible for yourself or anyone else.

Describing a programming solution

IBM i

- Print headings
- Read an employee record
- Do while more records exist:
 - Count each employee record read
 - Write detailed employee information
 - Read an employee record
- Print total number of employees
- End program

© Copyright IBM Corporation 2012

Figure 1-6. Describing a programming solution

AS067.0

Notes:

Suppose you were asked to write a program that prints a report that displays one line item for each employee on staff, listing each employee's name, serial number, department and employee type. After all employee records were read, your report should print a total count of employees.

As we show in this visual, you might first express the problem using pseudocode in order to structure and think through a solution.

Some tool would need to be used to depict the desired report layout, perhaps a printer layout form or a GUI tool such as the RDP Report Designer.

The employee data file documentation would provide the needed data specifications for the record formats.

RPG IV terminology

IBM i

- Indicators
- Reserved words
- Procedure
- Fields
- Procedural processing
- Page overflow
- Figurative constants
- Specifications
- Opcodes



© Copyright IBM Corporation 2012

Figure 1-7. RPG IV terminology

AS067.0

Notes:

Modern RPG IV programs use procedural logic to manipulate data records. You control how the fields (RPG IV variables) from the record are manipulated by coding operation codes in your program. Operation codes can be coded inline or, better, in reusable units such as subroutines or subprocedures.

Field naming follows the rules of symbolic naming; a field can be up to 4096 characters long (usually 10 - 14 characters in reality). The first character of the name must be alphabetic (A through Z, \$, #, @) with the remaining characters alphabetic (A through Z, \$, #, @, _(underscore)) or numeric (0 through 9). Blanks are not allowed.

RPG IV provides a number of productive features to ease the coding effort. Indicators and figurative constants are two such features.

The indicators that we use in this class are called *named indicators*. Named indicators are not the numbered indicators you may have heard about or even seen. They are actual variable names that you reference in your logic. Because they have names, your code is much more easily understood and thus more easily maintained.

RPG IV specifications

IBM i

- Seven types of specifications:
 - (H) Control specifications
 - (F) File description specifications
 - (D) Definition specifications
 - (I) Input specifications
 - (C) Calculation specifications
 - (O) Output specifications
 - (P) Procedure specifications
- Specific functions
- Optional
- Fixed-position and free-form

© Copyright IBM Corporation 2012

Figure 1-8. RPG IV specifications

AS067.0

Notes:

RPG IV programs consist of various types of lines of code, called specifications. Each specification has a specific purpose. Some specifications are characterized by fixed-position entries, meaning that the location of an entry within a line of code is critical to its interpretation. Some specifications, however, allow free-form entries through the use of keywords.

The overall order of the specification types within the program is critical. If they are not grouped in the proper order when the program is compiled, the program object is *not* created. The order of specifications in a program must be as follows:

- H
- F
- D
- I
- C

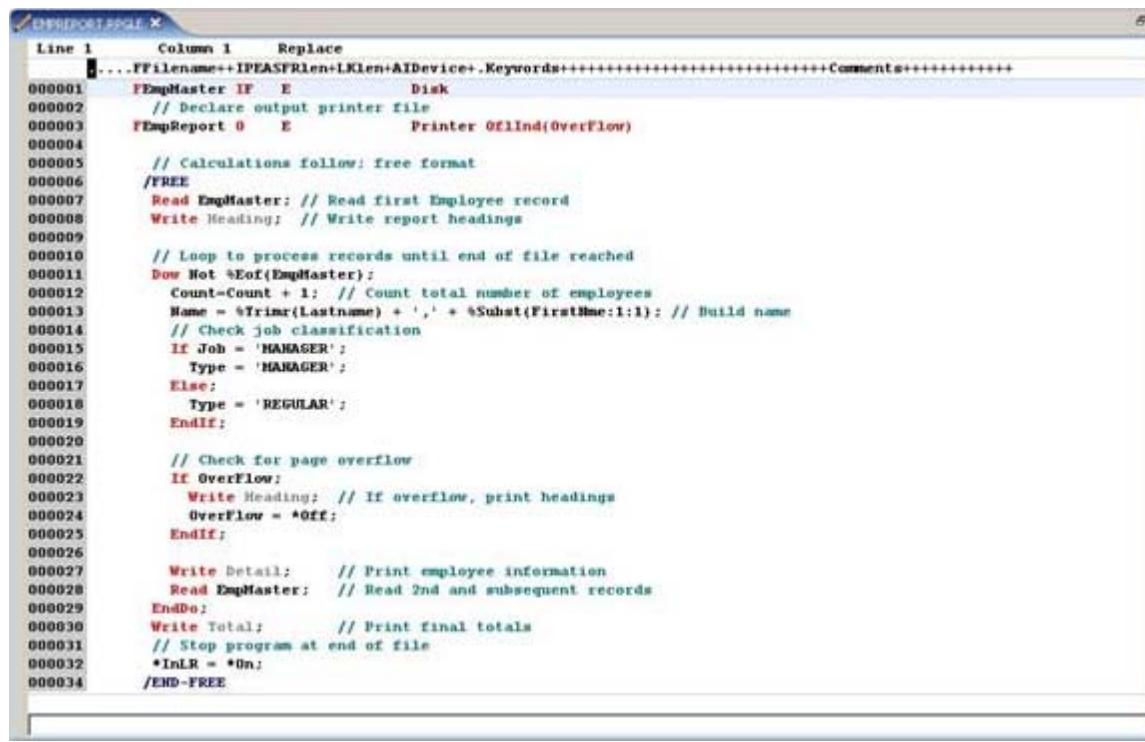
- O
- P

H, F, D and C specs are used throughout this class. Because all files are externally described, you never code I(nput) or O(utput) specifications in this class. The compiler generates them for you based on external file and format definitions. Your compilation listings show you all coding specifications, both coded and generated.

P(rocedure) specifications are used to write modular code that can be accessed (or called) by other programs. You learn about subprocedure and use the P-spec in the classes that follow this one.

Print program: RPG code

IBM i



```

Line 1      Column 1      Replace
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++Comments+++++
000001    FEmpMaster IF   E           Disk
000002        // Declare output printer file
000003    FEmpReport 0   E           Printer OffInd(Overflow)
000004
000005        // Calculations follow; free format
000006 /FREE
000007     Read EmpMaster; // Read first Employee record
000008     Write Heading; // Write report headings
000009
000010    // Loop to process records until end of file reached
000011    Dow Not *Eof(EmpMaster);
000012        Count=Count + 1; // Count total number of employees
000013        Name = *Trimr(Lastname) + ',' + *Subst(FirstName:1:1); // Build name
000014        // Check job classification
000015        If Job = 'MANAGER';
000016            Type = 'MANAGER';
000017        Else;
000018            Type = 'REGULAR';
000019        Endif;
000020
000021        // Check for page overflow
000022        If Overflow;
000023            Write Heading; // If overflow, print headings
000024            Overflow = *Off;
000025        Endif;
000026
000027        Write Detail; // Print employee information
000028        Read EmpMaster; // Read 2nd and subsequent records
000029    EndDo;
000030    Write Total; // Print final totals
000031    // Stop program at end of file
000032    *InLR = *On;
000033
000034 /END-FREE

```

© Copyright IBM Corporation 2012

Figure 1-9. Print program: RPG code

AS067.0

Notes:

This visual is a programming interpretation of the example we discussed using the pseudocode description earlier in the notebook. This is a basic print program:

- Each record of the EMPMASTER file is read.
- A copy of each record read is printed on a report.
- The program continues until end of file is reached.
- At end of file EMPMASTER, a total record is written on the report and the end of program indicator known as last record (*InLR) is set on.

Print program: Report

IBM i

PAGE	1	EMPLOYEE INFORMATION		
NAME	SERIAL #	DEPT	TYPE	
Haas,C	000010	A00	REGULAR	
Thompson,M	000020	B01	MANAGER	
Kwan,S	000030	C01	MANAGER	
Geyer,J	000050	E01	MANAGER	
Stern,I	000060	D11	MANAGER	
Pulaski,E	000070	D21	MANAGER	
Henderson,E	000090	E11	MANAGER	
Spenser,T	000100	E21	MANAGER	
Lucchesi,V	000110	A00	REGULAR	
:	:	:	:	
:	:	:	:	
:	:	:	:	
Marino,S	000240	D21	REGULAR	
Smith,D	000250	D21	REGULAR	
Johnson,S	000260	D21	REGULAR	
Perez,M	000270	D21	REGULAR	
Schneider,E	000280	E11	REGULAR	
Parker,J	000290	E11	REGULAR	
Smith,P	000300	E11	REGULAR	
Setright,M	000310	E11	REGULAR	
Mehta,R	000320	E21	REGULAR	
Lee,W	000330	E21	REGULAR	
Gounot,J	000340	E21	REGULAR	

NUMBER OF EMPLOYEES 32

© Copyright IBM Corporation 2012

Figure 1-10. Print program: Report

AS067.0

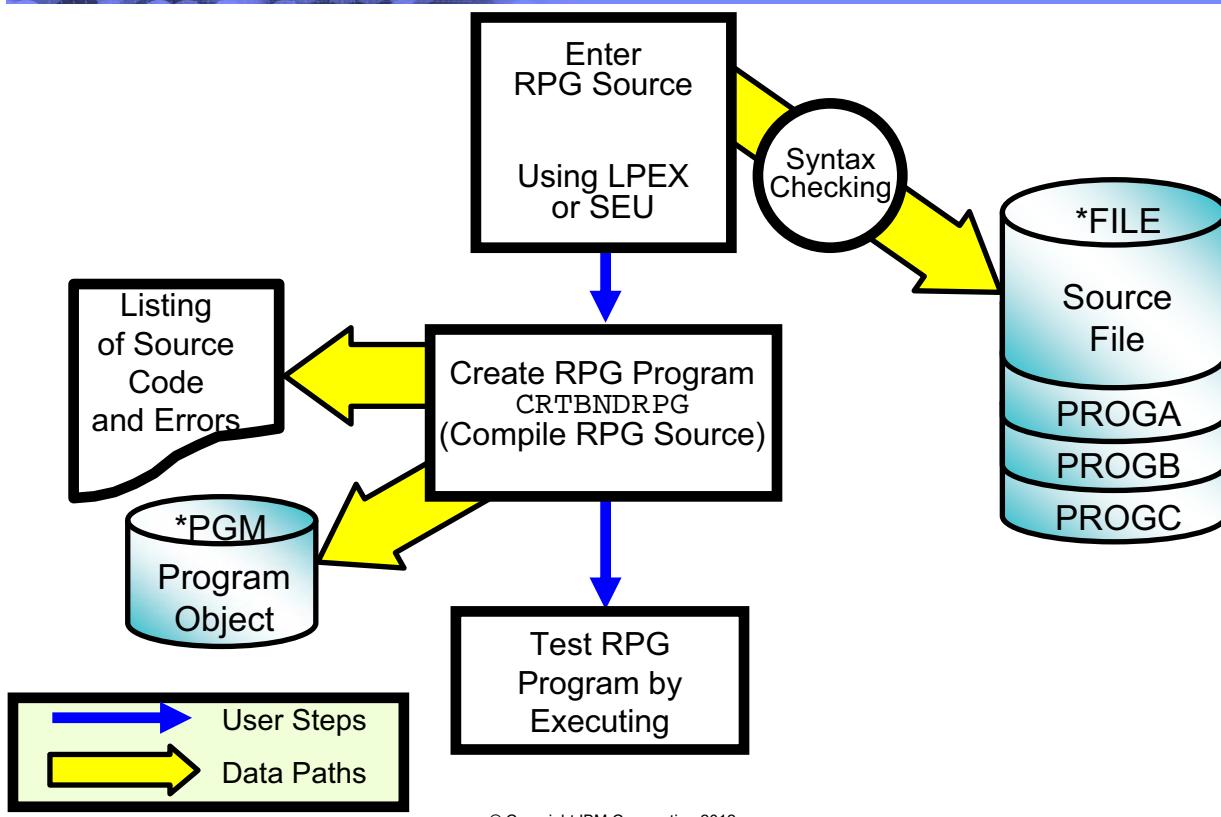
Notes:

The output from our program might appear as above. The layout of the report EMREPORT is defined using data description specifications, (DDS).

We discuss DDS in a later unit.

Create RPG IV program

IBM i



© Copyright IBM Corporation 2012

Figure 1-11. Create RPG IV program

AS067.0

Notes:

You enter an RPG IV program as source code using the system editor, source entry utility (SEU), or the Rational Developer for Power Systems editor, LPEX.

When your program has been entered and saved, it can be translated into machine-readable object code. Syntax checking of your program occurs while you are keying.

A unique feature of the LPEX Editor is that you can verify that your program compiles successfully, *before invoking the compiler on the i*.

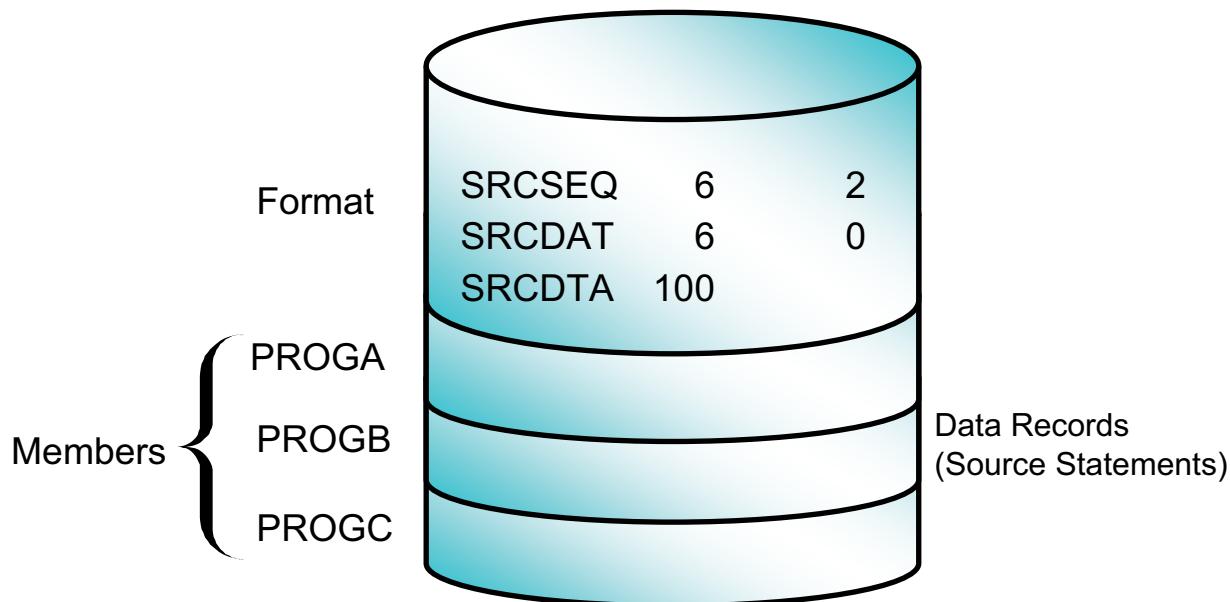
The translation of your source code into an executable program object (*PGM) is accomplished by compiling your program. If your program has no compilation errors, your program successfully compiles and an executable object is created.

Now, you must test your program. If there are no logic errors, you have completed your task. If logic errors do exist, you need to correct them using the editor and recompile the program. This recompile incorporates the changes into the object program.

Later, we introduce a tool, the source debugger, that can be used to help you find and correct logic errors.

IBM i source physical file

IBM i



© Copyright IBM Corporation 2012

Figure 1-12. IBM i source physical file

AS067.0

Notes:

A source physical file is a special file on the i that contains one or more source members. In our case, our source file contains RPG IV source. Notice that the record format of this file is perfect for keying source statements. Statement number, date entered, and a field for source statement data are all you need to enter various kinds of source statements.

The CL command CRTSRCPF creates a similar source file in the library of your choice. For example:

Create Source Physical File (CRTSRCPF)

Type choices, press Enter.

File	<u>QRPGLSRC</u>	Name
Library	AS0600LIB	Name, *CURLIB
Record length	<u>112</u>	Number
Member, if desired	*NONE	Name, *NONE, *FILE
Text 'description'	ILE RPG IV source	

F3=Exit F4=Prompt F5=Refresh F10=Additional parameters F12=Cancel
F13=How to use this display F24=More keys

This command will create a source file for RPG IV source members in the AS0600LIB library.

When you create a new member, for example, ITEMPINQ, in the QRPGLESRC source file, you will see a display similar to the following after pressing **F6**:

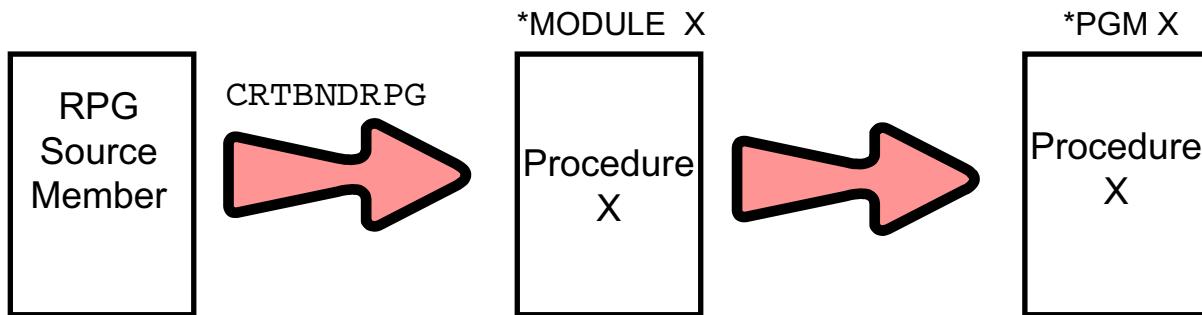
Start Source Entry Utility (STRSEU)

Type choices, press **Enter**.

The process of creating an RPG IV program

IBM i

- ILE programs might contain one or more modules.
- For single module programs: CRTBNDRPG.



© Copyright IBM Corporation 2012

Figure 1-13. The process of creating an RPG IV program

AS067.0

Notes:

The CRTBNDRPG command creates an executable program object using one command. Use this command only in this class to create your executable program objects.

Option 14 of the PDM Work with Members screen invokes CRTBNDRPG for source members type RPGL. You can also click the compile option in LPEX and specify the CRTBNDRPG command. You can set this command as your individual default.

Look at the figure above and notice the object *MODULE. A module is an i object type for RPG IV as well as other compilers. The *MODULE object that is created as part of the CRTBNDRPG command is automatically deleted after the *PGM object has been created.

Each RPG IV module contains what is called a Procedure, which is the entry point into the executable code contained in that module.

There is another method of creating RPG IV programs that involves creating programs from one or more modules. The classes that follow this one discuss modules and how to combine them to create programs.

CRTBNDRPG compile options

IBM i

- GENLVL (10)
- OPTION () :
 - *XREF/*NOXREF
 - *GEN/*NOGEN
 - *SHOWCPY/*NOSHOWCPY
 - *NOSECLVL/*SECLVL
 - *EXPDDS/*NOEXPDDS
 - *EXT/*NOEXT
- DBGVIEW (*STMT)
- OUTPUT (*PRINT/*NONE)
- INDENT (*NONE)
- REPLACE (*YES)
- TRUNCNBR (*YES)
- TGTRLS (*CURRENT)

© Copyright IBM Corporation 2012

Figure 1-14. CRTBNDRPG compile options

AS067.0

Notes:

Default values are shown for each parameter. For the **OPTION** parameter, the first value of each pair is the default.

GENLVL specifies whether a program object is generated, depending on the severity level of the errors encountered. Valid entries that act as program generation filters are 1 - 20.

OPTION specifies the options to use when the source member is compiled. You can specify any or all of the options in any order. Separate the options with one or more blank spaces.

***XREF** produces a cross reference listing.

***GEN** will create a program object if the severity level specified on **GENLVL** is not exceeded.

***SECLVL** prints second-level message text on the line following the first-level message text in the Message Summary section of the compile listing.

***SHOWCOPY** includes any source members copied into the compiled member using the /COPY directive.

***EXPDDS** shows the expanded definitions of externally described files.

***EXT** shows a listing of external procedures and fields.

DBGVIEW sets which level of debugging is available. The choices are *STMT, *SOURCE, *LIST, *COPY, *ALL and *NONE. The corresponding view is then available upon debugging the compiled object.

OUTPUT determines if a compile listing is produced.

INDENT indents the source code to highlight structured coding techniques.

REPLACE allows automatic deletion and replacement of a previously compiled program without a reminder message.

TRUNCNBR specifies whether a truncated value is moved to the result field or if an error is generated when numeric overflow occurs while running the program. This parameter has *no* effect on calculations done in expressions. We discuss expression calculations and the precision of those calculations in a later unit.

TGTRLS specifies the release level of the IBM i, i5/OS (OS/400) system on which you intend to run the program object being generated.

Compiler listing

IBM i

Listing Section	Option	Description
Prologue		Compile option summary
Source listing: In-line diagnostic messages /COPY members Externally described files	*SHOWCPY *EXPDDS	Source specifications error by line /COPY source members record, field, specifications
Additional diagnostic messages		Errors spanning multiple lines
Output buffer positions /COPY member table		Program described output specs list of copy members
Arrays		Array records
Tables		Table records
Key field information	*EXPDDS	Key field attributes
Cross reference table	*XREF	File, record, field indicator use
External references	*EXT	List of external procedures/fields
Message summary Second level test	*SECLVL	Messages, number of occurrences second level message text
Final summary		Final compilation message
Code generation errors		Error during code generation

© Copyright IBM Corporation 2012

Figure 1-15. Compiler listing

AS067.0

Notes:

Compiler listings provide you with information regarding the correctness of your code with respect to language syntax and semantics. They are designed to help you detect and correct any errors through a source editor. This chart summarizes compiler keywords with their associated listing information.

The following compiler directives can be included in your source code in order to customize your compile listings:

/TITLE specifies heading information at the top of listing pages:

```
/TITLE Your text for title information
```

/SPACE specifies line spacing within the source section of listing:

```
/SPACE xxx (where xxx = positive integer 1 - 112)
```

/COPY copies/inserts records from other files into this member:

```
/COPY MYLIB/MYFILE,mymember
```

/EJECT sets subsequent specs to begin on a new page of the listing.

Compiler directives

IBM i

- Format the listing:
 - /TITLE
 - /EJECT
 - /SPACE
- Include other members in the compilation
 - /COPY
- Conditionally include / exclude code:
 - /DEFINE
 - /UNDEFINE
- Free format coding group:
 - /FREE
 - /END-FREE
- Documentation / readability
 - * or //

From version 5, use // for all comments

© Copyright IBM Corporation 2012

Figure 1-16. Compiler directives

AS067.0

Notes:

The compiler directive statements /TITLE, /EJECT, /SPACE, and /COPY allow you to specify heading information for the compiler listing, to control the spacing of the compiler listing, to insert records from other file members during a compile, and to select or omit source records.

/TITLE (Positions 7-12)

Use the compiler directive /TITLE to specify heading information (such as security classification or titles) that is to appear at the top of each page of the compiler listing. The following entries are used for /TITLE:

Positions	Entry
-----	-----
7-12	/TITLE
13	Blank
14-100	Title information

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until another /TITLE statement is encountered. A /TITLE statement must be the first RPG specification encountered to print information on the first page of the compiler listing. The information specified by the /TITLE statement is printed in addition to compiler heading information.

The /TITLE statement causes a skip to the next page before the title is printed. The /TITLE statement is not printed on the compiler listing.

/EJECT (Positions 7-12)

Positions	Entry
-----	-----
7-12	/EJECT
13-49	Blank
50-100	Comments

Enter /EJECT in positions 7 through 12 to indicate that subsequent specifications are to begin on a new page of the compiler listing. Positions 13 through 49 of the /EJECT statement must be blank. The remaining positions can be used for comments. If the spool file is already at the top of a new page, /EJECT will not advance to a new page. /EJECT is not printed on the compiler listing.

/SPACE (Positions 7-12)

Use the compiler directive /SPACE to control line spacing within the source section of the compiler listing. The following entries are used for /SPACE:

Positions	Entry
-----	-----
7-12	/SPACE
13	Blank
14-16	A positive integer value from 1 through 112 that defines the number of lines to space on the compiler listing. The number must be left-adjusted.
17-49	Blank
50-100	Comments

If the number specified in positions 14 through 16 is greater than 112, 112 will be used as the /SPACE value. If the number specified in positions 14 through 16 is greater than the number of lines remaining on the current page, subsequent specifications begin at the top of the next page.

/SPACE is not printed on the compiler listing, but is replaced by the specified line spacing. The line spacing caused by /SPACE is in addition to the two lines that are skipped between specification types.

/COPY (Positions 7-11)

The /COPY compiler directive causes records from other files to be inserted, at the point where the /COPY occurs, with the file being compiled. The inserted files can contain any valid specification including /COPY up to the maximum nesting depth specified by the COPYNEST keyword on the H-spec (32 when not specified).

The /COPY statement is entered in the following way:

Positions	Entry
-----	-----
7-11	/COPY
12	Blank
13-49	libraryname/filename,membername Identifies the location of the member to be copied (merged).
50-100	Comments

*** (Position 7)**

The comment directive is one that should be used more frequently than any other. An asterisk (*) in column 7 of any specification makes that spec a comment regardless of anything else in the spec.

Conditional Compilation Directives

The conditional compilation directive statements allow you to conditionally include or exclude sections of source code from the compilation.

Condition-names can be added or removed from a list of currently defined conditions using the defining condition directives /DEFINE and /UNDEFINE.

Condition expressions DEFINED(condition-name) and NOT DEFINED(condition-name) are used within testing condition /IF groups.

Testing condition directives, /IF, /ELSEIF, /ELSE and /ENDIF, control which source lines are to be read by the compiler.

The /EOF directive tells the compiler to ignore the remaining lines of code in the current source member.

/DEFINE (Positions 7-13)

The /DEFINE compiler directive defines conditions for conditional compilation. The entries in the condition-name area are free format (do not have to be left justified). The following entries are used for /DEFINE:

Positions Entry**----- -----**

7 - 13	/DEFINE
14	Blank
15 - 80	condition-name
81 - 100	Comments

The /DEFINE directive adds a condition-name to the list of currently-defined conditions. A subsequent /IF DEFINED(condition-name) would be true. A subsequent /IF NOT DEFINED(condition-name) would be false.

/UNDEFINE (Positions 7-15)

Use the /UNDEFINE directive to indicate that a condition is no longer defined. The entries in the condition-name area are free-format (do not have to be left justified).

Positions Entry**----- -----**

7 - 15	/UNDEFINE
16	Blank
17 - 80	condition-name
81 - 100	Comments

The /UNDEFINE directive removes a condition-name from the list of currently-defined conditions. A subsequent /IF DEFINED(condition-name) would be false. A subsequent /IF NOT DEFINED(condition-name) would be true.

/FREE and /END-FREE

The /FREE compiler directive specifies the beginning of a free-form calculation specifications block. /END-FREE specifies the end of the block. Positions 12 through 80 must be blank. The remaining positions can be used for comments.

Machine exercise: Coding and compiling RPG IV

IBM i



© Copyright IBM Corporation 2012

Figure 1-17. Machine exercise: Coding and compiling RPG IV

AS067.0

Notes:

Perform the coding and compiling RPG IV exercise.

Checkpoint (1 of 2)

IBM i

1. True or False: Today's RPG IV programs must use ILE features like binding and subprocedures.
2. In order to create a program from an RPGL E type source member, which of the following answers is correct?
 - a. CRTRPGPGM
 - b. CRTBNDRPG
 - c. CRTRPGMOD and CRTPGM
 - d. CRTRPGMOD
3. The (blank) compiler directives specify the beginning and end of a free-form calculation specification block.
 - a. /DEFINE, /UNDEFINE
 - b. /BEGIN-FREE, /END-FREE
 - c. /FREE, /END-FREE
 - d. /FREE, /END
4. True or False: In order to include code from another source member in your RPG IV program, you would use the /COPY compiler directive.

© Copyright IBM Corporation 2012

Figure 1-18. Checkpoint (1 of 2)

AS067.0

Notes:

Checkpoint (2 of 2)

IBM i

5. Number the following RPG specification types in correct sequence.
 - P Specifications
 - I Specifications
 - F Specifications
 - D Specifications
 - H Specifications
 - O Specifications
 - C Specifications

6. A source physical file for RPG IV source members has a blank byte record length?
 - a. 80
 - b. 92
 - c. 100
 - d. 112

© Copyright IBM Corporation 2012

Figure 1-19. Checkpoint (2 of 2)

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Explain the general purpose of each RPG IV specification type
- Describe the steps to code and test an RPG IV program
- List several unique features of the RPG language and discuss RPG's strengths and weaknesses
- Browse an RPG IV compiler listing

© Copyright IBM Corporation 2012

Figure 1-20. Unit summary

AS067.0

Notes:

Unit 2. Coding specifications for RPG IV

What this unit is about

This unit describes the specifications that must be programmer-coded in the RPG IV language and how each specification is used.

We perform instructor-led desk exercises during lecture to practice coding specifications, and two machine exercises as we progress through the unit.

We also introduce how a program is structured. This unit familiarizes the students with the structure of an RPG IV program.

We show the different specification templates in the process of covering the structure of an RPG IV program. Simple I/O operations such as READ and WRITE are also introduced.

We perform instructor-led desk exercises during lecture to practice coding specifications followed by machine exercise at the end of the unit.

What you should be able to do

After completing this unit, you should be able to:

- Describe the purpose of each RPG IV specification type
- Code specifications in the order required for an RPG IV program to compile successfully
- Describe how to reference externally described database and printer files in RPG IV
- Write a simple listing program

How you will check your progress

Accountability:

- Checkpoint questions
- Machine exercises
- Given a simple RPG IV program, the class correctly sequences an RPG IV source member where the specifications are out of sequence.
- Given a problem statement, the students:

- Code and enter an RPG IV program using the SEU or the LPEX Editor
- Compile the program
- Execute the program
- This program produces a listing of records and places that listing in the student's OUTQ

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe the purpose of each RPG IV specification type
- Code specifications in the order required for an RPG IV program to compile successfully
- Describe how to reference externally described database and printer files in RPG IV
- Write a simple listing program

© Copyright IBM Corporation 2012

Figure 2-1. Unit objectives

AS067.0

Notes:

Source specifications

IBM i

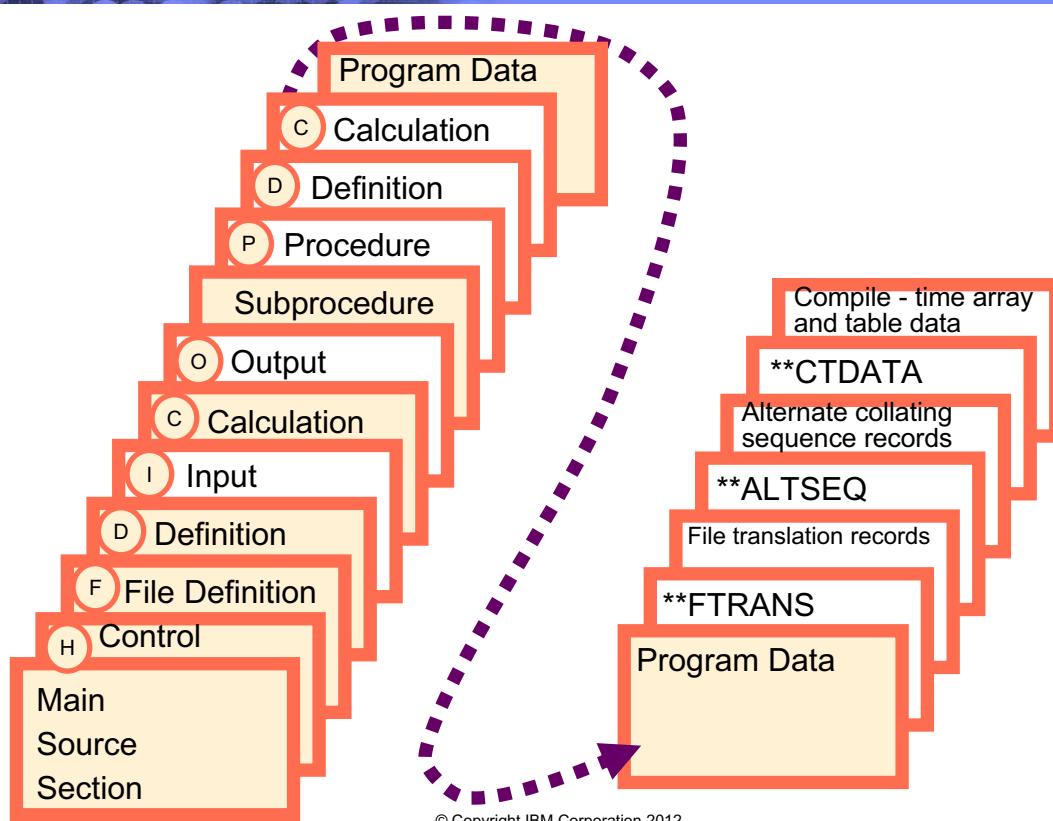


Figure 2-2. Source specifications

AS067.0

Notes:

- Specification type (position 6):
 - H, F, D, I, C, O, P, D, C
 - Must be coded in this order
- Comment line created by placing * in position 7 or by placing // in position 8 or higher (recommended method).
- Keywords are used in many specifications.

Because all files that we use are externally described, we do not spend a lot of time on I(nput) and O(utput) specifications. You are encouraged to review your compiler listings that show the I- and O-specs that are generated by the compiler.

Symbolic names

IBM i

- Symbolic names are used for:
 - Fields
 - Key field lists
 - Labels
 - Named constants
 - Parameter lists
 - Record names
- Other RPG names
- Use *RPG IV Style Guide* suggestions

© Copyright IBM Corporation 2012

Figure 2-3. Symbolic names

AS067.0

Notes:

A symbolic name is a name that uniquely identifies a specific entity in a program or procedure. In the RPG IV language, symbolic names are used to describe the items in the visual. The rules for specifying symbolic names are:

- The first character of the name must be alphabetic (the characters \$, #, and @ are valid).
- The remaining characters must be alphabetic or numeric, including the underscore (_).
- The name must be left-adjusted in the entry on the specification form except in fields that allow the name to float (definition specification, keyword fields, and the extended factor two field).
- A symbolic name cannot be an RPG IV reserved word.
- A symbolic name can be from 1 to 4096 characters.



Note

The practical limits are determined by the size of the entry used for defining the name. A name that is up to 15 characters can be specified in the Name entry of the definition or procedure specification. For names longer than 15 characters, use a continuation specification.

A symbolic name must be unique within the procedure in which it is defined.

RPG IV style tip

In the ITSO Redbook, *Who Knew You Could Do That with RPG IV?*, there is a reference to the *RPG Style Guide*. We will reference it and suggest coding style tips for you to consider. Here are some tips for your RPG:

- Make sure that your names fully and accurately describe an item.
- Use mixed case to enhance the meaning of the name.

Generally, avoid using special characters. However, there are exceptions.

References:

1. *Who Knew You Could Do That with RPG IV?*, Chapter 2, *Programming RPG IV with Style*
2. *ILE Concepts*, Chapter 2, *ILE Basic Concepts*

A copy of the *RPG IV Style Guide* is included as an appendix of this notebook.

Field naming rules

IBM i

- First character must be alphabetic.
- Rest may be alpha, numeric or \$, #, @.
- Name may be up to 4096 in length.
 - Recommend 10 to 14 (based on RPG IV specification where used)
- Mixed case is valid.
 - Use mixed case to create meaningful names.
 - Must be unique: Compiler converts all to upper case.
 - LastName = lastname = LASTNAME.
- If long name is necessary, you may continue using ellipsis (...).

```

FIG44.RPGLE
Line 1   Column 1   Replace   Browse
000001  D ExtraLongFieldNameToContainAlphaChars...
000002  D                                     10    Inz
000003
000004  /Free
000005  ExtraLongFieldNameToContainAlphaChars = 'abcdef';
000006  /End-free

```

© Copyright IBM Corporation 2012

Figure 2-4. Field naming rules

AS067.0

Notes:

This visual emphasizes the rules for naming fields. Although long names are supported, you can see that coding them can be awkward. Not shown in the visual, continuation is done using an ellipsis (...). Free format avoids the need for an ellipsis usually. It would most often be necessary when using fixed format.

The *RPG IV Style Guide* recommends that you limit your names to 10 to 14 positions whenever possible, and that you avoid using any special characters, \$, #, @.

Use mixed case to make the use of the field clear.

RPG IV indicators

IBM i

- Are switches
- Are used to test status of data (+, -, 0, blank, nonblank, >, <)
- Can condition logic flow of calculations and output produced
- Include:
 - 01 - 99 general purpose
 - Special purpose LR (Last Record) represented as *inLR
 - Not necessary to declare explicitly numbered indicators
 - Named indicators can map to numbered indicators in D-Specs - named indicator variables must be defined (except printer overflow)
- Examples:

```

REmpReport O E           PRINTER OflInd(Overflow)

// Check for page overflow
If Overflow;
  Write Heading; If overflow, print headings
  Overflow = *off;
ENDIF;

D Exit
  /FREE
  Draw NOT Exit;
  Read Record;
  EndData
  /END-FREE

```

- Can also be externally defined in DDS (PRTF/DSPF)

© Copyright IBM Corporation 2012

Figure 2-5. RPG IV indicators

AS067.0

Notes:

An indicator is a one position character field that contains one of two possible values:

1. A value of **on**, also known as **true**, represented by a **1**
2. A value of **off**, also known as **false**, represented by **0**

An indicator is generally used to indicate the result of a operation or to condition (or control) the processing of an operation.

Indicators are defined either by an entry on the specification or by the RPG IV program itself. The positions on the specification in which you define an indicator determine how the indicator is used. An indicator that has been defined can then be used to condition calculation and output operations.

The RPG IV program sets and resets certain indicators at specific times during the program cycle. In addition, the state of most indicators can be changed by calculation operations.

The most common ways to set an indicator are by using an expression. Often, you see indicators set on and off using ***on** and ***off**. ***on** and ***off** are RPG IV **figurative constants**.

Machine exercise: Sequencing RPG IV specifications and compiling

IBM i



© Copyright IBM Corporation 2012

Figure 2-6. Machine exercise: Sequencing RPG IV specifications and compiling

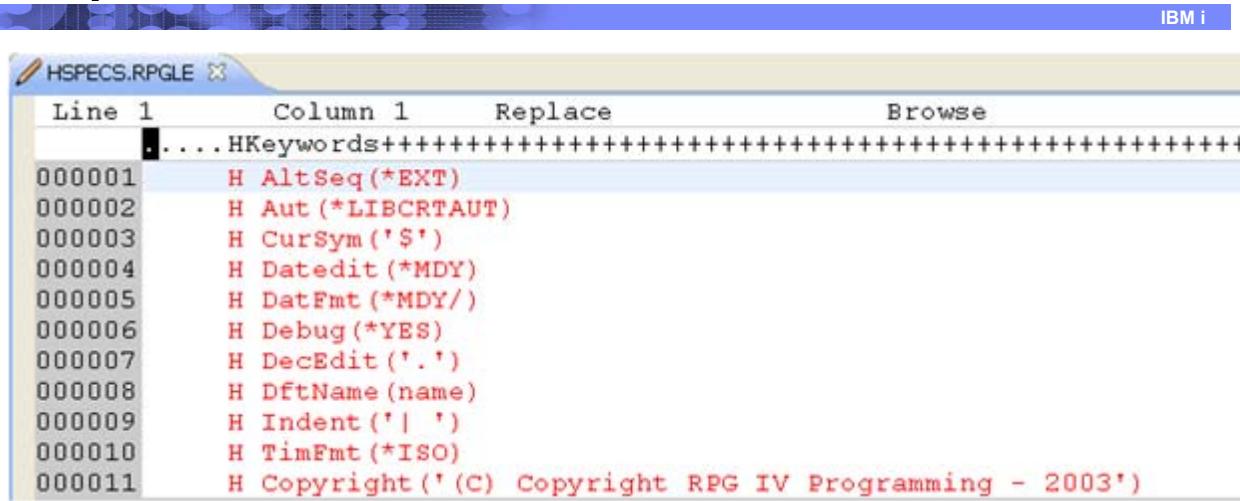
AS067.0

Notes:

Let us practice understanding the order in which RPG IV specifications must be sequenced. In this exercise, you are given a program that is not in the correct order. Rearrange the specifications and compile the source to create an executable program.

Perform the sequencing RPG IV specifications and compiling exercise.

H-specification



```

Line 1      Column 1      Replace      Browse
. ....HKeywords+++++
000001      H AltSeq(*EXT)
000002      H Aut (*LIBCRTAUT)
000003      H CurSym('$')
000004      H Datedit(*MDY)
000005      H DatFmt (*MDY/)
000006      H Debug(*YES)
000007      H DecEdit('.')
000008      H DftName(name)
000009      H Indent('| ')
000010      H TimFmt(*ISO)
000011      H Copyright(' (C) Copyright RPG IV Programming - 2003')

```

- Free format
- Keyword driven
- Style:
 - One keyword per line
 - Leave space after H

© Copyright IBM Corporation 2012

Figure 2-7. H-specification

AS067.0

Notes:

The control specification describes your program's requirements for compilation and run time through the use of keywords and parameter values for those keywords.

You do not have to code an H-specification. If you do not include control specifications in your source code, the system searches for them in two data areas. In this class, you should code your H-Specs in your programs. We discuss data areas in the class that follows this one:

- RPGLEHSPEC in *LIBL during compile time
- DFTLEHSPEC in QRPGLE

If the specifications are not found in either place, the keywords are assigned their system defaults.

If multiple keywords are required, they can be listed on consecutive lines, separated by a blank, until finished.

Style tip

Use the H spec to code your compilation and runtime options. By doing this, you include in the source code specific documentation about how your program is to be compiled and executed. Also, leave a space after the H and code only one keyword per line.

Reference:

ILE RPG Language Reference Manual, Chapter 12, Control Specifications

H-specification keywords

IBM i

RPG IV Keyword	Meaning
ALTSEQ	Alt. Collating Seq. Table
AUT	Specifies Authority
CURSYM	Currency Symbol
DATFMT	Default Date Format
DEBUG	Dump statements executed
DECEDIT	Decimal notation
DFTNAME	Prog. or module ID
INDENT	Specifies Indent Character
TIMFMT	Time format

© Copyright IBM Corporation 2012

Figure 2-8. H-specification keywords

AS067.0

Notes:

This is a list of some of the H-spec keywords. See Chapter 12 of the *ILE RPG Reference Manual* for a complete list and description of all keywords. You can also use the prompting facility of the SEU or LPEX Editors while you edit your program to review the options and to understand how to use a specific keyword.

- Date and Time formats should be specified unless the system defaults are adequate.
- Defaults for most other keywords are sufficient.

F-specifications

IBM i

```

Line 1      Column 1      Replace      Browse
000001     FEmpMaster IF   E          Disk
000002           // Declare output printer file
000003     FEmpReport O    E          Printer OflInd(Overflow)
000004

```

- Defines files to be processed by your program:
 - Name of file
 - Type (input, output, update, combined)
 - Format (program-described or externally-described)
 - Device (disk, printer, workstation)
- Keywords area to specify additional functions required

© Copyright IBM Corporation 2012

Figure 2-9. F-specifications

AS067.0

Notes:

For each file that your program processes, you must define the file in your program. Both of the files in this figure are externally described (notice the **E** in column 22).

In this example, we are defining two files:

- **EmpMaster**, a disk file that we read only; we do not change the file (notice the **I** in column 17).
- **EmpReport**, a printer file that holds the printed output of the employee records that the program reads and processes.

Each file requires its own specification line. Positions and keywords are used as needed:

- File name (positions 7 - 16)
- File type (position 17) I, O, U, C
- File designation (position 18) P, S, F
- File format (position 22) F, E

- Device (positions 36 - 42) Disk, Printer, WorkStn
- Keywords (positions 44 - 80)

File names can be 10 characters long. The first character of the name must be alphabetic (A through Z, \$, #, @) with the remaining characters alphabetic (A through Z, \$, #, @, _(underscore)) or numeric (0 through 9).

Blanks are not allowed.

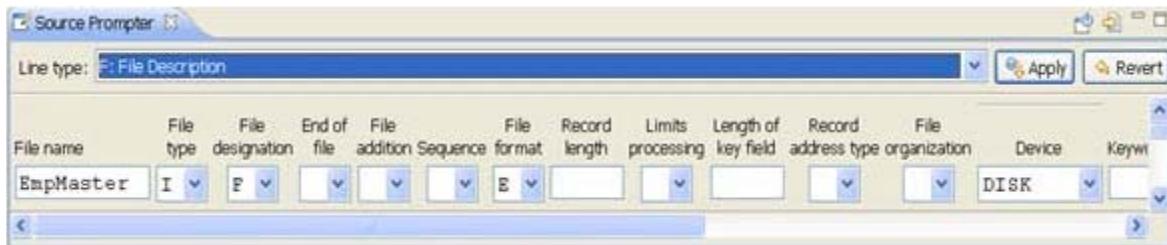
F-specification prompting

IBM i

SEU

Prompt type . . .	FX	Sequence number . . .	<u>0000.01</u>		
Filename	File Type	File Designation	End of File	File Addition	File Sequence
<u>EmpMaster</u>	I	F			
File Limits		Record			
Format Processing		Address Type	Device		
E	-	-	Disk		
Keywords			Comment		

LPEX



© Copyright IBM Corporation 2012

Figure 2-10. F-specification prompting

AS067.0

Notes:

The F-spec is a mixture of fixed- and free-format coding. To help you, the source editors include a *prompt* panel. This resembles an application data-entry display.

The SEU and LPEX prompting for the EMPMASTER F-spec is shown above.

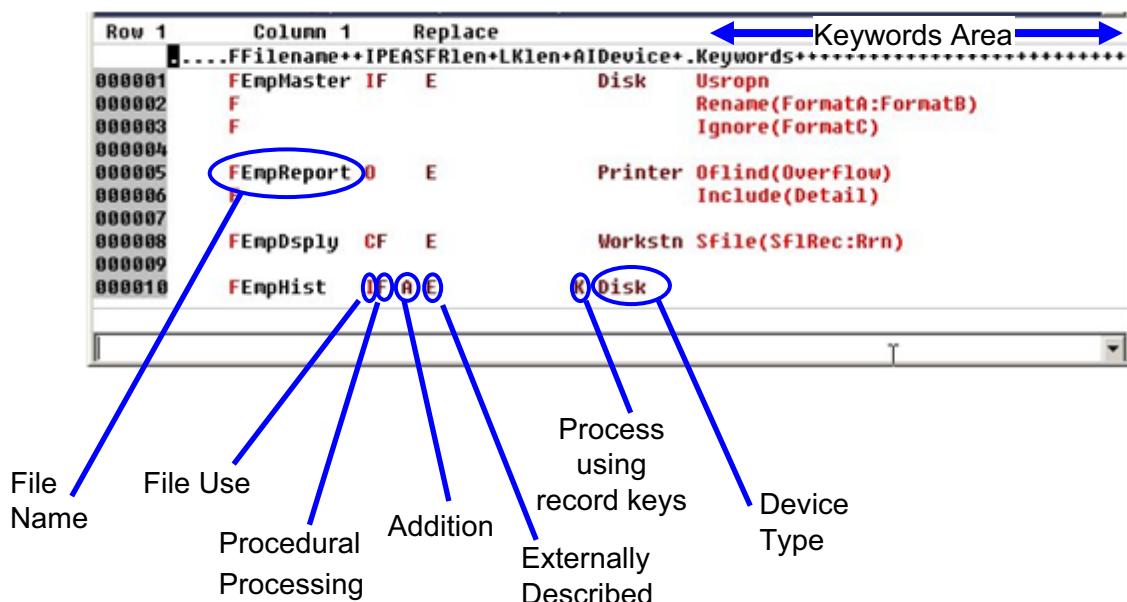
Existing source lines can be prompted by placing the cursor on the appropriate line and selecting the **F4** function key. Alternatively, use the P line-command in the left margin of the edit display and press **Enter**.

To add a new F-spec source line use the IPF line-command in the left-margin of the edit display and press **Enter**.

Most of the entry fields in the prompt window are fixed-format. The exception is the **Keywords** item, which allows free-format, keyword-oriented input.

F-specification components

IBM i



- Fixed column format except keywords area
- One F-specification for each file used by program

© Copyright IBM Corporation 2012

Figure 2-11. F-specification components

AS067.0

Notes:

This visual shows a number of examples of how you may use the F-specification to define files.

Columns on the F-spec are used to allow you to quickly and easily describe how a file is to be processed.

The keywords used in RPG IV F-specs provide keywords in areas such as defining how to open a file and what record format to use. These free-format keywords are optionally used to provide further detail, such as the name of the variable used to access a set of display records known as a subfile.

Device type and processing

IBM i

USE	DISK	PRINTER	WORKSTN
Input	Y (+ Addition)	-	(Y)
Output	Y Addition	Y	(Y)
Update	Y (+ Addition)	-	-
Combined	-	-	Y

(Y): Can define WORKSTN file this way,
but combined is recommended.

© Copyright IBM Corporation 2012

Figure 2-12. Device type and processing

AS067.0

Notes:

This visual shows the ways that a file can be processed depending upon how it is defined in the program.

Remember to make sure you define your file in a fashion consistent with how you intend to process the data in the file. Consider whether your disk, printer, or workstation files will be used for input, output, both input and output, or update, and define them appropriately.

Database processing options

IBM i

- A file can be processed:
 - Sequentially:
 - By RRN (RAT = blank)
 - By key value (RAT = 'K')
 - Randomly:
 - By RRN (RAT = blank)
 - By key value (RAT = 'K')
- RAT and opcode determine how file processed
- Sequential by RRN: Consecutive
- Random by RRN: Direct
- RRN: Relative Record Number
- RAT: Record Address Type (column 34)

© Copyright IBM Corporation 2012

Figure 2-13. Database processing options

AS067.0

Notes:

Processing sequentially by key is also known as indexed sequential processing.

H- and F-specifications: Employee print program

IBM i

```

000100
000200      H DatFmt (*ISO)  DecPrec(31)  ExprOpt(*ResDecPos)
000300      FEmpMaster IF   E           K Disk
000400          // Declare output printer file
000500      FEmpReport O   E           Printer OflInd(OverFlow)
000600          // Calculations follow; free format
000700      /FREE
000800          Read EmpMaster; // Read first Employee record
000900          // Print headings if at least one record
001000          If not %Eof(EmpMaster);
001100              Write Heading; // Write report headings;
001200          Endif;
001300          // Loop to process records until end of file reached
001400          Dow not %Eof(EmpMaster);
001500              Count = Count + 1; // Count total number of employees
001600              // Check for page overflow
001700              If OverFlow;
001800                  Write Heading; // If overflow, print headings
001900                  OverFlow = *off;
002000              Endif;
002100              Write Detail; // Print employee information
002200              Read EmpMaster; // Read 2nd and subsequent records
002300          EndDo;
002400          // Print total count of employees
002500          Write Total;
002600          // Stop program at end of file
002700          *inLR = *On;
002800      /END-FREE
002900
003000
003100

```

© Copyright IBM Corporation 2012

Figure 2-14. H- and F-specifications: Employee print program

AS067.0

Notes:

Let us look again at the employee print program.

Can you now see that how the files are defined to the program determines how that file may be processed?

Notice that both files are externally described and that the disk file is full procedural. If you do not specify full procedural, you get compilation errors.

How are we processing the file? We are using a READ opcode that says read the next record (sequential). But, notice also that the disk file has an RAT = K. That means we use the index and that the file is processed sequentially by key.

Desk exercise

Let us practice some coding at this point. You can code the following on a piece of paper, or, better, use the system and the SEU or LPEX Editor to create the following statements:

Code an H-spec that meets these specifications:

- Select your local country's currency symbol to be used in any editing.

- A run-time error is to be generated when numeric overflow is detected.
- The literal, '*AS06 RPG IV Programming - Fundamentals*' is to be defined as the copyright string for the program.

Code the F-specifications for these files.

Note: All files are externally described. All disk files are *full procedural*:

- Physical file ORDER (input) to be processed consecutively.
- Logical file CUSTOMER (input) to be processed by key.
- Printer file AUDITLIST; use indicator PageOver to identify page overflow.
- Physical file STOCK (update) to be processed by key, allowing records to be added.
- Display file INQUIRY.

D-specifications

IBM i

- D-specification defines program working variables:
 - Standalone fields
 - Named constants
 - Data structures and their subfields
 - Arrays
 - Named indicators

© Copyright IBM Corporation 2012

Figure 2-15. D-specifications

AS067.0

Notes:

The **(D)efinition** specification allows definition of data items specific to a program (that is, not from a file). Miscellaneous items, such as standalone fields, named constants, data structures, and arrays, are good examples.

D-specification prompting

IBM i

SEU

Prompt type . . .	D	Sequence number . . .	<u>0000.01</u>
Declaration			
Name	E	S/U	Type
<u>Character</u>	-	-	<u>S</u>
Internal	Decimal		
Data Type	Positions	Keywords	
A			
Comment			

LPEX



Figure 2-16. D-specification prompting

AS067.0

Notes:

Like the F-spec, the D-spec is also a mixture of fixed- and free-format coding.

The SEU and LPEX prompting for the CHARACTER D-spec is shown above.

Existing source lines can be prompted by placing the cursor on the appropriate line and selecting the **F4** function key. Alternatively, use the P line-command in the left margin of the edit display and press **Enter**.

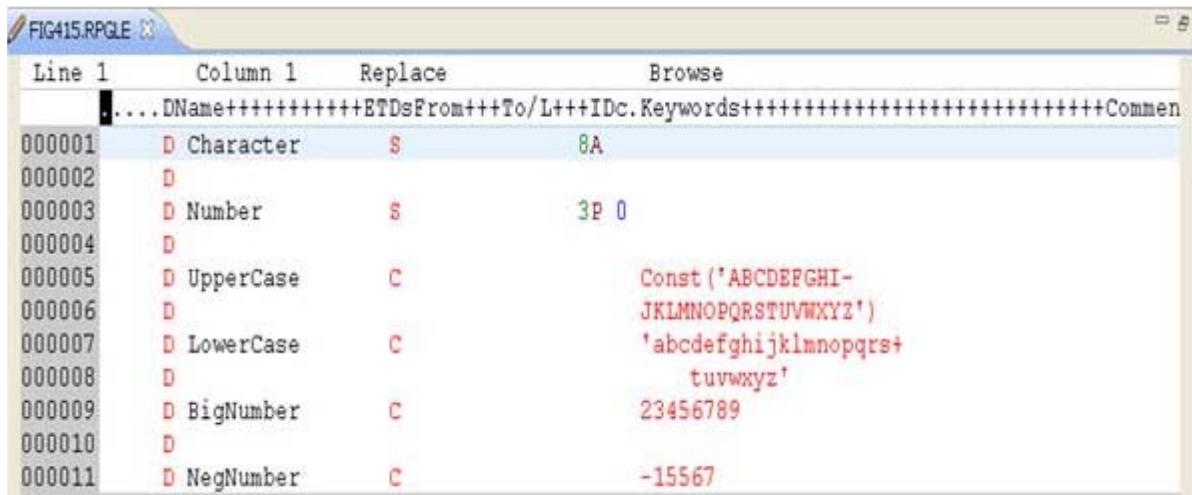
To add a new D-spec source line use the IPD line-command in the left-margin of the edit display and press **Enter**.

Most of the entry fields in the prompt window are fixed-format. The exception is the **Keywords** item, which allows free-format, keyword-oriented input.

D-specification to define data

IBM i

Standalone fields:



The screenshot shows a code editor window titled 'FIG415.RPGLE'. The code is a series of D-specifications. Line 1 contains a header. Lines 2 through 11 define individual fields with their types and values.

Line	Column	Type	Replace	Browse
1	1		...DName++++++ETDsFrom++To/L++IDc.Keywords+++++Comment	
000001	2	D Character	S	8A
000002	2	D		
000003	2	D Number	S	3P 0
000004	2	D		
000005	2	D UpperCase	C	Const ('ABCDEFGHI- JKLMNOPQRSTUVWXYZ')
000006	2	D		
000007	2	D LowerCase	C	'abcdefghijklmnopqrstuvwxyz'
000008	2	D		
000009	2	D BigNumber	C	23456789
000010	2	D		
000011	2	D NegNumber	C	-15567

© Copyright IBM Corporation 2012

Figure 2-17. D-specification to define data

AS067.0

Notes:

- **Name:** Name of data item, can indent to show structures (field, constant, data structure (DS), DS subfield)
- **E:** Indicates external DS; blank means program described
- **T:** Type of DS: (S - program status, U - data area); blank means not program status nor data area DS
- **Ds:** Entry of DS means this is a data structure; also means (sub)field type
- **C:** Constant
- **S:** defines standalone field or array
- **DS:** Data structure: Covered in a subsequent course
- **Blank:** data structure subfield
- **From:** From position (used with data structures)
- **To/L:** To position or length

- **I:** Internal data type
- **Dc:** Decimal positions
- **Keywords:** Functions using keywords (similar to DDS)
- **Keywords-cont:** If positions 7-43 are blank, continued from previous D-spec

For constants:

- Use C in the Ds column
- No length required
- Value specified in keywords area

Notice continuation of values:

- Character values continued with dash (-) continue with first column in keywords on the continuation line.
- Character values continued with plus sign (+) continue with first non-blank column in keywords area of continuation line.
- Numeric values require no continuation character and continue with the first numeric character on continuation line.

Reference:

ILE RPG Language Reference Manual, Chapter 14, Definition Specifications

Defining a data structure and array

IBM i

The screenshot shows the RPGLE editor window titled 'FIG416.RPGLE'. The code defines a data structure 'Personal' with subfields 'Name', 'FirstName', 'LastName', and 'Phone_No'. It also defines an array 'Array1' with one element 'AnyField'. The 'DS' (Data Structure) keyword is circled in blue. The 'Dim(10)' keyword is also circled in blue.

```

Line 1      Column 1      Replace      Browse
.....DName++++++ETD=From++To/L+++IDc.Keywords+++++
000100    D Personal     DS          Inz
000200    D Name          30A
000300    D FirstName    15A  Overlay (Name)
000400    D LastName     15A  Overlay (Name: *next)
000500    D Phone_No     10A
000600
000700
000800    D Array1       S           5S 0 Dim(10)
000900    D AnyField     S           7P 2
000901
000902

```

DS - data structure

Name through Phone_No - data structure subfields

Dim(10) - array is a set of 10 elements

5 0 - each element is zoned numeric five digits, zero decimals

© Copyright IBM Corporation 2012

Figure 2-18. Defining a data structure and array

AS067.0

Notes:

Two new concepts are introduced here. Both data structures and arrays are groupings of data:

- Data structure:** A data structure is an area in program storage that can be used to define the layout of subfields within that area. Data structures can be used to subdivide a single field or to group subfields into a single structure.
- Array:** A groups of related data. All elements same data type. Think of all the telephone numbers in a telephone directory. Each number is the same data type and length.

In the visual:

- **Personal** is a data structure containing the subfields **Name** and **Phone_No**. **Name** and **Phone_Number** are subfields (or redefinitions of the storage area occupied by **Personal**). Notice that subfields are blank in the **Ds** position of the D-spec. Also notice that the first non-blank value in the **Ds** position terminates the subfield definitions within the data structure.

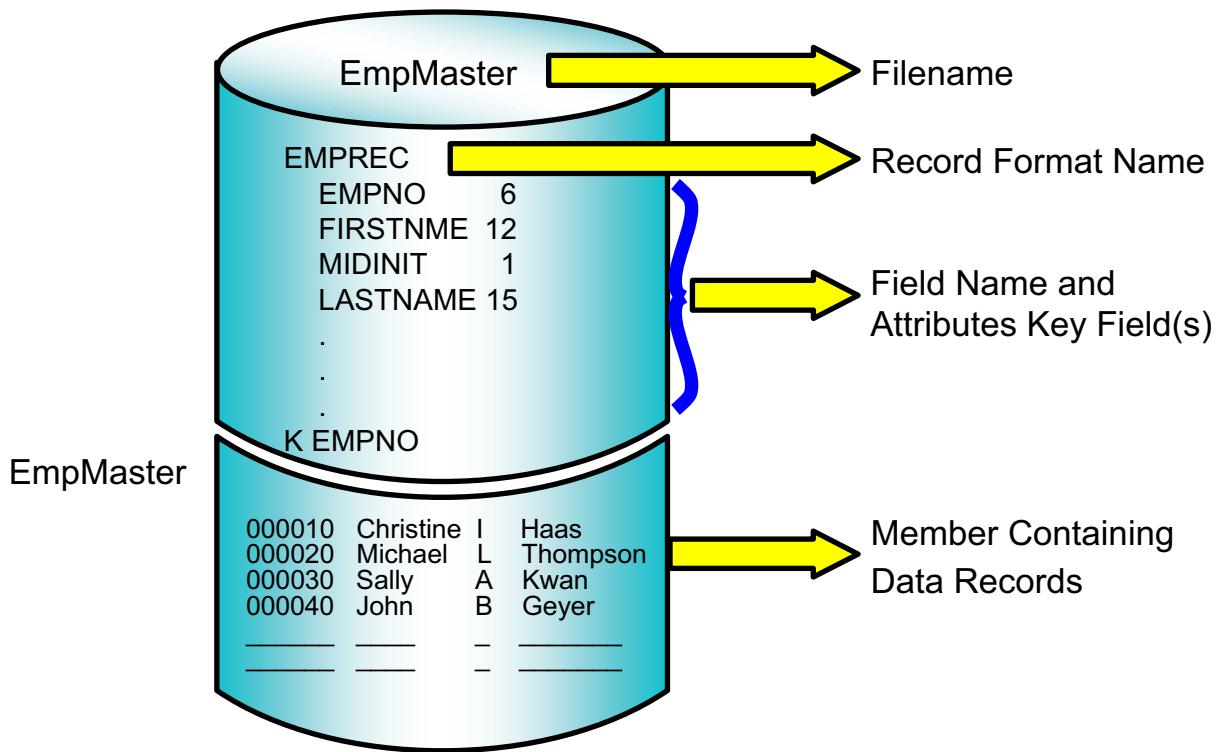
- **Array1** is an array of 10 elements, each a zoned numeric field of 5 digits. The **DIM** keyword is used to specify how many elements are in the array. You access an individual element of an array using an index. The index can be a constant or a variable. For example, **Array1(3)** operates on the third element of the array. **Array(Idx)** operates on the third element of the array if **Idx = 3**.

The example shows the definition of a data structure containing several subfields. Note the indentation of the subfields to show the structure.

Notice also that both data structures and arrays define groups of data. The subfields in the data structure do not have to be defined as the same data type and length. The elements of an array must be defined as the same data type and length.

Where are externally described descriptions?

IBM i



© Copyright IBM Corporation 2012

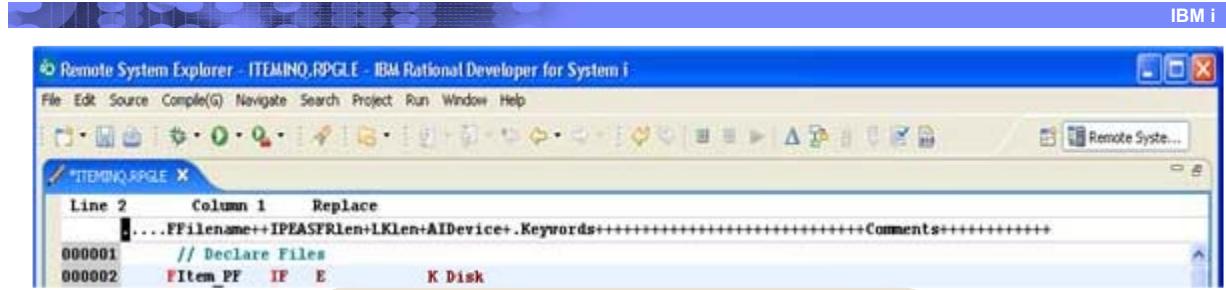
Figure 2-19. Where are externally described descriptions?

AS067.0

Notes:

When you describe a file using DDS (or SQL), the field names, data types, and field lengths are stored with the object. When you declare a file as externally described, the RPG IV compiler gets a copy of the record format and automatically generates input (or output) specifications based on the external definitions.

I-specification for input data



The screenshot shows the 'Remote System Explorer - ITEMNO.RPGLE' window in IBM Rational Developer for System i. The code editor displays an I-specification for an input file named 'ITEM_PF'. The code includes declarations for the file and its record formats, along with detailed descriptions of fields and their formats.

```

Line 2    Column 1    Replace
....FFilename++IPEASPRlen+LKlen+AIDevice+.Keywords+++++Comments+++++
000001 // Declare Files
000002 FItem_PF IF E K Disk

*
* RPG name          : ITEM_PF           External name      : AS06V1LIB/ITEM_PF
* File name. . . . . : ITEM_FMT        ITEM_FMT
* Record format(s) . . . . . : ITEM_FMT
*
24=IITEM_FMT
*
* RPG record format . . . . . : ITEM_FMT
* External format . . . . . : ITEM_FMT : AS06V1LIB/ITEM_PF
* Format text . . . . . : Item Master Record
*
25=I          P   1   3 0ITMNBR
26=I          A   4   28 ITMDESCR
27=I          P   29  32 0ITMQTYOH
28=I          P   33  36 0ITMQTYOO
29=I          P   37  39 2ITMCOST
30=I          P   40  42 2ITMPRICE
31=I          P   43  45 0VNNDNBR
32=I          A   46  52  ITMVNDCAT#

```

© Copyright IBM Corporation 2012

Figure 2-20. I-specification for input data

AS067.0

Notes:

Notice that no field level information is required for the externally described file.

The necessary I(nput) specifications are automatically generated for you by the compiler. Using externally described files not only saves you coding effort, remember that when you make a change to a file, all you have to do is recompile the programs that reference the file.

Reference:

- *ILE RPG Reference Manual, Chapter 14, "Definition Specifications"*

C-specifications

IBM i

- Arithmetic functions
- Processing logic:
 - Read, write, and update records
 - Indicator control
- Conditional execution
- Data manipulation
- Iterative processing

© Copyright IBM Corporation 2012

Figure 2-21. C-specifications

AS067.0

Notes:

After you have coded the control specification, defined the files used in the program, and defined any standalone fields, you are ready to code the C-specifications used to describe your program logic and all record I/O, some of which might be controlled by indicators. In the C-spec, define exactly what processing needs to be done to meet the requirements defined for a program.

Reference:

- *ILE RPG Reference Manual*, Chapter 16, *Calculation Specifications*

C-specification structure

IBM i

FIG420.RPGLE

Line 1	Column 1	Replace	Browse
.....DName++++++ETDsFrom+++To/L+++IDc. Keywords++++++			
000001 D Count S 5P 0 Inz(15)			
000002 D Positive S N			
000003 D Negative S N			
000004 D Zero S N			
000005 /Free			
000006 Select;			
000007 When Count > 0;			
000008 Positive = *On;			
000009 When Count < 0;			
000010 Negative = *On;			
000011 Other;			
000012 Zero = *On;			
000013 EndSl;			
000014 *InLR = *On;			
000015 /End-Free			
000016			

© Copyright IBM Corporation 2012

Figure 2-22. C-specification structure

AS067.0

Notes:

The basic structure of calculation specifications is illustrated in this visual:

1. This visual illustrates calculations. Notice they follow the D-specs. If there were I-specs coded in the program, the c-specs would follow them.
2. In this example, we perform some operations if a condition on the WHEN is satisfied; in this case a SELECT group is used to set indicators (positive, negative, or zero) based on the value of **Count**.

Typical calculations

IBM i

```

000100      H DatFmt(*ISO)  DecPrec(31)  ExprOpts(*ResDecPos)
000200      FEmpMaster IF   E           K Disk
000300      FEmpReport O    E           Printer Of1Ind(OverFlow)
000400      /FREE
000500      Read EmpMaster; // Read first Employee record
000600      Write Heading; // Write report headings
000700      // Loop - process records until end of file reached
000800      Dow Not %Eof(EmpMaster);
000900          Count=Count + 1; // Count total number of employees
001000          Name = %Trimr(Lastname) + ',' + %Subst(FirstName:1:1); // Build name
001100          // Check job classification
001200          If Job = 'MANAGER';
001300              Type = 'MANAGER';
001400          Else;
001500              Type = 'REGULAR';
001600          EndIf;
001700          // Check for page overflow
001800          If OverFlow;
001900              Write Heading; // If overflow, print headings
002000              OverFlow = *Off;
002100          EndIf;
002200          Write Detail; // Print employee information
002300          Read EmpMaster; // Read 2nd and subsequent records
002400      EndDo;
002500      Write Total; // Print final totals
002600      // Stop program at end of file
002700      *InLR = *On;
002800  /END-FREE
002801
002802

```

© Copyright IBM Corporation 2012

Figure 2-23. Typical calculations

AS067.0

Notes:

This is our employee listing program once again. One thing we want you to notice is that there is a loop that is executed until end of file is reached. Notice the **%Eof** Built-in Function that is used to test whether we have read the last record in the **EmpMaster** file.

Common opcodes

IBM i

BegSr	Begin subroutine
EndSr	End subroutine
ExSr	Execute a subroutine
CallP	Call with prototype
Chain	Random read with key or RRN
Delete	Delete record
DoW	Do While condition is true
DoU	Do Until condition is not true
Endxx	End group (Do, If, and so forth)
Eval	Evaluate an expression
ExFmt	Write then read a screen with format
For	Perform a group of calcs for iterations
If	Perform calcs that follow if expression is true
Else	Perform calcs that follow when if expression is false
Read	Read a record
Return	Return to calling program / module
Select	Begins a Select group
When	Part of select; perform calcs based on condition
Other	Part of select; catch all after Whens
SetLL	Set file cursor at first record
SetGT	Set file cursor at last record
Update	Update record
Write	Write record

© Copyright IBM Corporation 2012

Figure 2-24. Common opcodes

AS067.0

Notes:

This visual shows some of the available operation codes. We use many of them throughout the remainder of the course. Detailed descriptions of all RPG IV operation codes can be found in the *Reference Manual*.

Rules of free-format coding

IBM i

- /Free and /End-free groups:
 - Begin with /Free column 7-12
 - End with /End-Free column 7-16:
 - Column 6 must be blank
 - Rest of line *must be* blank
 - Every statement terminated by semicolon;
 - Semicolon (;) can follow the end of the statement in any position
 - RPG IV code starts in col 8 or higher
 - Every statement must begin with the opcode:
 - Optional for Eval and CallP
 - Only if no extenders
 - Operation extender immediately follows opcode --no space
 - Opcodes, such as CHAIN, are written opcode Factor1 Factor2 Result;
 - Comments (//)
 - Can start on separate line in column 8 or higher
 - Can be written after semicolon (;) of line of RPG IV code
 - Style Guideline:
 - Use indentation
 - Remember:
 - Opcodes requiring a resulting indicator are not supported
 - Must use free form BIF equivalent

© Copyright IBM Corporation 2012

Figure 2-25. Rules of free-format coding

AS067.0

Notes:

To take advantage of this new syntax, a few basic rules apply:

- Each free-format block must begin with /FREE and end with /END-FREE.
- Nothing is entered in column 6. It must be blank.
- /Free and /End-Free must start in column 7. The rest of line *must be* blank.
- When coding RPG IV statements and comments, columns 6 and 7 must be blank. Every statement in the block must be terminated by a semicolon.
- Every statement must begin with the opcode (except in the case of Eval where the opcode can be omitted).
- Comments (//) can start on separate line beginning in column 8 or later; or, they can be coded following the semicolon (;) that terminates a statement.
- Operation extender must immediately follow the opcode (no blanks).
- Procedure calls with no parameters must be coded with empty parentheses. This is true for both EVAL and CALLP (also true for fixed-format coding).

These rules differentiate free format calculations from fixed and extended format calculations.

This visual can be used as a reference when you are coding in free format.

Additional rules:

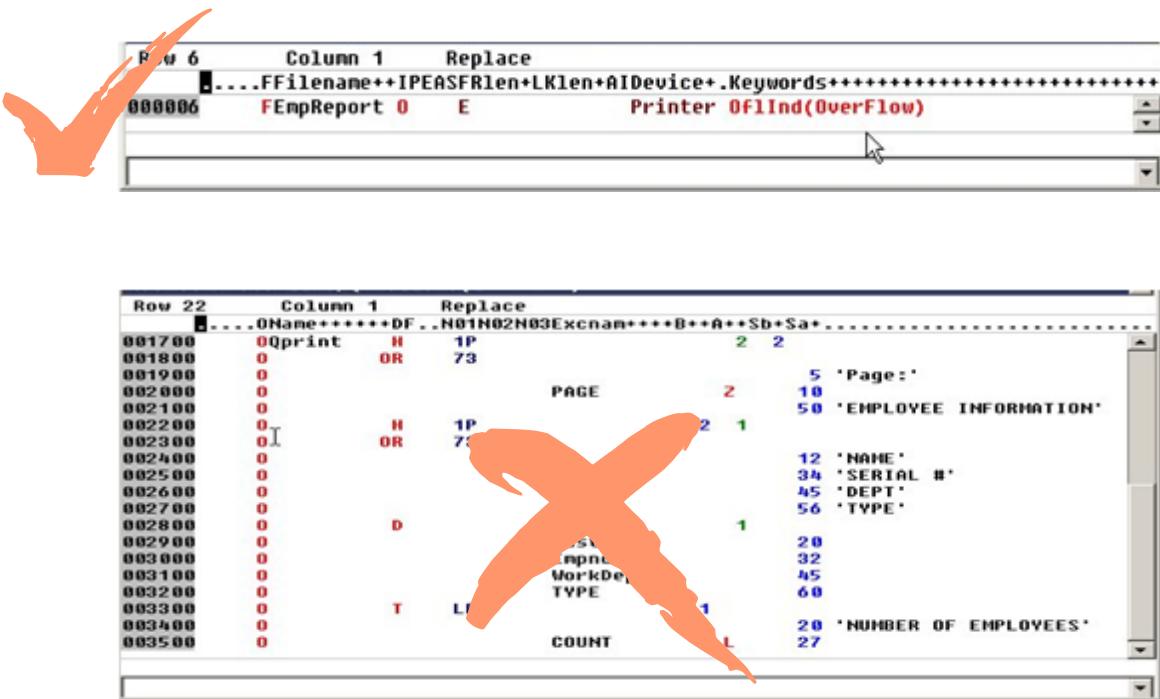
These are included for completeness only. They do not apply at this point in your RPG IV training, but, they apply as you learn additional techniques:

- Opcodes requiring a resulting indicator are not supported, (must use BIFs instead).
- Every statement must begin with the opcode (except in the case of `Eval` and `CallP` where the opcode can be omitted when no extenders are specified).

`CallP` and extenders are thoroughly discussed in a subsequent class.

Output specifications

IBM i



Row 6 Column 1 Replace
Ffilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
 000006 FEmpReport 0 E Printer OF1Ind(Overflow)

Row 22 Column 1 Replace
OName++++DF..N01N02N03Exnam+++B++A++Sb+Ss+.....
 001700 Qprint H 1P 2 2
 001800 OR 73
 001900
 002000
 002100
 002200 H 1P 2 1
 002300 OR 73
 002400
 002500
 002600
 002700
 002800 D 1
 002900
 003000
 003100
 003200
 003300 T L 1
 003400
 003500
 PAGE
 Ss
 Input
 WorkDef
 TYPE
 COUNT
 L

© Copyright IBM Corporation 2012

Figure 2-26. Output specifications

AS067.0

Notes:

You should never have to code these specifications by hand. All files should be externally described including report files. We show you this example for your information only. The output specifications in this example are generated by the compiler based upon the definitions of the report file.

Class exercise: Coding a simple listing

IBM i

- Given:
 - VENDOR_PF: Vendor Physical File
 - VENDORPRT: DDS for Printer File
- We: Code a program to list all records until end of file reached
 - For each record accumulate a record count
 - Print the record count once end of file is reached

© Copyright IBM Corporation 2012

Figure 2-27. Class exercise: Coding a simple listing

AS067.0

Notes:

Before you perform the machine exercise, your instructor leads a class exercise where you write a listing program. Be prepared to be actively involved and ask a lot of questions.

Your level of involvement benefits you greatly when you write your first program from scratch in the exercise that follows.

Machine exercise: Coding a report program and adding overflow

IBM i



© Copyright IBM Corporation 2012

Figure 2-28. Machine exercise: Coding a report program and adding overflow

AS067.0

Notes:

Perform the exercises:

- Coding a report program
- Adding overflow

Checkpoint

IBM i

1. True or False: An H-spec MUST be coded into every RPG IV source member.

2. Which of the following are functions of the F-spec in RPG IV?
 - a. To reflect the status of the last input operation
 - b. To designate the file as Input or Output
 - c. To specify a file described externally or in the program

3. True or False: The D-spec is NOT needed to define fields from an externally described file.

© Copyright IBM Corporation 2012

Figure 2-29. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe the purpose of each RPG IV specification type
- Code specifications in the order required for an RPG IV program to compile successfully
- Describe how to reference externally described database and printer files in RPG IV
- Write a simple listing program

© Copyright IBM Corporation 2012

Figure 2-30. Unit summary

AS067.0

Notes:

Unit 3. Data representation and definition

What this unit is about

This unit describes how data can be defined to the RPG IV compiler. It reviews the various data types and describe how they can be defined to the database and then brought into your RPG programs. This unit covers the most commonly used data types: character, indicator, packed and zoned decimal, as well as integer.

What you should be able to do

After completing this unit, you should be able to:

- List the most common data types supported in RPG IV
- Use D-specifications to define these data types

How you will check your progress

- Checkpoint questions
- Classroom exercise:

We will assign desk exercises to practice using D-specs to define all work fields in RPG IV programs.

- Machine exercise:

Students add file and data definitions as required to an existing program.

Unit objectives

IBM i

After completing this unit, you should be able to:

- List the most common data types supported in RPG IV
- Use D-specifications to define these data types

© Copyright IBM Corporation 2011

Figure 3-1. Unit objectives

AS067.0

Notes:

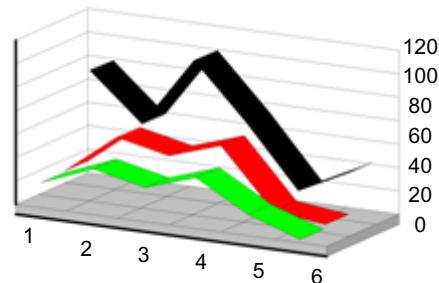
RPG IV data types

IBM i

AaBbCcDd



1.45



1001▲1111▲1010▲1111▲1101▲0000
1010▲1111▲0010▲1111▲0001▲1100

© Copyright IBM Corporation 2011

Figure 3-2. RPG IV data types

AS067.0

Notes:

There are many different data types that you can manipulate in your RPG IV programs. To use them effectively, you must understand how and where they can be used. In the rest of this topic, we discuss each data type.

Often, the data type and its use are familiar to those with an extensive programming background. We encourage you to ask lots of questions to make sure you understand how the data type is used on the i versus any other system or language with which you are familiar.

Character data

IBM i

C 9	C 2	D 4
IBM = 1100▲1001	1100▲0010	1101▲0100

F 1
*IN50 = *ON = 1111▲0001

- 1 byte = 1 character
- Data type A

© Copyright IBM Corporation 2011

Figure 3-3. Character data

AS067.0

Notes:

Character, or alphanumeric, data is one of the most common data types. It can be anywhere from one byte to 65,535 bytes in length. Character fields are initialized to blanks by the compiler.

You might assume that character data only included data items, such as name fields or address fields.

In addition, the numeric (and certain special) indicators that RPG IV uses are a 1-character field. They can be used to condition execution of certain pieces of your code. They can be used to inform you that a certain record of a specific file has been read to condition logic in your calculations and to condition output. Indicators have two possible values:

On = '1'

Off = '0'

Indicators are initialized with the value '**0**'. There are 99 numeric indicators in RPG IV. You see them in the following forms:

- 85 means indicator 85 is on (***On or '1'**).

- N40 means indicator 40 is off (***Off or '0'**).
- *IN15 means indicator 15 is on (***On or '1'**).

Indicators might also be represented by an RPG IV field name, as you have seen in previous examples. We show you the indicator data type in this unit. As we have been showing you in all the examples so far, you can write all your programs using named indicators rather than using the numeric indicators which are the character data type.



Note

There are other indicators, such as ***INLR**, ***INOF**, ***INOV**, and so forth. There are over 100 indicators available in the RPG IV language in total.

Numeric data

IBM i

ZONED		
F 1	F 2	F 3
123 = 1111 ▲0001	1111 ▲0010	1111 ▲0011

ZONED		
F 4	F 5	F 6
456 = 1111 ▲0100	1111 ▲0101	1111 ▲0110

PACKED				
4	5	6	7	8 F
456.78 = 0100 ▲0101	0110 ▲0111	1000 ▲1111		

Decimal point is IMPLIED

© Copyright IBM Corporation 2011

Figure 3-4. Numeric data

AS067.0

Notes:

There are several types of numeric data that we discuss. Numeric data in RPG IV is described as a number of digits including zero or more digits to the right of the *implied* decimal point. Because the decimal point is implied, and is not stored with the numeric data, you will not see it when you display the data. However, when you edit a numeric field with a decimal point, it is included.

A numeric variable is initialized to zero unless you initialize it in your program (on the D-spec) to some specific value.

For example, the notation (5 1) means that the field contains five digits in total, four digits to the left of the decimal and one digit to the right of the decimal. Remember, the decimal is not stored as part of the number. It is implied.

Zoned decimal

IBM i

ZONE DIGIT			SIGN			
F	1	F	2	F	3	SIGN
+123 =	1111 ▲0001	1111 ▲0010	1111 ▲0011			

SIGN						
F	1	F	2	D	3	SIGN
-123 =	1111 ▲0001	1111 ▲0010	1101 ▲0011			

- 1 byte = 1 digit
- Data type S

© Copyright IBM Corporation 2011

Figure 3-5. Zoned decimal

AS067.0

Notes:

Zoned decimal format requires one byte of storage for each numeric digit. The **zone** portion of the byte is set to **1111** and the digit portion is the binary value of the digit. The maximum length is 63 digits (including decimals) beginning with V5R3. (The maximum was 30 digits.)

Zoned formats should be specified for numeric fields when there is a requirement for converting the numeric data to character data, for this can be done quite easily. Zoned numeric is also compatible with numeric data defined in display and printer files.

X'F' is the positive sign, while X'D' is the negative sign. Other signs are tolerated (A/B/C/E) by the system and are changed to F or D when they are operated on for the first time.

Packed decimal

IBM i

	2	5	6	F
256 =	0010	▲ 0101	0110	▲ 1111

	0	2	5	D
-25 =	0000	▲ 0010	0101	▲ 1101

- 1 byte = 2 digits
- Data type P

© Copyright IBM Corporation 2011

Figure 3-6. Packed decimal

AS067.0

Notes:

Packed decimal format can hold two numeric digits per byte. The only exception to this is that the second half of the low order byte contains the sign for the number. The convention on the i is that hexadecimal 'F' is positive and hexadecimal 'D' is negative. You may have a total of 63 digits (V5R3 or higher). (The maximum was 30 digits.) Because packed decimal data is stored two digits to a byte except the low order byte (which includes the sign), numeric data requires:

(# of digits + 1) / 2

bytes of storage (round up any fraction). For example:

6 digits requires (6 + 1) / 2 = 4 bytes

9 digits requires (9 + 1) / 2 = 5 bytes

Wherever possible, you should define packed fields with an odd number of digits so as not to waste the first half of the high-order byte. Doing this also reduces the overhead required when operating on the packed field.

Packed decimal is the default form of numeric data on the i. Numeric calculations are designed to perform best when they operate on packed decimal data.

Since most arithmetic in RPG is done in packed format, it makes sense to define numeric data in this format in order to avoid unnecessary internal data conversion.

Finally, other values for signs are supported to ease exchange of data between systems. You find them in the *ILE RPG Language Reference*.

Integer/Unsigned

IBM i

- Integer/Unsigned:
 - Data types I/U
 - 1, 2, 4, or 8 bytes in length
 - 3, 5, 10, or 20 digits respectively

© Copyright IBM Corporation 2011

Figure 3-7. Integer/Unsigned

AS067.0

Notes:

You define an integer field by specifying I in the data-type entry of the appropriate specification.

The length of an integer field is defined in terms of number of digits; it can be 3, 5, 10, or 20 digits long. A 3-digit field takes up 1 byte of storage; a 5-digit field takes up 2 bytes of storage; a 10-digit field takes up 4 bytes; a 20-digit field takes up 8 bytes. The range of values allowed for an integer field depends on its length.

Integer (and character) data types are the only formats common across most platforms and languages and therefore popular if data interchange is a requirement. Application Program Interface (API) usage also demands integer data processing.

The length determines the range of values as follows:

Field length	Range of Allowed Values
3-digit signed	-128 to 127
5-digit signed	-32768 to 32767
10-digit signed	-2147483648 to 2147483647
20-digit signed	-9223372036854775808 to 9223372036854775807
3-digit unsigned	0 to 255

Field length	Range of Allowed Values
5-digit unsigned	0 to 65535
10-digit unsigned	0 to 4294967295
20-digit unsigned	0 to 18446744073709551615

The **unsigned** format is like integer format except that no negative values are allowed.

Named indicators

IBM i

```

000012 FVendor_PF IF E K DISK
000013 FVendorPrt O E Printer OFFLINE(PrtOverflow)
000014
000015 D NoRec S N
000016
000017 /FREE
000018 Write Heading;
000019
000020 /* Read first record
000021 Read Vendor_Fmt;
000022
000023 NoRec = %EOF(Vendor_PF); // Problem! We have an empty vendor file
000024
000025 Dow not %EOF (Vendor_PF); //While NOT EOF
000026
000027 // Check for overflow
000028
000029 IF PrtOverflow;
000030   Write Heading;
000031 EndIF;
000032
000033 Write Detail;

```

AS06U2LIB/QRPGLESRC(FIG58) resequenced and saved

- Use named indicators rather than numeric *Inxx (except *InLR).
- The data type is N.

© Copyright IBM Corporation 2011

Figure 3-8. Named indicators

AS067.0

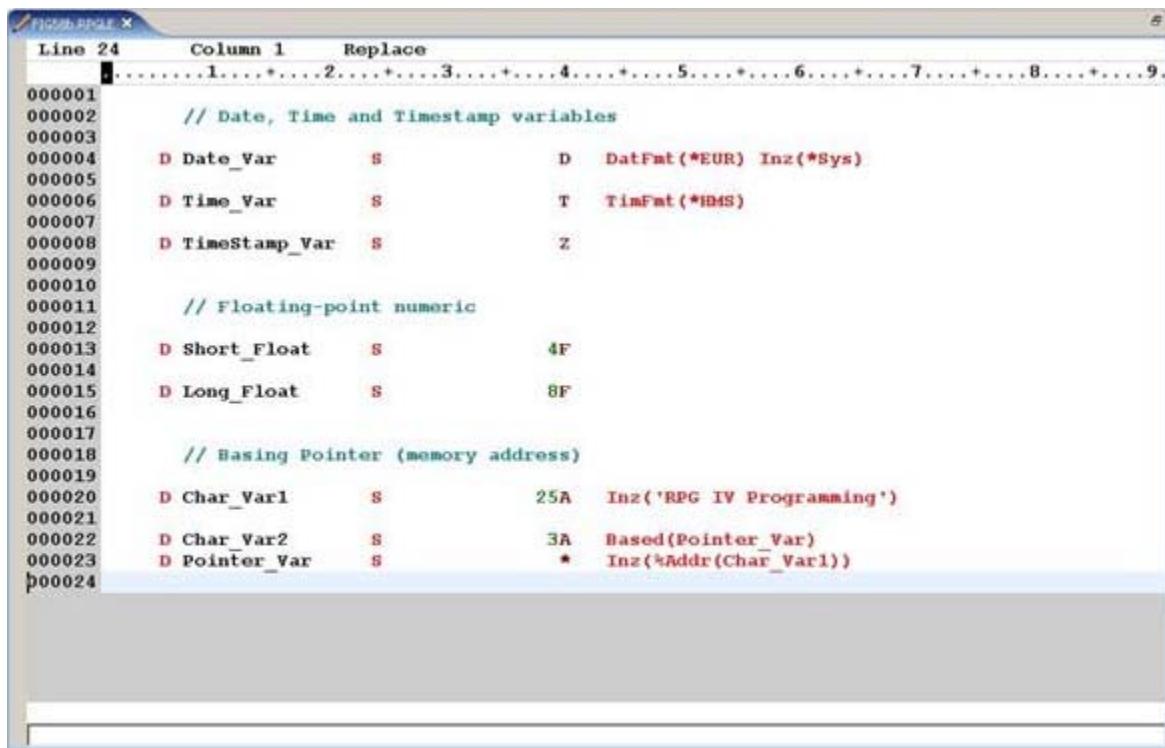
Notes:

The indicator data type allows you to use a meaningful field name instead of a numeric indicator. Notice that the data type is N.

In this example, notice the use of a named indicator for overflow, PrtOverflow. It is implicitly defined. Also, notice the named indicator, NoRec. This indicator is explicitly defined and is set when the Vendor_PF file is empty.

Other data types

IBM i



```

Line 24   Column 1   Replace
.....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9.
000001      // Date, Time and Timestamp variables
000002
000003
000004 D Date_Var      S          D DatFmt(*EUR) Inz(*Sys)
000005
000006 D Time_Var      S          T TimFmt(*IBMS)
000007
000008 DTimeStamp_Var  S          Z
000009
000010
000011      // Floating-point numeric
000012
000013 D Short_Float   S          4F
000014
000015 D Long_Float    S          8F
000016
000017
000018      // Basing Pointer (memory address)
000019
000020 D Char_Var1     S          25A  Inz('RPG IV Programming')
000021
000022 D Char_Var2     S          3A  Based(Pointer_Var)
000023 D Pointer_Var   S          *  Inz(%Addr(Char_Var1))
000024

```

© Copyright IBM Corporation 2011

Figure 3-9. Other data types

AS067.0

Notes:

For completeness we have listed other types of variable you will encounter in RPG IV programs. These data types are covered on subsequent courses.

Date and time: With this data type, you can find the number of days between two dates or add 10-days to a date giving a new date, without having to be concerned about rolling into new months or years. This means that date calculations take into consideration the number of days in a month when calculating a new date, for example.

Floating point: Another numeric data type. As with packed, zoned and integer data types, floating point can be used for routine arithmetic operations. It is especially useful for engineering and scientific calculations where a variable might contain a large range of values – from very small to very large.

Basing pointer: A pointer contains an address in memory, not a value. Any variable based on a pointer will therefore map over this memory location. Pointer manipulation is an advanced programming technique, not for the novice.

Figurative constants

IBM i

• *BLANK/*BLANKS	Initialize character or graphic fields
• *ZERO/*ZEROS	Initialize character or numeric fields
• *ALL'x..'	Initialize character field to string
• *ALLX'F1..'	Initialize character field to hex string
• *HIVAL	Highest value for data type
• *LOVAL	Lowest value for data type
• *ON/*OFF	All '1's or '0's (character data only; usually with indicators)
• *ALLG'ok1k2i'	Initialize graphic field to string
• *Null	Initialize to nulls
• *START/*END	Used to position at the beginning or end of database files

© Copyright IBM Corporation 2011

Figure 3-10. Figurative constants

AS067.0

Notes:

These are special variables that are supplied to you by the language. You use them just like you would a variable in your program. They assume the characteristics of the field you're moving them to.

If you move the string ***ALL'xyz'** to a five byte field, the field contains **'xyzxy'**.

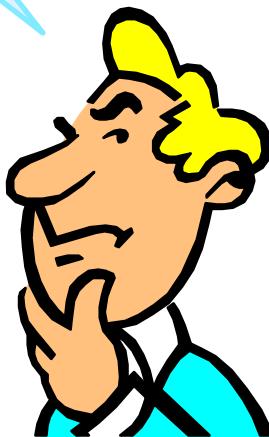
For figurative constraints:

- No length or decimal positions required.
- Adapt to the definition of field operated upon

Data definition

IBM i

Where can data
be defined in an
RPG IV program?



© Copyright IBM Corporation 2011

Figure 3-11. Data definition

AS067.0

Notes:

RPG allocates only one storage area to each field that you define. You can define the field more than once if you like, but the definitions must be consistent or you get an error at compilation time. Even if the field is defined for a second time in another file, it is allocated only *one* storage area.

Externally described files

IBM i

- Uses definitions in DB2 UDB for i
- Assures consistent file and field names:
 - Easier to maintain applications
 - Can override field names from database
- Has other uses:
 - Parameter list
 - Data structure

© Copyright IBM Corporation 2011

Figure 3-12. Externally described files

AS067.0

Notes:

Earlier, we introduced the subject of externally described file descriptions. We contrasted their use to that of program described file descriptions. Using externally described data is the preferred way of defining data files. For the reasons stated in the visual, they are much easier to use and maintain.

Next, you are shown an example of how the names of fields in an externally described file could be made unique within a program.

Defining externally described data

```

VENDORLIST.RPGLE
Line 1      Column 7      Replace      Browse
.....1....+....2....+....3....+....4....+....5....+....6....+....7....+....E
000001      //*****          Vendorlist
000002      // Vendor Report
000003      //*****
000004      // Print listing of ALL vendors from Vendor_PF file.
000005      //
000006      //
000007      // INDICATORS:
000008      //   PrtOverflow - Printer overflow
000009      //   LR - Close files, end program
000010      //
000011      //*****
000012      #Vendor_PF IF   E           K DISK

112=IVENDOR_FMT
*-----
* RPG record format . . . . : VENDOR_FMT
* External format . . . . : VENDOR_FMT : AS06v6LIB/VENDOR_PF
* Format text . . . . . : Vendor Master File Record
*-----
113=I          P    1     3  OVNDNBR
114=I          A    4     28 VNDNAME
115=I          A    29    53 VNDSTREET
116=I          A    54    76 VNDCITY
117=I          A    77    78 VNDSTATE
118=I          P    79    81 OVNDZIPCODE
119=I          P    82    83 OVNDAREACD

```

© Copyright IBM Corporation 2011

Figure 3-13. Defining externally described data

AS067.0

Notes:

You can retrieve the file and field definitions for a record format as simply as this. Remember that each field is given a storage address only once. If you have the same field name in multiple record formats or in different files, they will share one storage address.

Externally described files have an E on the F-spec as shown to indicate an external file definition. These definitions are copied into the program at compile time.

If the database format of **Vendor_PF** changes, all you have to do is recompile this program to implement the changes. If the program and database definitions do not match, the system will inform you of the problem with a Level Check message. The nice feature is that you do not have to change the program's record definition (because the file is externally described). All you do is recompile.

Overriding externally defined data names



The screenshot shows an IBM i RPGLE editor window. The code being edited is:

```

Line 4    Column 1    Insert  10 changes
....FFilename++IPEASFRlen+1Klen+AIDevice+.Keywords+++++Comments+++++
000100  FPayReg  UF  E      DISK  PREFIX(Pay:3)
000200  F          INCLUDE(PayRec)
000201  FPayReg1 UF  E      DISK  PREFIX('':3)
000202  FPayReg2 UF  E      DISK  PREFIX('FIDS.')

```

The code defines four fields: FPayReg, FPayReg1, FPayReg2, and FPayReg3. The first field uses the `PREFIX` keyword with a length of 3 to override the externally defined PayReg file. The second field uses the `INCLUDE` keyword to include the PayRec file. The third and fourth fields also use the `PREFIX` keyword with a length of 3.

- Assume externally described PayReg file has fields named:
 - `EmpNumber`
 - `EmpLastNme`
 - `EmpInitial`
 - `EmpFrstNme`
 - `EmpMonthRt`
 - `EmpAddress`
 - `:`
- Using `PREFIX(Pay:3)` will result in fields named:
 - `PayNumber`
 - `PayLastNme`
 - `PayInitial`
 - `PayFrstNme`
 - `PayMonthRt`
 - `PayAddress`
 - `:`

© Copyright IBM Corporation 2011

Figure 3-14. Overriding externally defined data names

AS067.0

Notes:

Notice that you need only to reference the file name and the record format that you want to define. Also, using the keyword `INCLUDE` (or in contrast, `IGNORE`) saves you program storage space at execution time as you are not defining any unused data in your program.

In this visual, you can also see the `PREFIX` keyword on the F-spec. Using this keyword allows you to prefix a variable name with the characters of your choice.

`PREFIX` allows global renaming of all fields in the file.

When you use `PREFIX`, you can concatenate characters to all field names and / or replace a specified number of characters in all the original field names.

The `PREFIX` keyword of the F-spec helps you to maintain uniqueness of the field names that you use in your program. It *does not* change the names of the fields in your database files. These are maintained the same as they were. However, the fields in your program override those names while the program is executing and, therefore, update any values that are changed in your program.

PREFIX can be used to replace a portion of the database field name or to concatenate a prefix to them. Usually, we think that programmers will use the replace capability of the **PREFIX** keyword to maintain the current length of their existing field names.

To remove characters from the beginning of every name, specify an empty string as the first parameter: **PREFIX('':number_to_remove)**.

For example, **PREFIX("":3)** would remove the first three characters of every field name.

Characters used in your prefix can be specified as a name, for example **PREFIX(F1_)**, or as a character literal, for example **PREFIX('F1_')**. A character literal must be used if the prefix contains a period; for example **PREFIX('F1DS.')** or **PREFIX('F1DS.A')**.

You can still override field names specifically on your input specs (see the *RPG IV Reference* manual for details). We suggest that you use the **PREFIX** keyword rather than using the override capability on input specs. Thus, maintenance is easier.

D-specification layout

IBM i

```

FIG514.RPGL X
Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom++To/L++IDc.Keywords+++++
000001      D VndSales1    S           16
000002      D TimeNow     S           6S 0
000003      D Counter     S           9P 0

```

- Field name can start in column 7
- Style guide recommends:
 - Define all working fields on D-specifications
 - Leave a space
 - Use meaningful names
 - Use upper and lower case
- Easy maintenance

© Copyright IBM Corporation 2011

Figure 3-15. D-specification layout

AS067.0

Notes:

- **TimeNow** is a 6-digit Zoned Decimal numeric.
- **Counter** is a 9-digit Packed Decimal numeric.
- **VndSales1** is a 16-character variable.

Definition specifications should be used to define:

- Standalone fields
- Named constants
- Data structures and their subfields (covered in AS07/AS070)
- Arrays (covered in AS07/AS070)

Use of the definition spec is the way to define *work* fields because of the function and the flexibility D-specs provide. Maintenance is made easy because all fields are grouped in one place in your program and, if you follow the *RPG IV Style Guidelines*, your names will be easily understood and identified.

For your information, data that is now defined on the D-spec, was defined on the C-spec in the past. The following code illustrates the technique:

```

FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords.....
FVendor_PF IF   E           Disk

6=IVENDOR_FMT
*-----
* RPG record format . . . . : VENDOR_FMT
* External format . . . . . : VENDOR_FMT : OL86V6LIB/VENDOR_PF
* Format text . . . . . . : Vendor Master File Record
*-----

7=I          P    1    3 0VNDNBR
8=I          A    4    28 VNDNAME
9=I          A    29   53 VNDSTREET
10=I         A    54   76 VNDCITY
11=I         A    77   78 VNDSTATE
12=I         P    79   81 0VNDZIPCODE
13=I         P    82   83 0VNDAREACD
14=I         P    84   87 0VNDTELNO
: : :
24=I          P  140  144 2VNDBALANCE
CL0N01Factor1++++++Opcode&ExtFactor2++++++Result++++Len++D+..en++D+..
C           Add      VndBalance     TotBalance      11 2

```

Notice that this type of definition is supported only in Fixed Format calculations. You cannot use this form when coding free format. One could say that a benefit of free format is that it forces all work data to be defined on the D-specs.

Defining standalone fields

IBM i

- Use S in Ds column
- Length notation
- Can be an array

Line 1	Column 1	Replace	Browse
.....DName++++++ETDsFrom++To/L+++IDc.Keywords++++++			
000001	D AlphaImplicit	S	8
000002	D AlphaExplicit	S	10A
000003	D InitField	S	3 0 INZ(999)
000004	D AnInteger	S	10I 0 INZ(12345)
000005	D PackedNumber	S	9P 2 INZ(345687)
000006	D ZonedNumber	S	6S 0
000007	D BinaryNumber	S	4B 1 INZ(123.4)
000008	D UnsignedNum	S	5U 0
000009	D FloatingNum	S	4F INZ(1.00E05)
000010			

© Copyright IBM Corporation 2011

Figure 3-16. Defining standalone fields

AS067.0

Notes:

When standalone fields are defined on the D-spec, maintenance is easier if all the internal data definitions are grouped together in the D-specs.

Note the use of the INZ keyword to assign an initial value for a standalone field. You can assign a specific value or if one is not assigned with the INZ keyword, the compiler will use a default value. For example, an alphanumeric field defaults to blanks and a numeric field to zeroes. If you specify a value using INZ, be aware of the data type. For example, an alphanumeric field value must be surrounded by *single quotes*.

Another important point to note is that when you define a numeric field on the D-spec, you must specify the *actual length of the field in digits* rather than being concerned about the amount of storage required to hold the numeric variable. For example:

```
Define 10 digits packed      (10P) requires 6 bytes of storage
10 digits integer          (10I) requires 4 bytes of storage
But....10 characters alpha (10A) requires 10 bytes of storage
```

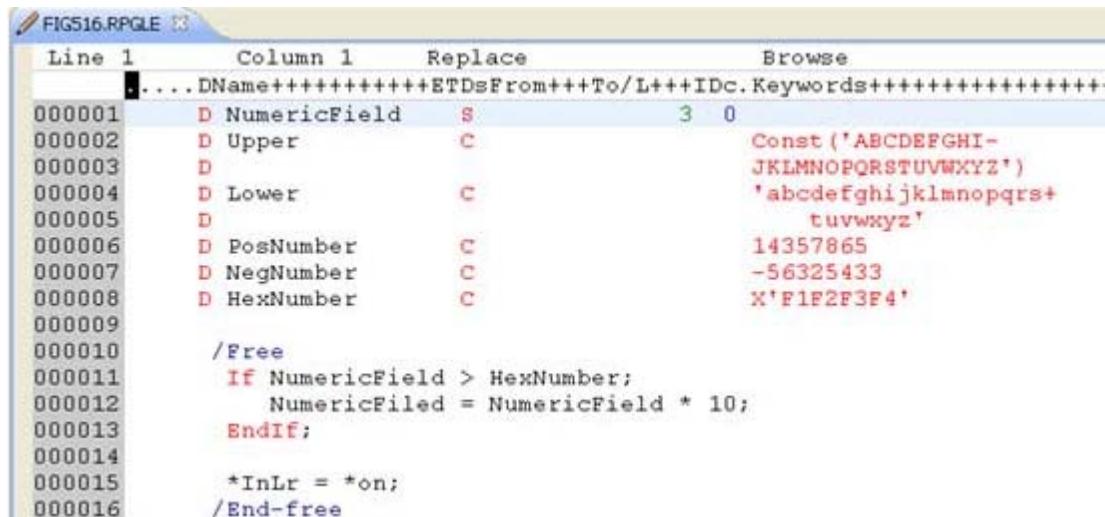
Do not forget to define packed fields with an odd length where possible.

If you do not specify a data type, the field will default to Packed (P) if you specified decimal positions. Otherwise, the default is Alphanumeric (A).

Defining named constants

IBM i

- Use C in Ds column.
- No length is required.
- Value is specified in keywords area.



The screenshot shows an IBM i editor window titled 'FIG516.RPGLE'. The code defines several named constants:

```

Line 1      Column 1      Replace      Browse
. .... DName++++++ETDsFrom+++To/L+++IDc. Keywords+++++++
000001    D NumericField   S           3 0
000002    D Upper          C
000003    D
000004    D Lower          C
000005    D
000006    D PosNumber       C
000007    D NegNumber       C
000008    D HexNumber       C
000009
000010    /Free
000011    If NumericField > HexNumber;
000012        NumericFiled = NumericFiled * 10;
000013    EndIf;
000014
000015    *InLr = *on;
000016    /End-free

```

© Copyright IBM Corporation 2011

Figure 3-17. Defining named constants

AS067.0

Notes:

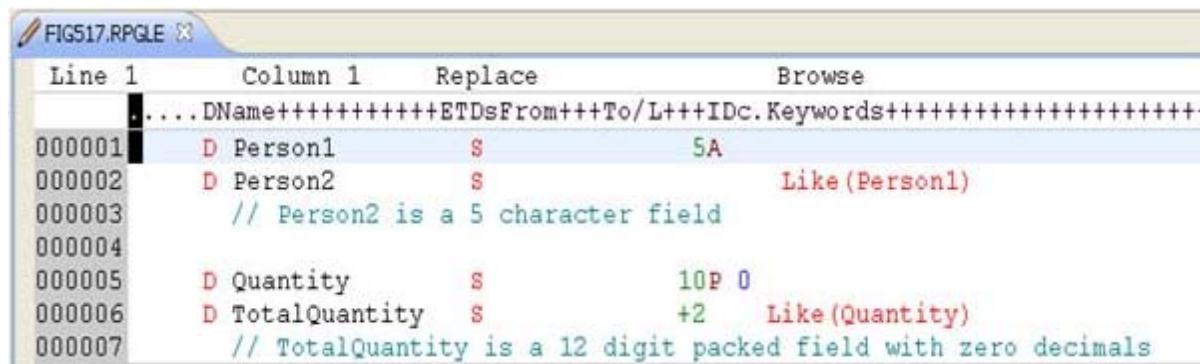
Notice that when the value of the constant is too long for a single line that a continuation of a value is coded:

- Character values continued with dash (-) continue with first column in keywords on the continuation line.
- Character values continued with plus sign (+) continue with first non-blank column in keywords on continuation line.
- Numeric values require no continuation character and continue with the first numeric character on continuation line.
- Named constants are not variables and, therefore, you cannot change the value of a named constant.

LIKE

IBM i

- **LIKE:** Length, decimal positions and data type copied
- Increase (+) or decrease (-) length
- Use with fields, subfields



The screenshot shows an IBM i RPGLE editor window titled 'FIG517.RPGLE'. The code is as follows:

```

Line 1      Column 1      Replace          Browse
.....DName++++++ETDsFrom++To/L++IDc.Keywords+++++
000001    D Person1      S               5A
000002    D Person2      S               Like(Person1)
000003    // Person2 is a 5 character field
000004
000005    D Quantity     S               10P 0
000006    D TotalQuantity S               +2   Like(Quantity)
000007    // TotalQuantity is a 12 digit packed field with zero decimals

```

© Copyright IBM Corporation 2011

Figure 3-18. LIKE

AS067.0

Notes:

The **LIKE** keyword copies the data type as well as the length and decimal positions of other fields. Length adjustments can be done. The **LIKE** keyword is used to define an item similar to an existing one. When the **LIKE** keyword is specified, the item being defined takes on the data type, length, and the data format of the item specified as the parameter. Standalone fields and others can be defined using this keyword. The parameter of **LIKE** can be a standalone field or certain other data types. The data type entry (position 40) must be blank.

Note that the parameter of **LIKE** could be a field defined in an externally defined database. In this situation, if the characteristics of the database field change, the field that is defined **LIKE** the database field is changed also. Only recompilation of the program would be necessary. No lines of program code would need to be changed.

Machine exercise: Data definition

IBM i



© Copyright IBM Corporation 2011

Figure 3-19. Machine exercise: Data definition

AS067.0

Notes:

Perform the data definition machine exercise.

Checkpoint

IBM i

1. True or False: Named indicators are an effective alternative to numeric indicators as they allow you to use a meaningful field name.

2. Data is defined in an RPG program with:
 - a. Input specifications and program described files
 - b. Externally described files
 - c. D-specification definitions
 - d. All of the above

3. True or False: The D-spec is NOT needed to define fields from an externally described file.

© Copyright IBM Corporation 2011

Figure 3-20. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- List the most common data supported in RPG IV
- Use D specifications to define these data types

© Copyright IBM Corporation 2011

Figure 3-21. Unit summary

AS067.0

Notes:

You have now learned the basics about how data is represented on the i and how you can access it by using RPG IV programs.

Unit 4. Manipulating data in calculations

What this unit is about

This unit describes how you can operate on data in calculations using free-format operation codes and expressions.

Conditional coding is introduced at this point as part of expressions.

One of the most important considerations when coding expressions is how to determine the size of the numeric result of an expression. This is taught as a part of this unit.

In addition, this unit explains RPG IV's rich set of compiler built-in functions so that you can use some of the more common ones to save valuable coding time.

What you should be able to do

After completing this unit, you should be able to:

- Code calculation specifications using free-form operation codes, and expressions for:
 - Assignment of numeric, character, and logical values
 - Arithmetic operations
 - Logical operations
 - String handling operations
- Code built-in functions in expressions
- Determine the size of a numeric result field
- Use H-spec keywords to manage the numeric result of an expression

How you will check your progress

- Checkpoint questions
- Machine exercises
- The students calculate a field's value using an expression (the program is an enhancement to one of the earlier programs written in lab). Also, a final total is calculated.

- Given a report layout, the students code a new program using more expressions to produce the desired results. During the lab, the students code:
 - Arithmetic and logical expressions
 - String handling operations

Unit objectives

IBM i

After completing this unit, you should be able to:

- Code calculation specifications using free-form operation codes, and expressions for:
 - Assignment of numeric, character, and logical values
 - Arithmetic operations
 - Logic operations
 - String handling operations
- Code built-in functions in expressions
- Determine the size of a numeric result field
- Use H-specification keywords to manage the numeric result of an expression

© Copyright IBM Corporation 2012

Figure 4-1. Unit objectives

AS067.0

Notes:

Types of expressions

IBM i

- Assignment or data conversion
- Arithmetic
- Logical
- String

© Copyright IBM Corporation 2012

Figure 4-2. Types of expressions

AS067.0

Notes:

Assignment opcodes

IBM i

- Eval:
 - Assignment using simple expression
 - Result = Expression
 - Supports all data types
 - Data types must be compatible
 - Left-justified with character operands
- EvalR: Same as Eval but:
 - Character only
 - Right-justified

© Copyright IBM Corporation 2012

Figure 4-3. Assignment opcodes

AS067.0

Notes:

The Eval and EvalR operation codes evaluate an assignment statement of the form Eval Result = Expression. The expression is evaluated and the result placed in result. Therefore, result cannot be a literal or constant but is an RPG IV field name. Eval is valid with both numeric and character operands. Operand data types must be compatible. EvalR is valid with character data only and, when used, right-justifies data.

Additional information:

You see the Move opcodes used in legacy applications using fixed-format calculations. Moves are not supported in any free form calculations.

There are three Move opcodes. The Move opcode copies data from the field in factor2 to the result field starting from the rightmost byte of factor2 to the rightmost byte of the result field.

C	Move	Factor2	Result
---	------	---------	--------

The MoveL opcode copies data from the field in factor2 to the result field starting with the leftmost byte of factor2 to the leftmost byte of the result field. Data is moved a byte at a time

and the operation stops when there is no more data to be moved in factor2 or the result field cannot accept any more data because it is full.

MoveA is the third opcode. It copies data to/from arrays. We discuss arrays on a subsequent course.

So, the length of the shorter field determines how much data is moved. RPG IV *does not* clear the result field prior to starting the Move operation. The data in the result field is overlaid by the data from factor2. What this means is that data in the result field that is not affected by the Move (when the result field is longer than factor2) is *not changed*. You can use the operation extender P (pad with blanks) with the Move operation to clear the result field first.

We see examples of operation extenders in this unit.

Assignment of numeric variable

IBM i

```

-----DName*****ETDsFrom***To/L***IDc.Keywords*****
000001  D NumericVar      S          6S 0
000002  D Num2            S          4P 2
000003  /Free
000004
000005  // Assign value to numeric field
000006  NumericVar = 567; // Eval NumericVar = 567;
000007  Num2 = 0 - Num2;
000008
000009  // Set numeric field to zero
000010  NumericVar = 0;
000011  NumericVar = *Zeros;
000012  Clear NumericVar;
000013
000014  *InLR = *off;
000015  /End-Free

```

© Copyright IBM Corporation 2012

Figure 4-4. Assignment of numeric variable

AS067.0

Notes:

A number of capabilities are introduced in this visual:

- The Eval opcode is implied in the expressions above. You can code the Eval using this form:

```
Eval NumericVar = 567;
```

- Numeric assignment using the Eval opcode (implicit).
- Clearing a numeric field using Eval. Notice the use of the constant *0*, the figurative constant **zero* or **zeros*.
- Clearing a numeric field using the **Clear** opcode. **Clear** resets the value of the numeric field to its default initialization value of zero.

Rule of EVAL

The data types on the left and right of an expression must be compatible. For example, you cannot assign a numeric variable to a character variable without having performed a conversion first.

Assignment of character variable

IBM i

Row 1	Column 1	Replace
DName*****ETDsFrom***To/L***IDc.Keywords*****	
000001	D CharVar	S 25A
000002	/Free	
000003	CharVar = 'A_String';	
000004	// CharVar = 'A_String'	
000005		*
000006	EvalR CharVar = 'A_String'; // EvalR must be explicit	
000007	// CharVar = '' A_String'	
000008		*
000009	CharVar = '';	
000010	CharVar = *Blanks;	
000011	EvalR CharVar = *Blanks;	
000012	Clear CharVar;	
000013		*
000014		
000015		
000016		
000017	*InLR = *on;	
000018	/End-Free	

© Copyright IBM Corporation 2012

Figure 4-5. Assignment of character variable

AS067.0

Notes:

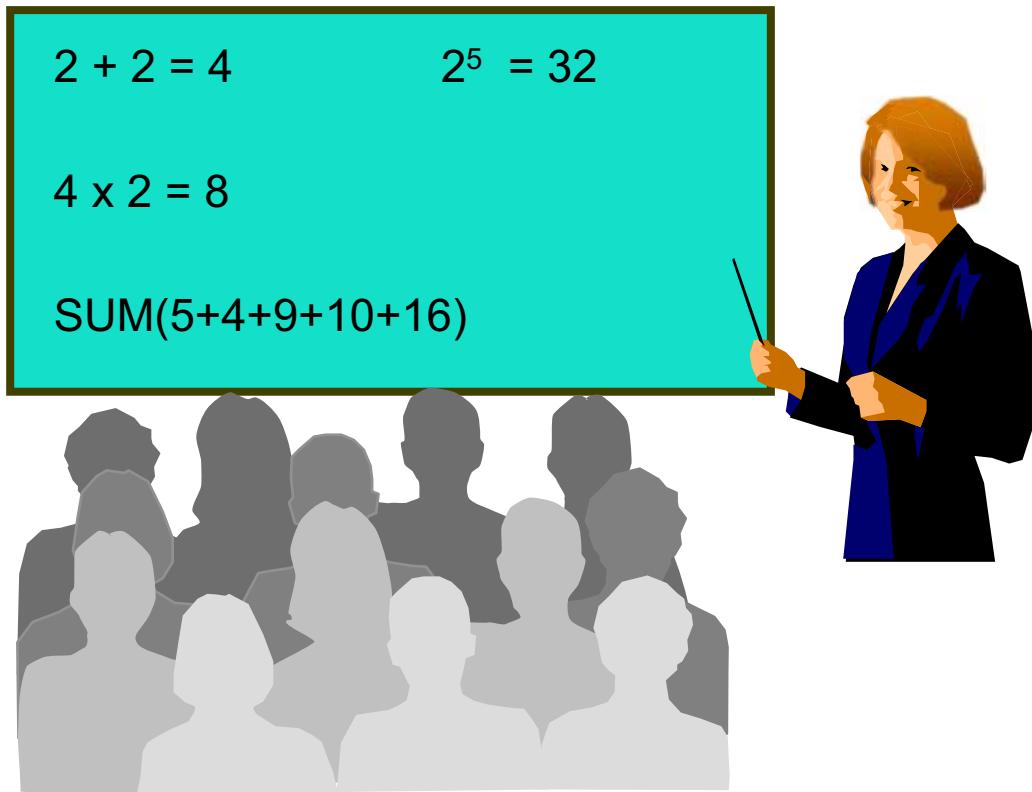
Note the ways that you can set the contents of a character field. Contrast the use of Eval versus EvalR to set the field **CharVar** to "A_String", a character constant. Eval clears the result field and left-justifies the data moved to the result. EvalR also clears the result field and right-justifies the data.

A character field can be cleared in several different ways. Note the use of the constant '' and the figurative constant *blanks.

You can also use the Clear opcode to initialize a character variable. Clear resets the value of a character field to its default initialization value of blanks.

Arithmetic operations

IBM i



© Copyright IBM Corporation 2012

Figure 4-6. Arithmetic operations

AS067.0

Notes:

Manipulating data also requires various arithmetic functions such as the usual functions of addition, subtraction, multiplication, and division as well as more unusual functions such as squaring, square roots, logarithms, tangents, percentages, and so forth. Let's see how RPG IV handles these functions.

Arithmetic opcodes

IBM i

- Legacy opcodes:
 - ADD
 - SUB
 - MULT
 - DIV
 - MVR
 - XFOOT
 - SQRT
 - Z-ADD
 - Z-SUB
- Best coded using EVALs and BiFs

© Copyright IBM Corporation 2012

Figure 4-7. Arithmetic opcodes

AS067.0

Notes:

These legacy operation codes are valid only with numeric fields, including numeric subfields, parameters, arrays, individual array elements, table elements and numeric literals.

Legacy opcodes:

- **ADD:** Addition of one field to another
- **SUB:** Subtraction of one field from another
- **MULT:** Multiply one field by another
- **DIV:** Divide one field by another
- **MVR:** Move the remainder following a divide into a field
- **XFOOT:** Crossfoot a set of fields (elements) in an array
- **SQRT:** Calculate the square root of a field
- **Z-ADD:** Zero the result field and add the value of another into the result
- **Z-SUB:** Zero the result field and subtract the value of another into the result

The i preferred numeric format is packed data. This means that the system converts all numeric data to be operated on into packed decimal before operating on it. Then, the

system converts it back to its original format when done. Remember this when you are building databases that define numeric data.

The sign of the result field is determined algebraically and is set to an F if positive and to a D if negative.

An RPG IV numeric **Packed or Zoned Decimal** field can be a maximum of 63 digits (V5R3; prior releases maximum is 30 digits) in length. The limits for other numeric data types are:

Binary - 4 or 9 digits

Integer - +/- 3, 5, 10 or 20 digits

Unsigned - +3, 5, 10 or 20 digits

Remember to define your result fields with adequate length and decimal places (precision) for the calculation.

The Xfoot (crossfoot) opcode sums are a set of numeric data stored in an array.

Crossfooting is discussed in the “**Arrays and Tables**” unit in AS07/S6198.

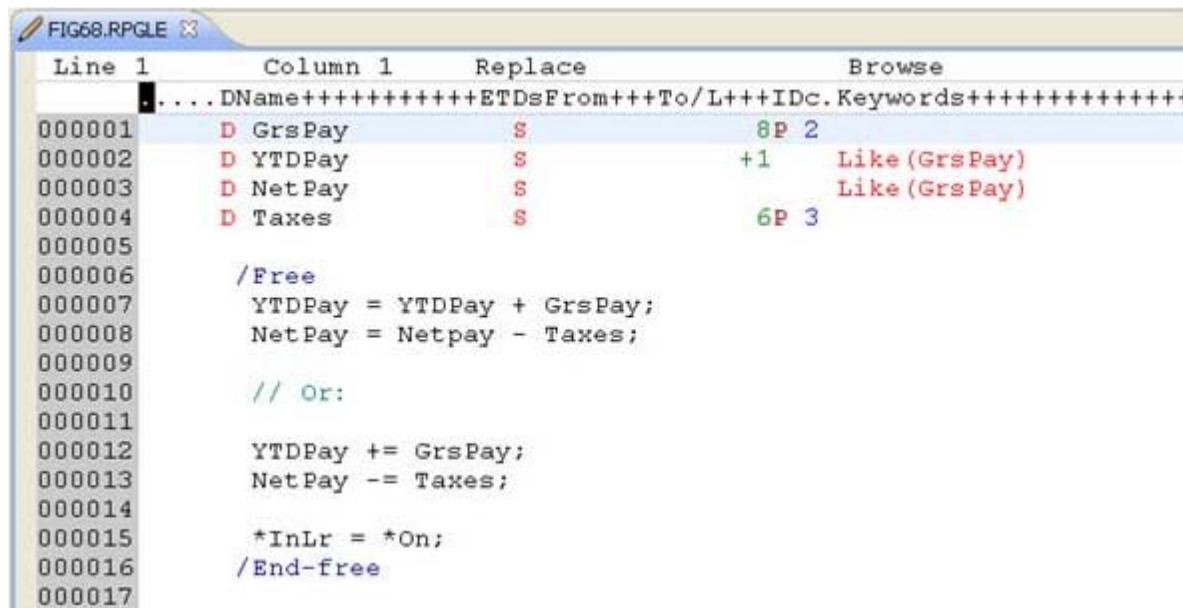
It is important for you to be aware of the legacy operation codes. In this class, we do not spend any time using them. We use Eval and built-in functions throughout the class.

Reference:

ILE RPG Language Reference, Chapter 22, Operation Codes Detail

Addition and subtraction

IBM i



```

FIG68.RPGLE X
Line 1      Column 1      Replace      Browse
. .... DName++++++ETDsFrom+++To/L+++IDc. Keywords+++++
000001     D GrsPay       S           8P 2
000002     D YTDPay       S           +1    Like (GrsPay)
000003     D NetPay       S           Like (GrsPay)
000004     D Taxes         S           6P 3
000005
000006     /Free
000007     YTDPay = YTDPay + GrsPay;
000008     NetPay = Netpay - Taxes;
000009
000010    // Or:
000011
000012    YTDPay += GrsPay;
000013    NetPay -= Taxes;
000014
000015    *InLr = *On;
000016    /End-free
000017

```

© Copyright IBM Corporation 2012

Figure 4-8. Addition and subtraction

AS067.0

Notes:

Note that the use of free-format coding (implicit Eval) makes your program very easy to read and understand.

In the lower portion of the visual, notice two unary operations. They reduce the number of keystrokes required when compared to the statements above them. Reaction to these unary operations has been mixed as they tend to make the purpose of the statement less clear than if the complete expression is coded.

These are all the unary operations:

```

+=
-=
*=
/=
**=

```

The following table lists the operators with examples of both short and long form. Notice that when you use short form, the operator must be coded with no blanks. For example, `+ =` is correct, but `+ =` will generate an error.

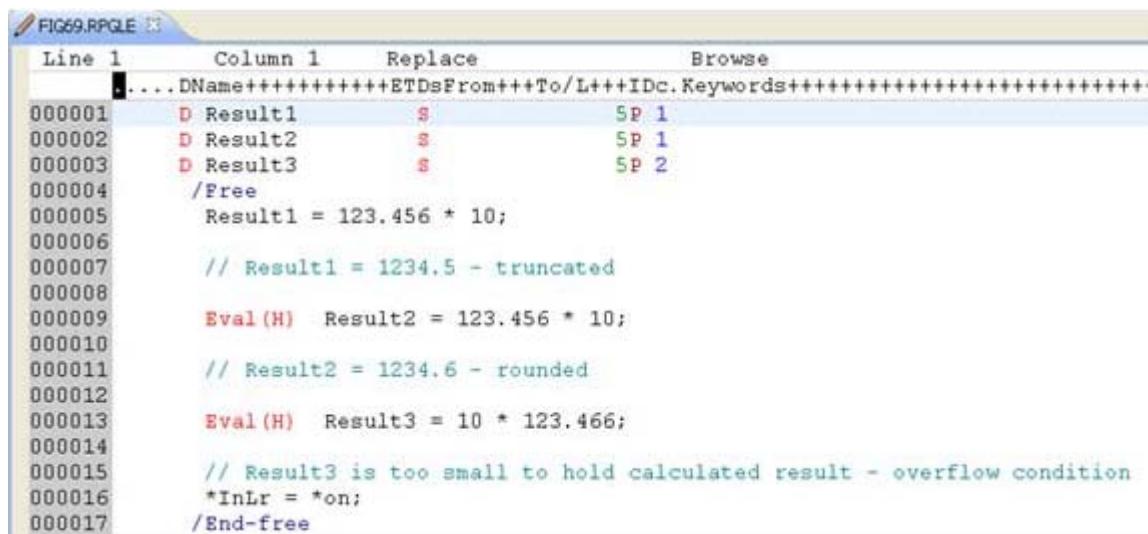
Short form operations during conversion

Table 1:

Operation	Complete Form	Short Form
Add	<code>a = a + b</code>	<code>a += b</code>
Subtract	<code>a = a - b</code>	<code>a -= b</code>
Multiply	<code>a = a * b</code>	<code>a *= b</code>
Divide	<code>a = a / b</code>	<code>a /= b</code>
Exponentiate	<code>a = a ** b</code>	<code>a **= b</code>

Multiplication

IBM i



```

FIG69.RPGLE
Line 1   Column 1   Replace   Browse
000001   D Result1    S      SP_1
000002   D Result2    S      SP_1
000003   D Result3    S      SP_2
000004   /Free
000005     Result1 = 123.456 * 10;
000006
000007     // Result1 = 1234.5 - truncated
000008
000009     Eval(H)  Result2 = 123.456 * 10;
000010
000011     // Result2 = 1234.6 - rounded
000012
000013     Eval(H)  Result3 = 10 * 123.466;
000014
000015     // Result3 is too small to hold calculated result - overflow condition
000016     *InLr = *on;
000017   /End-free

```

- When using operation extender, EVAL must be explicit

© Copyright IBM Corporation 2012

Figure 4-9. Multiplication

AS067.0

Notes:

These are several simple calculations to show you how to perform multiplications. They also show that you must make sure that your result fields are large enough to hold both the whole number and the decimal result of your calculations.

The **H** is known as an operation extender. The H operation extender rounds up the arithmetic result and then places it in the result field.

When you code the H or any other operation extender for an expression, you must code the Eval explicitly. If no extender is desired, you may omit the Eval as it is implied in the free format statement.

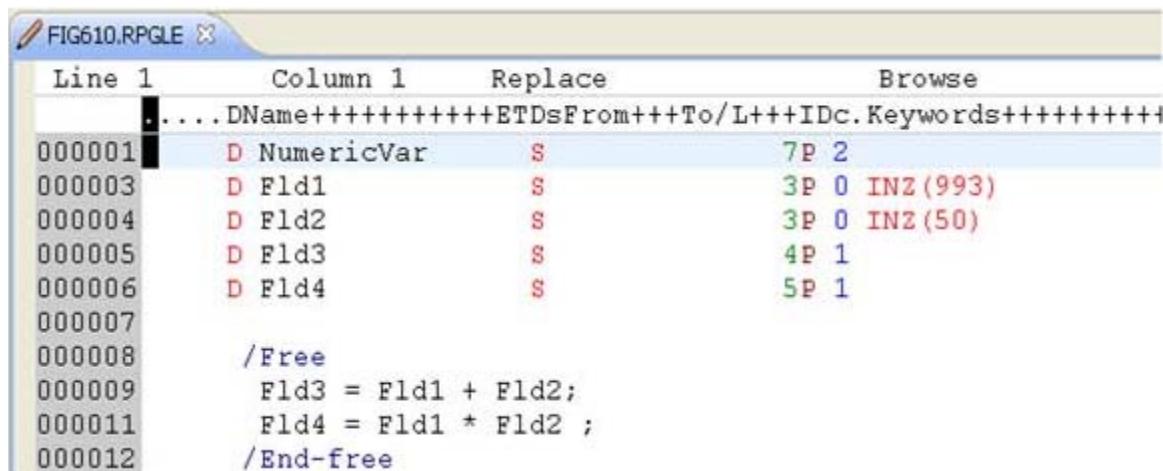
But, there is a problem with the last line of code. While it is valid format, when executed, you would get an execution time error like this:

RPG procedure performed an arithmetic operation which resulted in a value that is too large to fit in the target. If this is a numeric expression, the overflow could be the result of the calculation of some intermediate result.

Result3 is 5 digits with three positions before the decimal and two reserved for decimal places. The value calculated, 1234.66, requires four positions before the decimal. An error condition results.

Numeric overflow

IBM i



The screenshot shows an IBM i terminal window titled 'FIG610.RPGLE'. The code is as follows:

```

Line 1      Column 1      Replace      Browse
. .... DName++++++ETDsFrom++To/L+++IDc. Keywords+++++
000001    D NumericVar     S          7P 2
000003    D Fld1           S          3P 0 INZ(993)
000004    D Fld2           S          3P 0 INZ(50)
000005    D Fld3           S          4P 1
000006    D Fld4           S          5P 1
000007
000008    /Free
000009    Fld3 = Fld1 + Fld2;
000011    Fld4 = Fld1 * Fld2 ;
000012    /End-free

```

© Copyright IBM Corporation 2012

Figure 4-10. Numeric overflow

AS067.0

Notes:

When numeric overflow occurs in a free-form expression, a run-time error occurs. To avoid this problem, you should determine adequate field sizes for every field that is the result of an accumulation or multiplication using the following guidelines:

- **Addition**

The result field should be one digit larger to the left of the decimal than the larger field in the expression.

Example:

```
FldA is (4 2), that is, 99.99 (max. value)
FldB is (5 2), that is, 999.99 (max. value)
```

```
FldC = FldA + FldB (FldC = 1099.98)
```

```
Define FldC as (6 2) in order that it can hold 9999.99
```

- **Multiplication**

The result field should be two digits larger to the left of the decimal than the largest field in the operation.

Example:

```
FldA is (4 2)
```

```
FldB is (5 2)
```

```
FldC = FldA * FldB
```

Define FldC as (7 2) in order that it can hold 99999.99



Note

When your expressions include fields of different decimal positions, the rightmost digits are truncated based upon the number of decimal positions defined for your result field that contain the new value. You may round up using the half adjustment feature of RPG IV, the H operation extender.

In the *ILE RPG for i Reference*, read about the TRUNCNBR option of the RPG IV compile commands. It is also a keyword on the H-spec. TRUNCNBR specifies whether the truncated value is moved to the result field or an error is generated when numeric overflow occurs while running the program.

TRUNCNBR does *not* apply to calculations performed within expressions. If any overflow occurs within expressions calculations, a run-time message is issued.

In addition, TRUNCNBR does not apply to *any* arithmetic operations performed in integer or unsigned format.

Division and remainders

IBM i

```

000001 D NumericVar S      5P 2
000002 D Dividend S      5P 2 INZ(200.25)
000003 D Remainder S      5P 2
000004 D Months S      3P 0 INZ(54)
000005 D Years S      3P 0 INZ
000006 D RemMon S      2P 0 INZ
000007
000008 /Free
000009 >>1 NumericVar = 200 / 3; // NumericVar = 66.66
000010     Remainder = 200 - (NumericVar * 3); // Remainder = 000.02
000011
000012 >>2 Eval(H) NumericVar = 200 / 3; // NumericVar = 66.67
000013     Remainder = 200 - (NumericVar * 3); // Remainder = -000.01
000014
000015 >>3 NumericVar = %div(200:3); // NumericVar = 66.00
000016     Remainder = %rem(200:3); // Remainder = 2.00
000017
000018 >>4 Remainder = 200 - ((200/3) * 3); // Remainder = 0.00
000019
000020 >>5 Remainder = 200 - (%div(200 : 3) * 3); // Remainder = 2.00
000021
000022 >>6 Remainder = Dividend - (%div(%Int(dividend) : 3) * 3);
000023 // Remainder = 2.25
000024
000025 >>7 Years = Months / 12; // Years = 4
000026 RemMon = %rem(Months : 12); // PartMon = 6
000027
000028 *InLR= *on;
000029 /End-free

```

© Copyright IBM Corporation 2012

Figure 4-11. Division and remainders

AS067.0

Notes:

Free-form division is done using the familiar slash (/) character. The use of the H operation extender (half adjust) rounds up the result of the calculation prior to placing it in the result field.

In this visual, notice the code with the %div and %rem. These are known as built-in functions or, simply, BiFs. BiFs are included in the RPG IV compiler. We will be using them throughout the course.

Notice that you pass parameters to built-in functions. Parameters are enclosed in parentheses and are separated by a colon when two or more parameters are passed. A BiF returns a specific value when it is evaluated.

The two BiFs in this example:

- **%DIV** expects two parameters, where the first is divided by the second. %DIV returns the integer result of the division.
- **%REM** expects two parameters, where the first is divided by the second. %REM returns the remainder of the division as an integer.

Here are some further notes about the visual. The notes are based on the reference numbers in the figure:

1. We know that $200/3 = 66.66666667$ depending on how many decimal places you want to carry. In this example, we are not rounding and we carry only two decimal places. Therefore, the result of the division is 66.66 and the corresponding remainder is 000.02.
2. In this example, we use rounding and still carry two decimals. Therefore, the result of the division is 66.67 and the remainder is -000.01 ($66.67 \times 3 = 200.01$).
3. %Div and %Rem are handy tools but they return only integer values.
4. When you attempt to code an expression in this way, you must always be aware of what are called *intermediate results*. These have a dramatic influence on your final result because of the precision rules. We discuss them later.
5. Again, note that %Div returns an integer result. No fractions are returned.
6. This is a more accurate way of determining the remainder, but we are still limited to an integer result. Notice the use of nesting of functions. We will see more of this later.
7. This is another example of division. Because Years has no decimal places, Months / 12 will return the same result as %Div(Months : 12).

Coding BiFs

IBM i

- Coding syntax:
 - `%function-name{ (argument{:argument...}) }`
- Arguments can be:
 - Variables
 - Constants
 - Expressions
 - Prototyped procedure
 - BiFs
- Examples:
 - `%Div(200 : 3)`
 - `%Rem(200 : 3)`
 - `%Eof(EmpMaster)`
 - `%TrimL('? *' : String)`

© Copyright IBM Corporation 2012

Figure 4-12. Coding BiFs

AS067.0

Notes:

Built-in functions add tremendous function to the RPG IV language. All built-in functions have the percent symbol (%) as their first character. The syntax of built-in functions is:

```
%function-name{ (argument{:argument...}) }
```

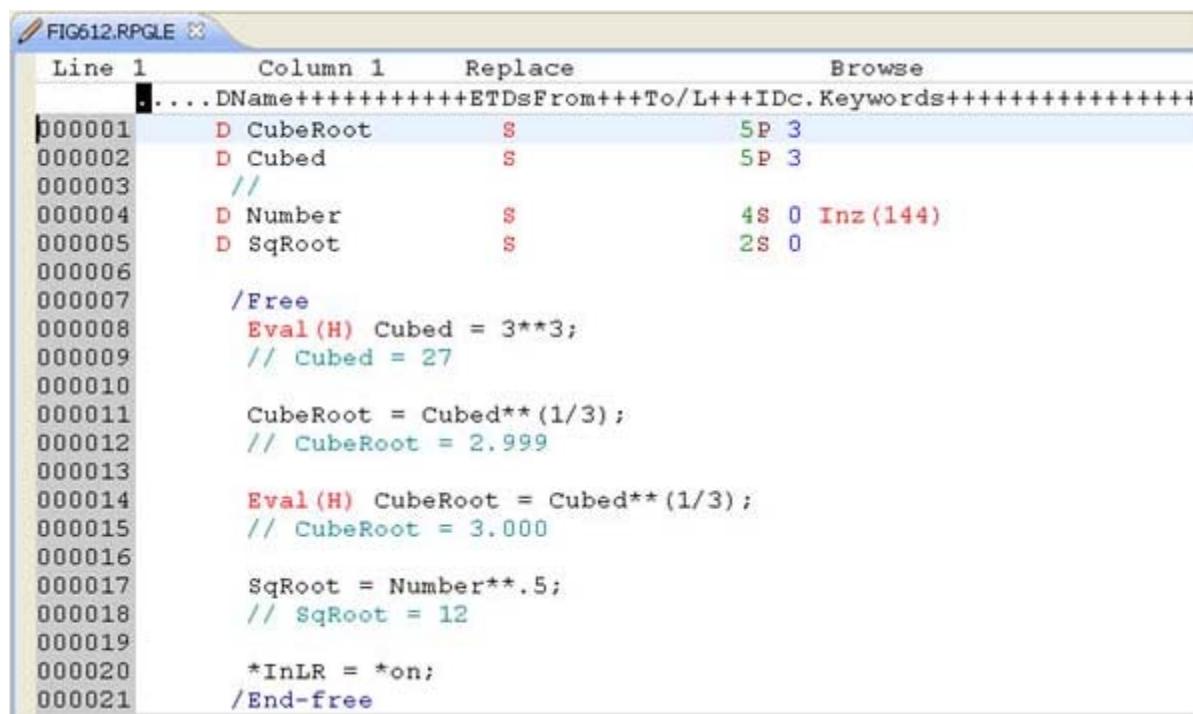
Arguments for the function may be variables, constants, expressions, a prototyped procedure, or other built-in functions. (Procedures are covered on a subsequent course.)

An expression argument can include a built-in function. BiFs can be nested and will work as long as the result of the nested BiF produces a valid parameter for the outer BiF.

BiFs can be used on the D-spec, C-spec in expressions or in free-format calculations. They cannot be used with legacy fixed-format opcodes.

Exponentiation

IBM i



```

FIG612.RPGLE X
Line 1      Column 1      Replace      Browse
. .... DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
p000001    D CubeRoot     S           5P 3
000002    D Cubed        S           5P 3
000003    //
000004    D Number        S           4S 0 Inz(144)
000005    D SqRoot        S           2S 0
000006
000007    /Free
000008    Eval(H) Cubed = 3**3;
000009    // Cubed = 27
000010
000011    CubeRoot = Cubed**(1/3);
000012    // CubeRoot = 2.999
000013
000014    Eval(H) CubeRoot = Cubed**(1/3);
000015    // CubeRoot = 3.000
000016
000017    SqRoot = Number**.5;
000018    // SqRoot = 12
000019
000020    *InLR = *on;
000021    /End-free

```

© Copyright IBM Corporation 2012

Figure 4-13. Exponentiation

AS067.0

Notes:

This visual illustrates how the **Eval (in explicit and implicit forms)** opcode is able to perform exponentiation. Exponentiation can be used to extract square and cube roots, for example.

Machine exercise: Adding arithmetic function

IBM i



© Copyright IBM Corporation 2012

Figure 4-14. Machine exercise: Adding arithmetic function

AS067.0

Notes:

Perform the adding arithmetic function machine exercise.

Comparison operations

IBM i

- $x = y$
- $x <> y$
- $x > y$
- $x \geq y$
- $x < y$
- $x \leq y$
- $x \text{ AND } y$
- $x \text{ OR } y$
- $x \text{ NOT } y$
- Equality
- Not equal
- Greater Than
- Greater Than or Equal
- Less Than
- Less Than or Equal
- Logical and
- Logical or
- Logical not

© Copyright IBM Corporation 2012

Figure 4-15. Comparison operations

AS067.0

Notes:

These operations are valid in expressions in RPG IV:

- = The *equality* operation returns **1** if the two operands are equal, and **0** if not equal.
- <> The *not equal (inequality)* operation returns **0** if the two operands are equal, and **1** if not equal.
- > The *greater than* operation returns **1** if the first operand is greater than the second.
- \geq The *greater than or equal* operation returns **1** if the first operand is greater or equal to the second.
- < The *less than* operation returns **1** if the first operand is less than the second.
- \leq The *less than or equal* operation returns **1** if the first operand is less or equal to the second.
- AND** The *logical and* operation returns **1** if both operands return a value of the indicator operand of **1**.

OR The *logical or* operation returns **1** if either operand returns a value of the indicator operand of **1**.

NOT The *logical not* operation is used with an indicator operand. It returns **1** if the value of the indicator operand is **0** and **0** if the indicator operand is **1**.

Expressions are used not only with the EVAL operation code, but also with conditional opcodes. We introduce the **If** opcode in this topic and cover other conditional opcodes in the *Structured Programming and Subroutines* unit.

Class exercise

IBM i

FIG615.RPGLE X

Line 1	Column 1	Replace	Browse
DName++++++ETDsFrom+++To/L+++IDc.Keywords++++++-		
000001	D Indicator1	S	N
000002	D Indicator2	S	N
000003	D X	S	5A
000004	D Y	S	Like (X)
000005	D Z	S	7 P 2
000006	/Free		
000007			
000008	1 Indicator1 = *On;		
000009	2 Indicator1 = (X > Y);		
000010	3 Indicator1 = (X > Y) AND (Z <> 0);		
000011	4 Indicator1 = ((X > Y) AND (Z <> 0)) OR Indicator2;		
000012	5 Indicator1 = (X > Y) OR NOT Indicator2;		
000013	*InLr = *On;		
000014	/End-free		
000015			
000016			
000017			
000018			
000019			

© Copyright IBM Corporation 2012

Figure 4-16. Class exercise

AS067.0

Notes:

Indicator1 and Indicator2 are defined as named indicators. Write an explanation that describes what is being done by each of the statements above:

1

2

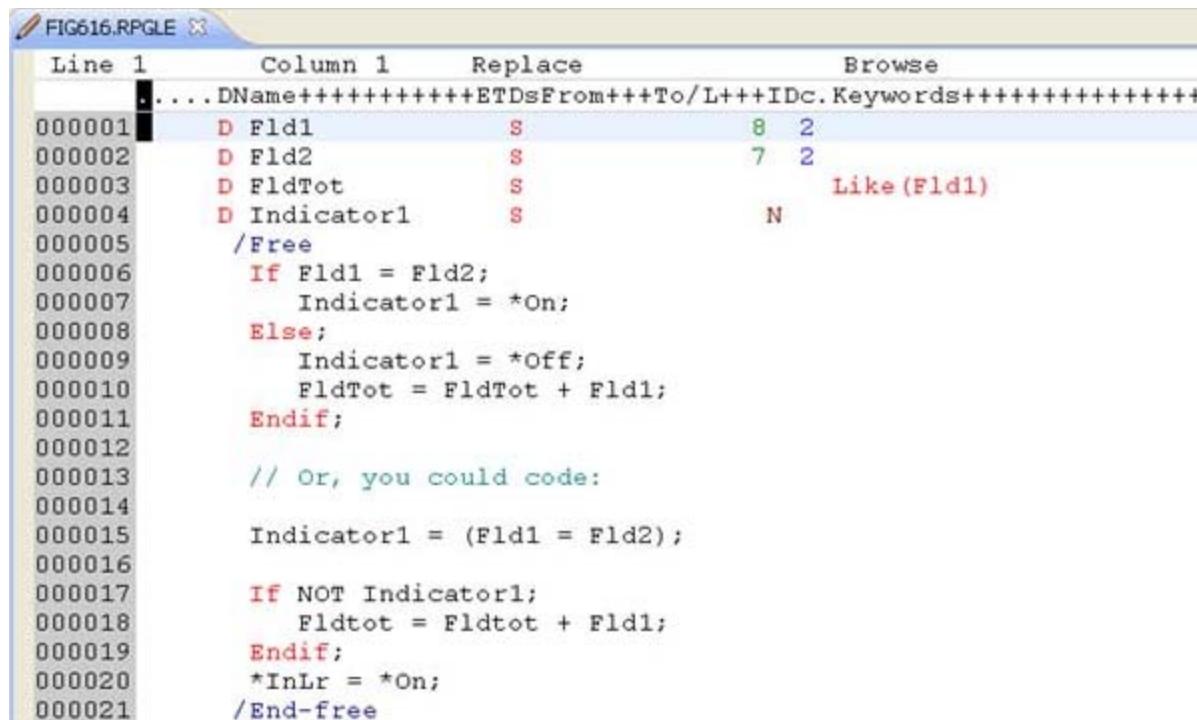
3

4

5

If/else/endIf

IBM i



The screenshot shows an IBM i editor window titled 'FIG616.RPGLE'. The code is as follows:

```

Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom+++To/L+++IDc.Keywords+++++
000001     D Fld1          S           8 2
000002     D Fld2          S           7 2
000003     D FldTot        S           Like(Fld1)
000004     D Indicator1    S           N
000005     /Free
000006     If Fld1 = Fld2;
000007       Indicator1 = *On;
000008     Else;
000009       Indicator1 = *Off;
000010       FldTot = FldTot + Fld1;
000011     Endif;
000012
000013 // Or, you could code:
000014
000015   Indicator1 = (Fld1 = Fld2);
000016
000017   If NOT Indicator1;
000018     Fldtot = Fldtot + Fld1;
000019   Endif;
000020   *InLr = *On;
000021 /End-free

```

© Copyright IBM Corporation 2012

Figure 4-17. If/else/endIf

AS067.0

Notes:

The If operation code allows a series of calculations that follow it to be processed if a condition is met. The logical condition is expressed by an indicator-valued expression (the result of the test of the expression is a **1** or a **0**). The operations controlled by the If operation are performed when the expression in the extended factor 2 field is true.

If statements are coded including an expression, such as those shown in this visual.

The rules for the comparison of an expression in the IF are the same as they are for any comparison. The fields must be the same type. Each IF is delimited by an End or ENDIF. If the result of the IF is true, then the code before the ELSE is executed. If the comparison is false, the code between the IF and the ELSE is skipped and the code after the ELSE is executed.

IF expressions

IBM i

```

000001      D Indicator1      S          N
000002      D Indicator2      S          N
000003      D X              S          SA
000004      D Y              S          Like(X)
000005      D Z              S          7P 2
000006
000007      /Free
000008          If Indicator1;
000009              // More logic
000010          Else;
000011              // More logic
000012              If (X < Y);
000013                  // More logic
000014          Else;
000015              // More logic
000016              If (X > Y) AND (Z <> 0);
000017                  // More logic
000018          Else;
000019              // More logic
000020              If ((X = Y) AND (Z < 0)) OR Indicator2;
000021                  // More logic
000022          EndIf;
000023      EndIf;
000024  EndIf;
000025
000026
000027      *inlr = *on;
000028  /End-free

```

© Copyright IBM Corporation 2012

Figure 4-18. IF expressions

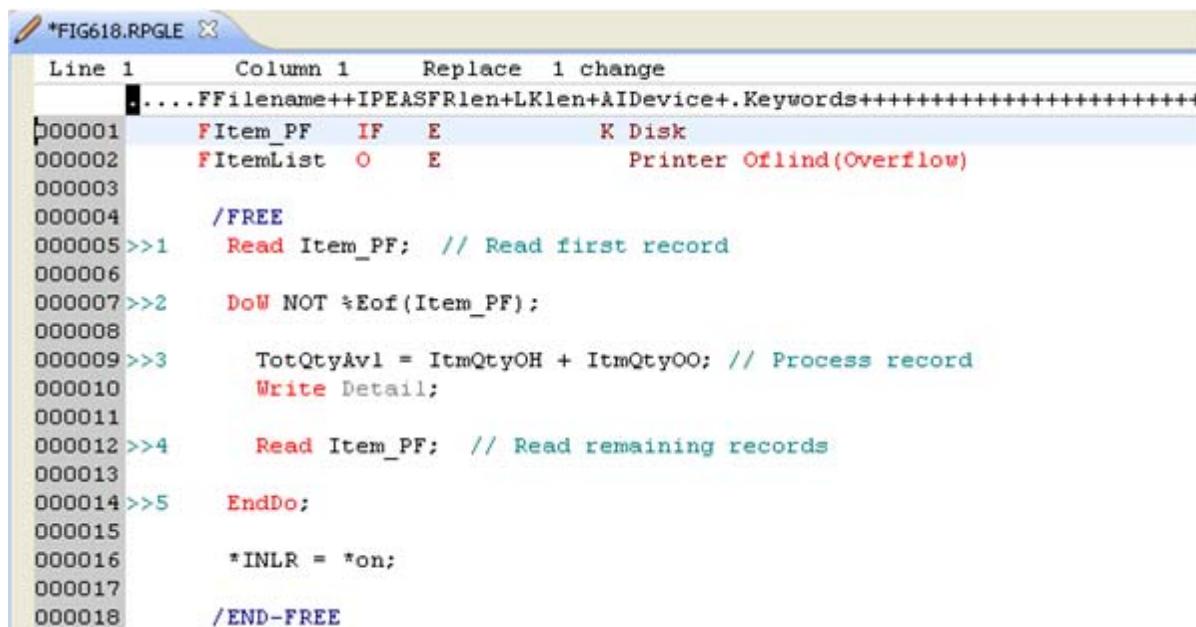
AS067.0

Notes:

Note not only the use of a conditional expression with the IF operation, but also that each IF is matched with an ENDIF.

Using do while

IBM i



```
*FIG618.RPGL X
Line 1      Column 1      Replace 1 change
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001    FItem_PF  IF   E          K Disk
000002    FItemList O   E          Printer Ofind(Overflow)
000003
000004    /FREE
000005 >>1    Read Item_PF; // Read first record
000006
000007 >>2    DoW NOT %Eof(Item_PF);
000008
000009 >>3    TotQtyAvl = ItmQtyOH + ItmQtyOO; // Process record
000010    Write Detail;
000011
000012 >>4    Read Item_PF; // Read remaining records
000013
000014 >>5    EndDo;
000015
000016    *INLR = *on;
000017
000018    /END-FREE
```

© Copyright IBM Corporation 2012

Figure 4-19. Using do while

AS067.0

Notes:

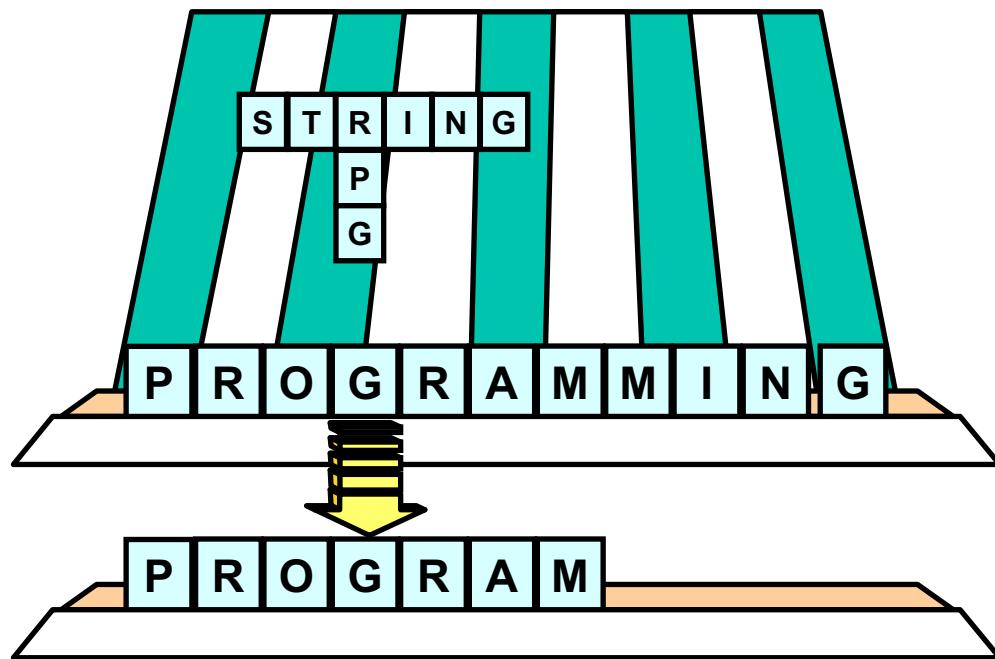
Use Do loops to perform calculations that are to be repeated as long as a certain condition is true. One of the most commonly used forms of DO is the DoW operation code. The logic between the DoW (2) and the EndDo (5) is performed as long as the expression to the right of the DoW is true. For this form of Do, the expression is tested before the logic is performed.

In this example, notice that we first perform a read of the `Item_PF` file. The reason we do this is to find out whether there are any records in the file. In the DoW (2), notice that the expression tests for end-of-file using the `%Eof` built-in function.

In the loop, we process each record (3), starting with the first one (which we have already read). Then we read the second through the last records (4). When we reach end of file, we drop out of the loop and end the program (5).

String handling

IBM i



© Copyright IBM Corporation 2012

Figure 4-20. String handling

AS067.0

Notes:

A common programming task is to manipulate character string data. RPG IV offers a number of features to make this simple for you to code and read when you are maintaining existing code.

String handling BiFs

IBM i

BiF	Purpose
%CHAR	Convert to character
%DEC	Convert to decimal
%INT	Convert to integer
%EDITC	Edit numeric using edit code
%EDITW	Edit numeric using edit work
%REPLACE	Replaces all or part of character string
%SCANRPL	Scan/replace search argument against character string
%SUBST	Extracts substring from a character string
%TRIM	Removes left and right characters from a string
%TRIML	Removes left characters from a string
%TRIMR	Removes right characters from a string

© Copyright IBM Corporation 2012

Figure 4-21. String handling BiFs

AS067.0

Notes:

Built-in functions are IBM-supplied routines that perform many popular functions. IBM-supplied BiFs always start with the % character.

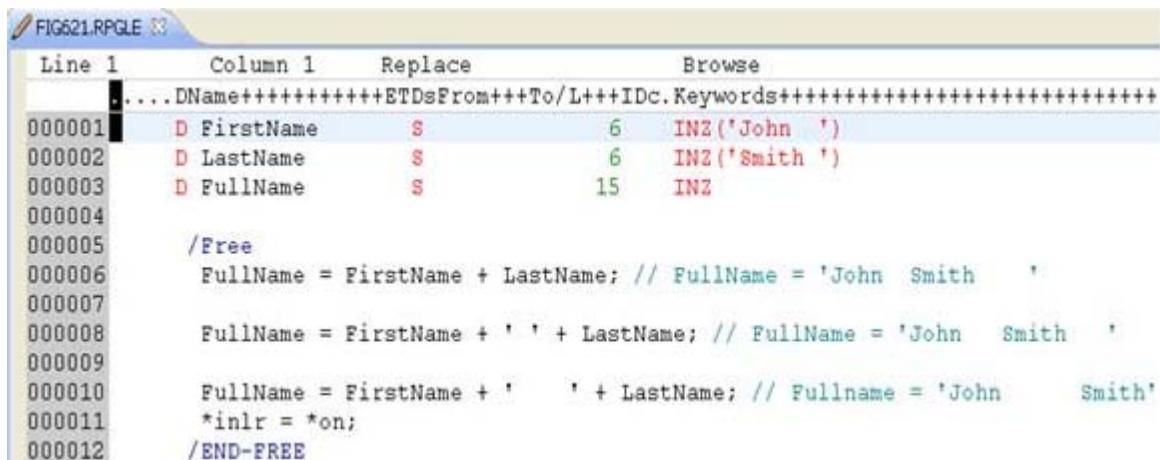
In this topic, we explain some of the BiFs that perform string handling. Other BiFs are discussed as we encounter them in later topics.

Reference:

ILE RPG Reference, Chapter 21, Built-in Functions

Concatenation

IBM i



The screenshot shows an IBM i terminal window titled 'FIG621.RPGLE'. The code is as follows:

```

Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom+++To/L+++IDc.Keywords+++++
000001      D FirstName    S          6   INZ('John  ')
000002      D LastName     S          6   INZ('Smith ')
000003      D FullName     S          15  INZ
000004
000005      /Free
000006      FullName = FirstName + LastName; // FullName = 'John  Smith  '
000007
000008      FullName = FirstName + ' ' + LastName; // FullName = 'John  Smith  '
000009
000010      FullName = FirstName + ' ' + LastName; // Fullname = 'John      Smith'
000011      *inlr = *on;
000012      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 4-22. Concatenation

AS067.0

Notes:

Notice that the (implicit) Eval opcode does not strip any blank characters from the result field. It simply concatenates strings.

Notice that the fields **FirstName** and **LastName** are each six characters in length. Each field's initial value is padded on the right with one or more blanks.

Question: In the last example, can you explain why the blank after 'Smith' in **LastName** is truncated?

Remove characters: %TRIM

IBM i

```
%TRIM(string:{chars})    %TRIML(string:{chars})
%TRIMR(string:{chars})
```

- %TRIM strips leading and trailing characters from a string (default blanks)
- %TRIML strips leading characters only (default blanks)
- %TRIMR strips trailing characters only (default blanks)

The screenshot shows an IBM i editor window with the following code:

```

Line 23  Column 1  Insert
.....1.....+....2.....+....3.....+....4.....+....5.....+....6.....+....7.....+....8.....+....9.....+....0
000001  D AString      S     17A  Inz('This is it. ')
000002  D Another      S     12A  Inz
000003  D YetAnother   S     20A  Inz
000004  D BigOne       S     26A  Inz
000005  D OneMore      S     10A  Inz
000006
000007  D FirstName    S      6   Inz('John ')
000008  D LastName     S      6   Inz(' Smith ')
000009  D FullName     S     15   Inz
000010
000011
000012 /FREE
000013  Another = %TrimL(AString);           // Another = 'This is it. '
000014  YetAnother = %TrimR(AString) + '????'; // YetAnother = ' This is it???? '
000015  OneMore = %Trim(YetAnother:'?');        // OneMore = 'This is it'
000016  BigOne = %TrimR(AString) + %TrimL(AString); // BigOne = ' This is itThis is it '
000017
000018  FullName = %TrimR.LastName)
000019          + ','
000020          + %TrimR.FirstName);           // FullName = 'Smith, John'
000021
000022  *InLR = *On;
000023 /END-FREE
000024
000025
000026
000027

```

© Copyright IBM Corporation 2012

Figure 4-23. Remove characters: %TRIM

AS067.0

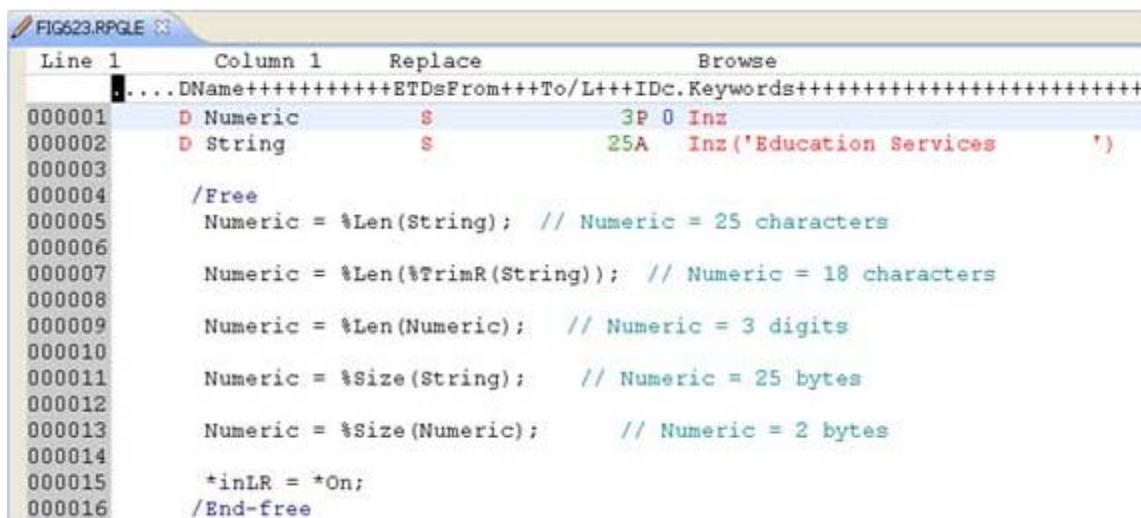
Notes:

The TRIMx built-in functions allow you to strip characters from the beginning or end of a character variable. Note that the results can be used wherever a character field can be used, including in a string manipulation expression, as in the example.

The characters to be stripped are defined as an optional second argument. If the second argument is not specified, blanks are assumed.

Length and size of a string: %Len / %Size

IBM i



```

FIG623.RPGLE X
Line 1      Column 1      Replace      Browse
....DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
000001 D Numeric      S           3P 0 Inz
000002 D String        S           25A   Inz('Education Services')
000003
000004 /Free
000005     Numeric = %Len(String); // Numeric = 25 characters
000006
000007     Numeric = %Len(%TrimR(String)); // Numeric = 18 characters
000008
000009     Numeric = %Len(Numeric); // Numeric = 3 digits
000010
000011     Numeric = %Size(String); // Numeric = 25 bytes
000012
000013     Numeric = %Size(Numeric); // Numeric = 2 bytes
000014
000015     *inLR = *On;
000016 /End-free

```

© Copyright IBM Corporation 2012

Figure 4-24. Length and size of a string: %Len / %Size

AS067.0

Notes:

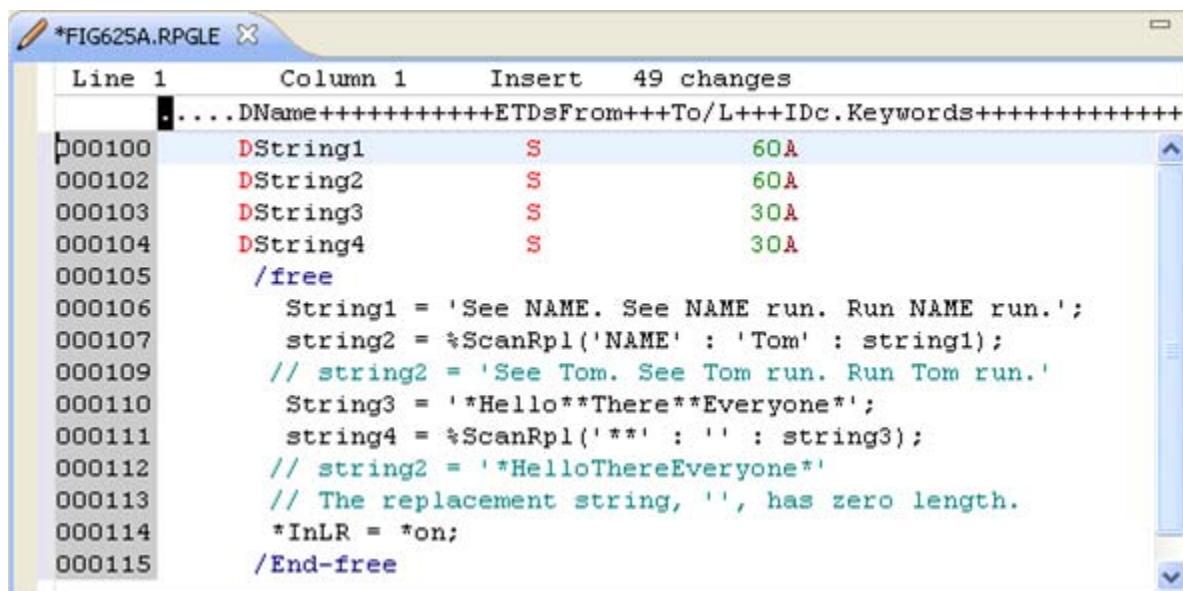
In the second example, we strip trailing blanks using %TrimR and use %Len to determine the length of the trimmed string. %Len is not sensitive to any blanks in the middle of the string value.

Then we contrast the use of %Len and %Size. Whereas %Size returns the allocated size of a field as a number of bytes, %Len returns the declared length of a field as a number of characters or digits.

Scan and replace for character string: %Scanrpl

IBM i

```
%SCANRPL(search argument : replacement string :
          source string {: start} {:length})
```



The screenshot shows an IBM i editor window titled '*FIG625A.RPGLE'. The code is as follows:

```

Line 1      Column 1      Insert   49 changes
.....DName++++++ETDsFrom+++To/L+++IDc.Keywords+++++
b00100    DString1      S       60A
000102    DString2      S       60A
000103    DString3      S       30A
000104    DString4      S       30A
000105    /free
000106    String1 = 'See NAME. See NAME run. Run NAME run.';
000107    string2 = %ScanRpl('NAME' : 'Tom' : string1);
000109    // string2 = 'See Tom. See Tom run. Run Tom run.'
000110    String3 = '*Hello**There**Everyone*';
000111    string4 = %ScanRpl('*' : '' : string3);
000112    // string2 = '*HelloThereEveryone*'
000113    // The replacement string, '', has zero length.
000114    *InLR = *on;
000115    /End-free

```

© Copyright IBM Corporation 2012

Figure 4-25. Scan and replace for character string: %Scanrpl

AS067.0

Notes:

%SCANRPL returns the string produced by replacing all occurrences of the search argument in the source string with the replacement string. The search for the search argument starts at the scan start position and continues for the scan length. The parts of the source string that are outside the range specified by the scan start position and the scan length are included in the result.

The first, second and third parameters must be character, graphic, or UCS-2 data types. They can be in either fixed-length or variable-length format. These parameters must all be of the same type and CCSID.

The fourth parameter represents the scan starting position, measured in characters, where the search for the scan string should begin. If it is not specified, the starting position defaults to one. The value may range from one to the current length of the source string.

The fifth parameter represents the number of characters in the source string to be scanned. If the parameter is not specified, the length defaults to the remainder of the source string starting from the start position. The value must be greater than or equal to zero, and less than or equal to the remaining length of the source string starting at the start position.

The starting position and length may be any numeric value or numeric expression with no decimal positions.

The returned value may be larger, equal to or smaller than the source string. The resulting length depends on the lengths of the scan string and the replacement string, and also on the number of times the replacement is performed. For example, assume the scan string is 'a' and the replacement string is 'bc'. If the source string is 'ada', the returned value has a length of five ('bcdcb'). If the source string is 'ddd', the returned value has a length of three ('ddd').

The returned value is varying length if the source string and replacement string have different lengths, or if any of the strings are varying length. Otherwise, the returned value is fixed length. The returned value has the same type as the source string.

Scan for character string: %Scan (1 of 2)

IBM i

```
%SCAN(search argument : source string {: start})
```

```

FIG624.RPGLE 3
Line 1      Column 1      Replace      Browse
.....1.....2.....3.....4.....5.....6.....7.

000001
000002      D Position      S          2P 0 Inz
000003      D String1       S          7A   Inz('ABCDEFG')
000004      D String2       S          2A   Inz('RS')
000005      D String3       S          5A   Inz('TUVXY')
000006      D String4       S          25A  Inz('Learn')
000007
000008      /Free
000009          Position = %Scan('CD' : String1);           // Position = 3
000010
000011          Position = %Scan(String2 : String3 : 3); // Position = 0
000012
000013      If %Scan(%Trim(String4) : String3) = *Zero;
000014          // (more logic here ....)
000015      EndIf;
000016
000017      *InLr = *On;
000018      /End-Free
000019

```

© Copyright IBM Corporation 2012

Figure 4-26. Scan for character string: %Scan (1 of 2)

AS067.0

Notes:

%SCAN returns the first position of the search argument in the source string where the search string was found, or 0 if it was not found. If the start position is specified, the search begins at the starting position. The result is always the position in the source string even if the starting position is specified. The starting position defaults to 1.

The search argument (first parameter) must be either a character (or a graphic) data type. The search string (second parameter) must be the same data type as the search argument. The starting position of the search (third parameter), if specified, must be numeric with zero decimal positions.

The data type of the return value is unsigned integer. This built-in function can be used anywhere that an unsigned integer expression is valid.

Note that the search argument used is compared in its complete form with the string to be scanned. If you do wanted to ignore leading and trailing blanks, you could use %TRIM to remove all unwanted blanks as shown in the last example.

The %SCAN BiF is case-sensitive.

Scan for character string: %Scan (2 of 2)

IBM i

```

FIG625.RPGLE X
Line 1      Column 1      Replace          Browse
.....DName++++++ETDsFrom++To/L++IDc.Keywords+++++
000001      D Source      S           15A   Inz('Dr. Doolittle')
000002      D Pos         S           5U   0
000003
000004      /Free
000005
000006 >>1    Pos = %Scan('oo': Source); // Pos = 6
000007
000008 >>2    Pos = %Scan('D': Source : 2); // Pos = 5
000009
000010 >>3    Pos = %Scan('abc' : Source); // Pos = 0
000011
000012 >>4    Pos = %Scan('Dr.': Source : 2); // Pos = 0
000013
000014      *InLR = *on;
000015      /End-free

```

© Copyright IBM Corporation 2012

Figure 4-27. Scan for character string: %Scan (2 of 2)

AS067.0

Notes:

- Pos = 6** because 'oo' begins at position 6 in 'Dr. Doolittle'.
- Pos = 5** because the first 'D' found starting from position 2 is in position 5.
- Pos = 0** because 'abc' is not found in 'Dr. Doolittle'.
- Pos = 0** because 'Dr.' is not found in 'Dr. Doolittle', if the search starts at position 2.

Extract substring: %Subst

IBM i

```
%SUBST(string:start{:length})
```



The screenshot shows an IBM i editor window titled 'FIG626.RPGLE'. The code demonstrates the use of the %SUBST function to extract substrings from fields. It includes comments, assignments, and logic to handle blank positions and reverse names.

```

Line 15      Column 6      Replace          Browse
+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000001 D Comment      S           16A INZ('It''s a great day')
000002 D NameField     S           20A INZ('John Smith')
000003 D FirstName    S           10A INZ
000004 D ReverseNm   S           20A INZ
000005 D BlankPos     S           2P 0 INZ
000006
000007 /Free
000008 %Subst(Comment:8:5) = 'lousy'; // Comment = 'It's a lousy day'
000009
000010 BlankPos = %Scan(' ' : NameField); // BlankPos = 5
000011
000012 FirstName = %SUBST(NameField : 1 : BlankPos - 1); // FirstName = 'John'
000013
000014 ReverseNm = %TrimR(%Subst(NameField : BlankPos + 1))
000015     + ', '
000016     + %Subst(NameField : 1 : BlankPos - 1); // ReverseNm = 'Smith , John'
000017
000018 *InLr = *On;
000019 /End-free

```

© Copyright IBM Corporation 2012

Figure 4-28. Extract substring: %Subst

AS067.0

Notes:

The substring built-in function allows you to extract a substring from a field. The first parameter is the field upon which the operation is to be performed. The parameter after the field name, delimited by a colon, is the starting position of this field where the operation is to begin. The last parameter, which is optional, is the length of the string. The length parameter represents the length of the substring.

If it is not specified, the length is the length of the string less the start value plus one. For example, if the operation is to take place on a 15 byte string, starting in the fifth position, the length of the substring is assumed to be 11 characters.

Note that in the final Eval in this visual, the previously calculated value of **FirstName** field could have been used rather than substringing the **NameField** again. We could have written:

```
ReverseNm = %TrimR(%Subst (NameField : BlankPos + 1))
+ ', ' + FirstName
```

Replace string of characters: %Replace

IBM i

```
%REPLACE(replacement string: source string(:start position
{:source length to replace}))
```

```

FIG627.RPGLE
Line 1      Column 1      Replace          Browse
.....DName++++++ETDsFrom+++To/L++IDc.Keywords+++++=====
000001    D Greetings     S      50A  Inz('We wish you, &C, +
000002    D                               a Happy Birthday!')
000003    D FullName       S      30A  Inz('Mr. Smith')
000004    D Position        S      2P 0
000005
000006 /Free
000007>>1 Position = %scan('&C' : Greetings); // Position = 14
000008>>2 If Position > 0;
000009>>3   Greetings = %replace(%trim(FullName) :
000010           Greetings : Position : 2);
000011 // Greetings = 'We wish you, Mr. Smith, a Happy Birthday!
000012 EndIf;
000013
000014 *InLr = *on;
000015 /End-free

```

© Copyright IBM Corporation 2012

Figure 4-29. Replace string of characters: %Replace

AS067.0

Notes:

The **%replace** BiF replaces a specified number of characters in one string with characters from another. The parameters are:

1. Replacement string
2. Source string
3. Start position
4. Number of positions to replace

This example illustrates the function of **%replace**. We are using a generic greeting message to wish someone a Happy Birthday. The **%replace** BiF is used to replace the characters, with a person's name.

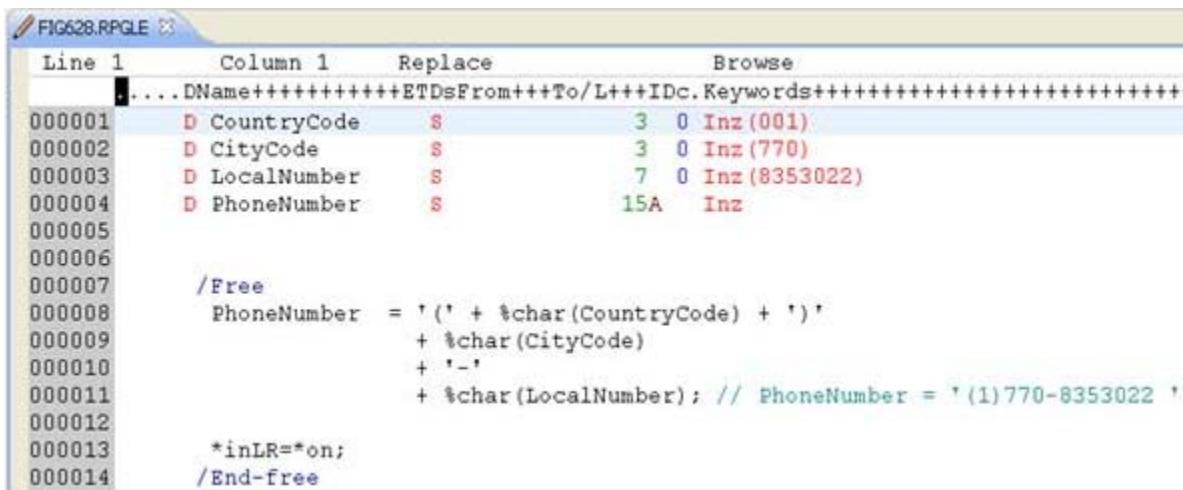
To do this, we should handle some other details in addition to the replacement:

1. Check to make sure that the string we expect to replace in the string is in the string and is **&C**. To do this, we use the **%scan** BiF to find the starting position of the **&C** substring.

2. If the starting position is greater than zero, we perform the replacement operation using the **%Replace** BiF.
3. We also use the **%trim** BiF to remove any leading and trailing blanks in the **FullName** string that will replace the **&C** characters.

Convert to character: %Char

IBM i



```

FIG628.RPGLE
Line 1      Column 1      Replace      Browse
000001      D CountryCode   S           3 0 Inz(001)
000002      D CityCode     S           3 0 Inz(770)
000003      D LocalNumber   S           7 0 Inz(8353022)
000004      D PhoneNumber   S           15A  Inz
000005
000006
000007      /Free
000008      PhoneNumber = '( + %char(CountryCode) + ')'
000009          + %char(CityCode)
000010          + '-'
000011          + %char(LocalNumber); // PhoneNumber = '(1)770-8353022 '
000012
000013      *inLR=*on;
000014      /End-free

```

© Copyright IBM Corporation 2012

Figure 4-30. Convert to character: %Char

AS067.0

Notes:

Remember that the Eval requires that all variables operated on the statement must be the same data type. In this example, we want to derive an *edited* character field based on three numeric fields.

Because **PhoneNumber** is a character field, all fields in the expression to the right of the **=** sign must be character and we use the **%Char** BiF to convert the numeric values to character.

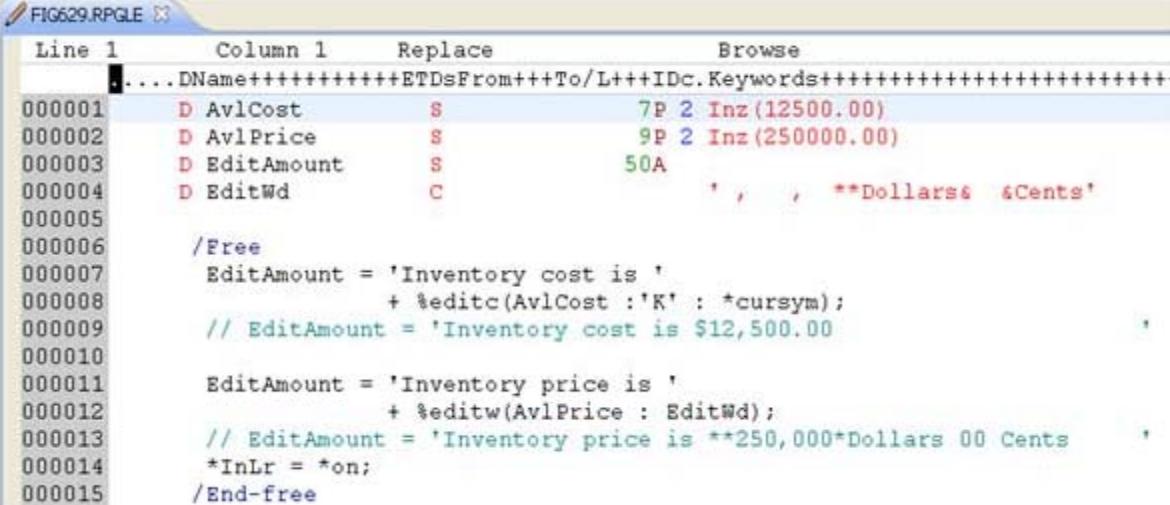
The **%char** BiF can operate on various data types, including graphic, UCS-2, numeric, date, time or timestamp data types. The converted value remains unchanged but is returned in a format that is compatible with character data.

Edit with %EditC / %EditW

IBM i

```
%EDITC(numeric : editcode {:: *ASTFILL | *CURSYM | currency-
symbol})
```

```
%EDITW(numeric : editword)
```



The screenshot shows the IBM i editor interface with a window titled 'FIG629.RPGLE'. The code in the editor is as follows:

```

Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
000001    D AvlCost      S          7P 2 Inz(12500.00)
000002    D AvlPrice     S          9P 2 Inz(250000.00)
000003    D EditAmount   S          50A
000004    D EditWd       C          ' , , **Dollars & Cents'
000005
000006    /Free
000007        EditAmount = 'Inventory cost is '
000008            + %editc(AvlCost :'K' : *cursym);
000009        // EditAmount = 'Inventory cost is $12,500.00'
000010
000011        EditAmount = 'Inventory price is '
000012            + %editw(AvlPrice : EditWd);
000013        // EditAmount = 'Inventory price is **250,000*Dollars 00 Cents
000014        *InLr = *on;
000015    /End-free

```

© Copyright IBM Corporation 2012

Figure 4-31. Edit with %EditC / %EditW

AS067.0

Notes:

%EditC and %EditW BiFs are very useful in the conversion of a numeric value to an edited character value.

%EditC edits a numeric field, producing a character value; it expects two (optionally three) parameters:

1. Numeric
2. Edit code
3. *cursym|*astfill|currency-symbol (optional)

This parameter can optionally provide asterisk fill, a floating system currency symbol, or a specified floating currency symbol.

%EditW edits a numeric field, producing a character value; it expects two parameters:

1. Numeric
2. Editword

The edit word can be supplied as a literal value or as a constant shown in the example.

We cover more of the available edit codes available in RPG IV in the next unit, *Printing from an RPG IV Program*.

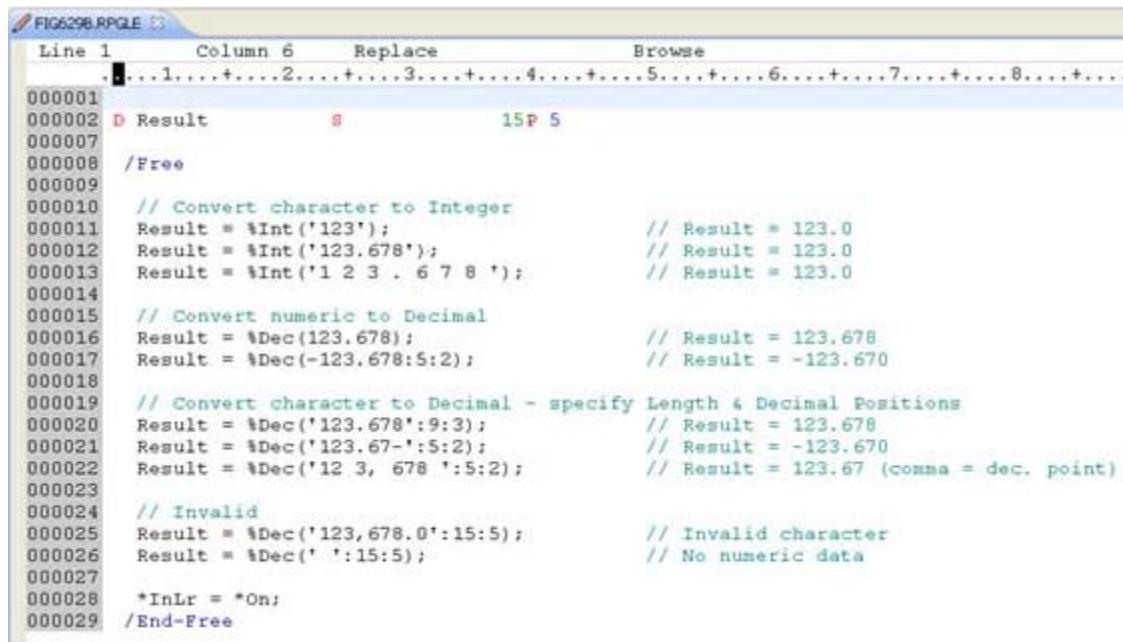
Reference:

IBM i Information Center: DDS Reference - Printer Files, Keyword Entries for Printer Files

Convert to numeric: %INT and %DEC

IBM i

%INT(expression)
 %DEC(expression:{precision:decimal places})



```

FIG629B.RPGLE
Line 1      Column 6      Replace      Browse
. ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....
000001      D Result      S          15P 5
000002
000003
000004
000005
000006
000007
000008
000009
000010 // Convert character to Integer
000011 Result = %Int('123');
000012 Result = %Int('123.678');
000013 Result = %Int('1 2 3 . 6 7 8 ');
000014
000015 // Convert numeric to Decimal
000016 Result = %Dec(123.678);           // Result = 123.678
000017 Result = %Dec(-123.678:5:2);     // Result = -123.670
000018
000019 // Convert character to Decimal - specify Length & Decimal Positions
000020 Result = %Dec('123.678':9:3);    // Result = 123.678
000021 Result = %Dec('123.67-':5:2);    // Result = -123.670
000022 Result = %Dec('12 3 , 678 ':5:2); // Result = 123.67 (comma = dec. point)
000023
000024 // Invalid
000025 Result = %Dec('123,678.0':15:5); // Invalid character
000026 Result = %Dec(' ':15:5);          // No numeric data
000027
000028 *InLr = *On;
000029 /End-Free
  
```

© Copyright IBM Corporation 2012

Figure 4-32. Convert to numeric: %INT and %DEC

AS067.0

Notes:

%INT converts the numeric or character expression to a signed integer. Any decimal digits are truncated.

%DEC converts the value of the numeric or character expression to decimal (packed) format with precision digits and decimal places. Precision and decimal places must be numeric literals, named constants that represent numeric literals, or built-in functions with a numeric value known at compile time.

The arguments precision and decimal places may be omitted if the type of expression is not float or character. If these arguments are omitted, the precision and decimal places are taken from the attributes of numeric expression.

In the examples above, blanks, a comma, a period, a minus (-) sign, and a plus (+) sign are permitted. A comma is interpreted as a decimal point.

Convert string: %XLATE

IBM i

```
%XLATE (from:to:string{:startpos})
```

```

FIG630.RPGLE X
Line 1      Column 1      Replace          Browse
.....DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
000001      D Phone        S               8   Inz('974 8088')
000002      D PhoneNbr     S               Like(Phone)
000003
000004      D Upper        C               'IVXL'
000005      D Lower        C               'ivxl'
000006
000007      D Atable       S               6   Inz('RPG iv')
000008      D Btable       S               Like(Atable)
000009
000010      /Free
000011      PhoneNbr = %Xlate(' ' : '-' : Phone);
000012      // PhoneNbr = '974-8088'
000013
000014      Btable = %Xlate(Lower : Upper : Atable);
000015
000016      // Btable = 'RPG IV '
000017      *inlr = *on;

```

© Copyright IBM Corporation 2012

Figure 4-33. Convert string: %XLATE

AS067.0

Notes:

%XLATE translates a **string** according to the values of **from**, **to**, and **startpos**.

1. The first parameter contains a list of characters that should be replaced.
2. The second parameter contains their replacements. For example, if the string contains the third character in from, every occurrence of that character is replaced with the third character in to.
3. The third parameter is the string to be translated. The fourth parameter is the starting position for translation. By default, translation starts at position 1.
4. The fourth parameter is a numeric variable or value (not floating point) with zero decimal positions for the starting position of the operation.

The first three parameters can be of type character, graphic, or UCS-2. All three must be the same data type. The value returned has the same type and length as **string**.

The first example in this visual shows the character " " being replaced by a hyphen. The second example shows the replacement of the lowercase characters, "iv", by the uppercase characters, "IV".

Precedence rules

IBM i

1. ()
2. Built-in functions
3. Negation (-,NOT)
4. ** Exponentiation
5. *,/ Multiplication and division
6. +,- Addition and subtraction
7. =,>=,>,<=,<,<> Comparison
8. AND
9. OR

Result=A + B * C ** (X * Y)

If A=3, B=4, C=5, X=1, Y=2, what will be the value of the Result?

© Copyright IBM Corporation 2012

Figure 4-34. Precedence rules

AS067.0

Notes:

When using expressions, precedence rules are used to determine the order of evaluation of the parts of the expression. These precedence rules are similar to the traditional rules in arithmetic expression elsewhere.

The most important point to note here is that parentheses have the highest precedence and, therefore, can be used to ensure correct results, as well as to make the expression more readable.

Precedence rules: Using parentheses

IBM i

() are highest in precedence; use them to:

- Break up complex expressions
- Ensure accurate results

```

FIG632A.RPGLE
Line 1      Column 1      Replace      Browse
...../FREE.....
000001      /Free
000002
000003      // 1. Unit price = ItemPrice - ItemDiscount
000004      // 2. Calculate the total price of quantity ordered (OrdQty)
000005
000006      TotalPrice = OrdQty * ItemPrice - ItemDiscount;
000007
000008      // 3. Above calculation is wrong. Should be coded:
000009
000010      TotalPrice = OrdQty * (ItemPrice - ItemDiscount);
000011      /End-free

FIG632B.RPGLE
Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
000001      D Exit      S      N
000002      D Indicator1  S      N
000003      D Count     S      5 0
000004      D Max       S      5 0 Inz(99999)
000005      /Free
000006
000007      If Exit or Count > Max AND Indicator1; // OK but ambiguous
000008
000009      If Exit or ((Count > Max) AND Indicator1); // Much better
000010
000011      /End-free

```

© Copyright IBM Corporation 2012

Figure 4-35. Precedence rules: Using parentheses

AS067.0

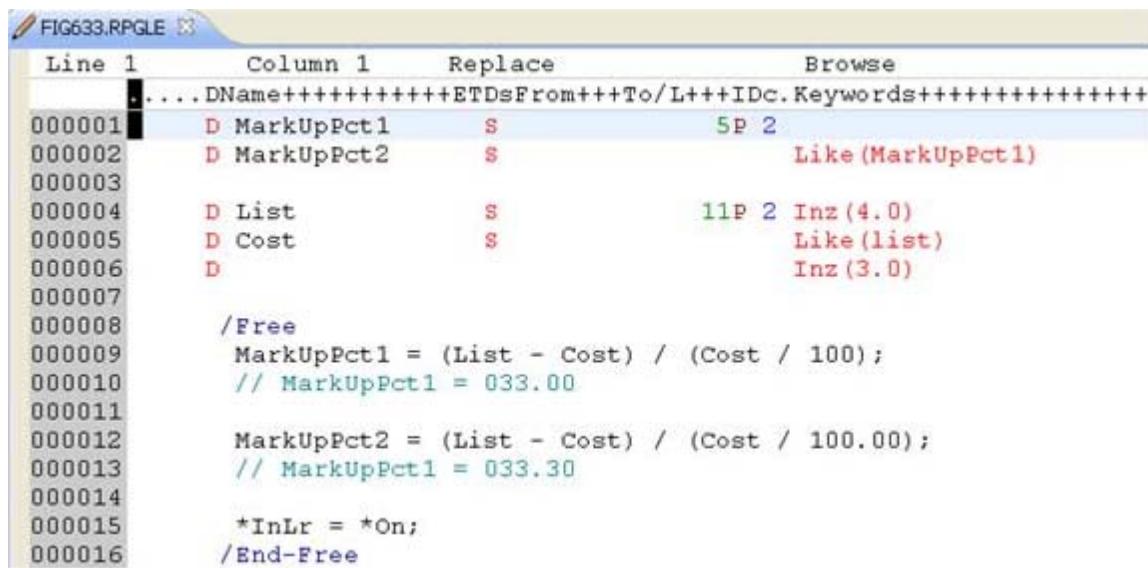
Notes:

It is *strongly* recommended that you use parentheses frequently and freely, not only to ensure correct calculation results, but also to make the code easier to read and understand, without requiring the reader to go through the precedence rules to understand the expression.

The use of parentheses makes the code less ambiguous to the programmer, and therefore, maintenance is not only easier to perform, but more reliable.

Precision: Expressions

IBM i



```

FIG633.RPGLE
Line 1      Column 1      Replace      Browse
.....DName++++++ETDsFrom++To/L+++IDc.Keywords+++++
000001      D MarkUpPct1    S           5P 2
000002      D MarkUpPct2    S           Like(MarkUpPct1)
000003
000004      D List          S           11P 2 Inz(4.0)
000005      D Cost          S           Like(list)
000006      D                   S           Inz(3.0)
000007
000008      /Free
000009      MarkUpPct1 = (List - Cost) / (Cost / 100);
000010      // MarkUpPct1 = 033.00
000011
000012      MarkUpPct2 = (List - Cost) / (Cost / 100.00);
000013      // MarkUpPct1 = 033.30
000014
000015      *InLr = *On;
000016      /End-Free

```

Expected value of mark up percentage is 33.33

© Copyright IBM Corporation 2012

Figure 4-36. Precision: Expressions

AS067.0

Notes:

Developers have always been responsible for ensuring accurate results of calculations in their programs. When coding expressions that are complex, it is important to test your code carefully to ensure that the calculations produce the correct results.

Precision is important when you have an expression that has intermediate results. It is these intermediate results that can impact the accuracy of your final result, particularly when you have multiple division operations and/or division involving integer variables or literals (zero decimals).

Precision rules state that:

- Intermediate integer or unsigned results have a precision of 10 digits.
- Precision rules do not affect floating point data, which has the precision of double precision floating point data.

The rules of precision as they apply to the intermediate results of expressions are described in the *ILE RPG Language Reference*. You should become familiar with them.

Review the pages containing the precision rules and formulae in the RPG Reference manual.

Note that the TRUNCNBR option of the CRTBNDRPG or the CRTRPGMOD commands *does not apply* to calculations performed within expressions. If overflow occurs during an operation involving expressions, a run time message is issued.

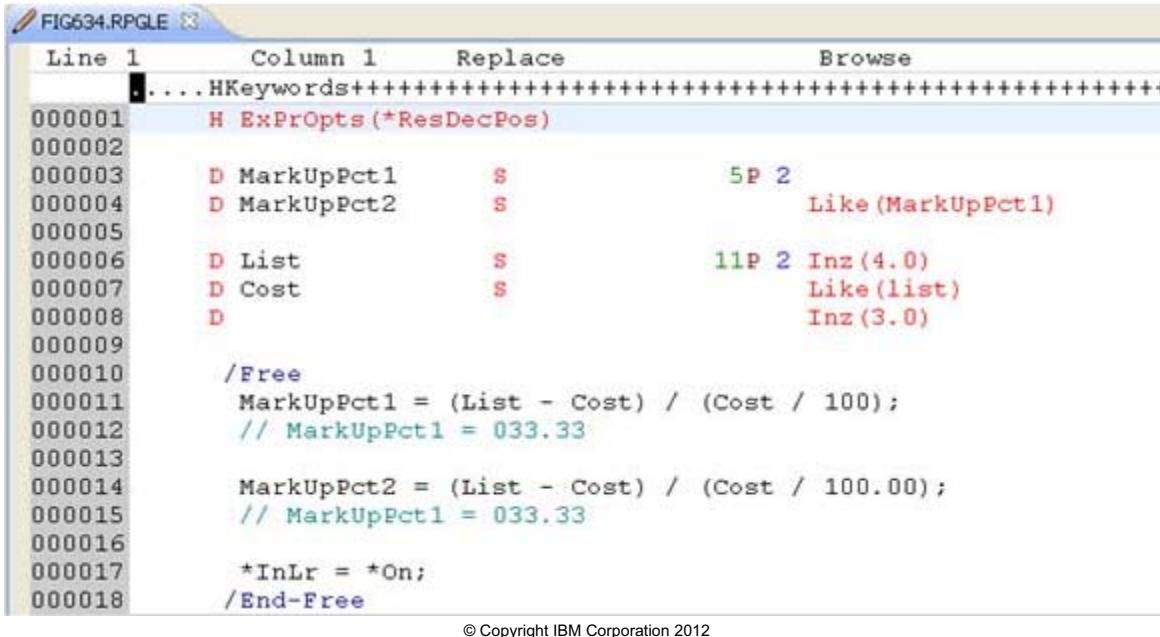
Reference:

ILE RPG Language Reference, Chapter 20, Expressions

Specifying precision rules

IBM i

- Resulting decimal positions:
 - RESDECPOS on H-specification applies to *all* Evals
 - R-extender without *RESDECPOS applies to single Eval



The screenshot shows an IBM i RPGLE editor window titled 'FIG634.RPGLE'. The code is as follows:

```

Line 1      Column 1      Replace      Browse
000001      . . . . HKeywords+++++
000002          H ExprOpts(*ResDecPos)
000003          D MarkUpPct1      S      5P 2
000004          D MarkUpPct2      S      Like(MarkUpPct1)
000005
000006          D List          S      11P 2 Inz(4.0)
000007          D Cost          S      Like(list)
000008          D                  Inz(3.0)
000009
000010          /Free
000011          MarkUpPct1 = (List - Cost) / (Cost / 100);
000012          // MarkUpPct1 = 033.33
000013
000014          MarkUpPct2 = (List - Cost) / (Cost / 100.00);
000015          // MarkUpPct1 = 033.33
000016
000017          *InLr = *On;
000018          /End-Free

```

© Copyright IBM Corporation 2012

Figure 4-37. Specifying precision rules

AS067.0

Notes:

The simplest way to control the precision of the results of expressions is to base them on the number of decimal positions of the result field. Result Decimal Position Precision Rules mean that the decimal accuracy you desire in the result field of the expression determines the decimal accuracy that the system must carry for Intermediate Results or Subexpressions.

The **EXPROPTS** (expression options) keyword specifies the type of precision rules to be used for an entire program. If not specified, you experience results that might not make sense.

If **EXPROPTS** is specified, with ***RESDECPOS**, the result decimal position precision rules apply and force intermediate results in expressions to have no fewer decimal positions than the result.

ResDecPos

At this point in your RPG IV training, you should always code the **ExprOpts(*resdecpos)** keyword and parameter on the H-spec.

Machine exercise: Data manipulation

IBM i



© Copyright IBM Corporation 2012

Figure 4-38. Machine exercise: Data manipulation

AS067.0

Notes:

Perform the data manipulation machine exercise.

Checkpoint

IBM i

- Given the following values for the variables, what will be the value of X after the EVAL statement?

A = 1, B = 2, C = 3

EVAL X = A * B + C ** B - 1

- Given the same initial values for the variables A, B and C, what will be the value of X after this EVAL statement?

EVAL X = A * B + C ** (B - 1)

- Given the same initial values for the variables A, B and C, and given that *IN03 is *ON, what will be the value of *INLR after this EVAL statement?

EVAL *INLR = *IN03 AND B > A OR C > A + B

© Copyright IBM Corporation 2012

Figure 4-39. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Code calculation specifications using free-form operation codes, and expressions for:
 - Assignment of numeric, character, and logical values
 - Arithmetic operations
 - Logic operations
 - String handling operations
- Code built-in functions in expressions
- Determine the size of a numeric result field
- Use H-specification keywords to manage the numeric result of an expression

© Copyright IBM Corporation 2012

Figure 4-40. Unit summary

AS067.0

Notes:

Unit 5. Printing from an RPG IV program

What this unit is about

This unit describes how to use externally described DDS to produce printed output from an RPG IV program. First, this unit briefly reviews the creation of printer DDS and discuss how it can be used within an RPG IV program.

This unit reviews some concepts taught in OL49/OL490, a prerequisite for this class. What is new in this unit is the actual design of a report, the entering of the DDS to describe the printer file (PRTF), creating a printer file, and writing to the printer file from your RPG IV program.

What you should be able to do

After completing this unit, you should be able to:

- Use DDS to define the layout of a report
- Create a printer file (PRTF)
- Write an RPG IV program that prints a report using a printer file defined in DDS

How you will check your progress

- Checkpoint questions
- Machine exercise:
 - Given a problem statement, the students create a printer file to format the data into detailed line items and a grand total. Then, the students code the program that prints the report using the externally described printer file. During this exercise, the student:
 - Creates a printer file with DDS incorporating:
 - Database fields
 - Programmer described fields
 - **EDTCDE, SKIP, SPACE, PAGE, DATE, TIME** keywords
 - Writes RPG IV code to use an externally described printer file and handle page overflow

Unit objectives

IBM i

After completing this unit, you should be able to:

- Use DDS to define the layout of a report
- Create a printer file (PRTF)
- Write an RPG IV program that prints a report using a printer file defined in DDS

© Copyright IBM Corporation 2012

Figure 5-1. Unit objectives

AS067.0

Notes:

Printing process

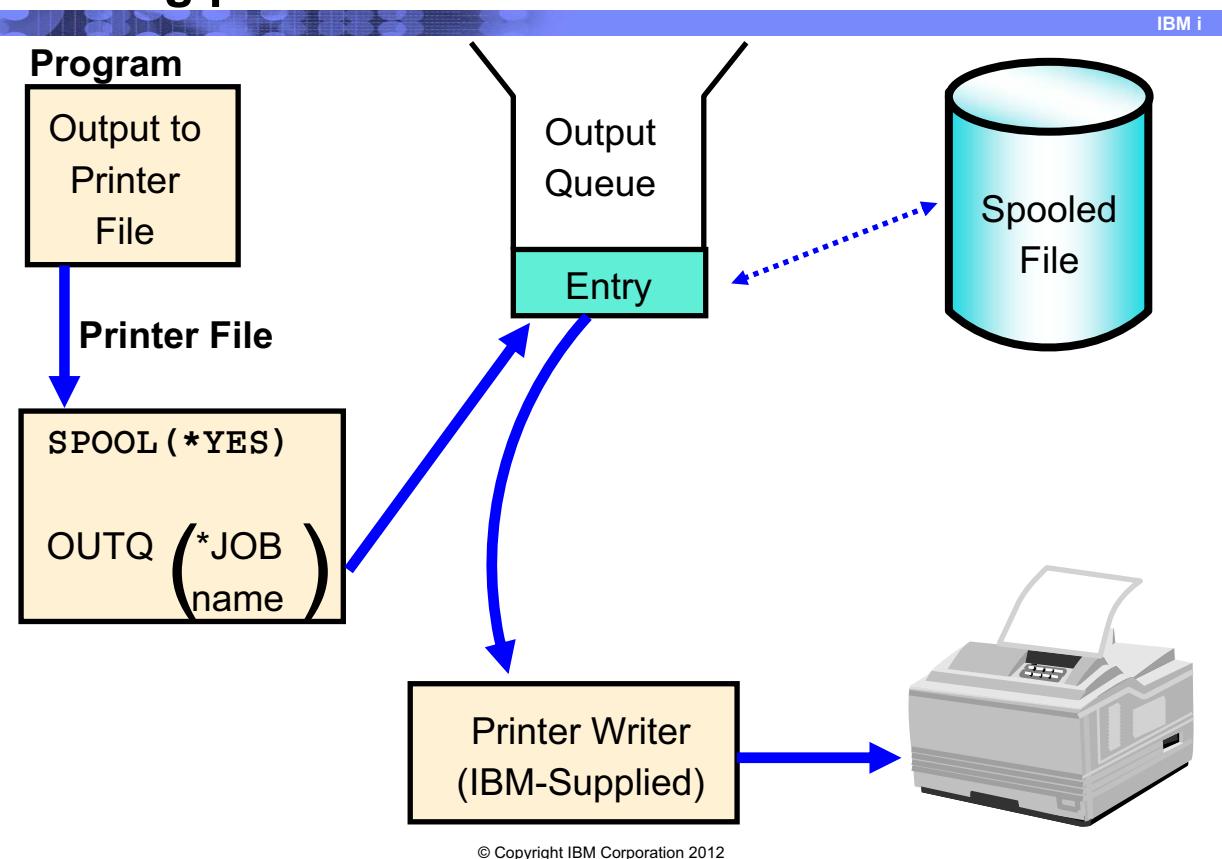


Figure 5-2. Printing process

AS067.0

Notes:

This is the process followed for printing:

1. RPG program plus printer file produce report
2. Printer file specifies *SPOOL=*YES
3. Spool file created
4. Spool file entry placed in OUTQ
5. Print writer uses OUTQ entry to move spool file to printer device
6. Physical printer produces report

When a user runs a program, its associated printer file is accessed, a spooled file member is generated and placed in the system spool file (in library QSPL). An entry is placed in an output queue that points to the spool file member. Now, we discuss the relationship between printer files, spooled files, and output queues.

Printer files

Printers attached to the i system are supported by the operating system through device descriptions. Printer files describe how the system operates on the data when the report is created by your application and eventually sent to a printer.

Every request for printing is handled by a printer file. You can create your own printer files using the create printer file (CRTPRTF) command or you can use system-provided printer files.

Printer files contain many parameters that tell the system how the output should be formatted, what font to use for the printed output, whether to print on both sides of the page, and more. The parameters that control how your output is handled and where it goes are:

- Spool the data (**SPOOL**)
- Device (**DEV**)
- Spooled output queue (**OUTQ**)

Simultaneous peripheral operations online

Almost all application programs that generate printed output make use of the spooling support provided with the i system. Whether spooling support is requested is determined by specifying **SPOOL=***YES or **SPOOL=***NO on the **SPOOL** parameter of a printer file command (CRTPRTF).

By creating and using your own printer file instead of using a system-supplied printer file, you can specify the printing control instructions that your application program receives.

You can use the create printer file (CRTPRTF) command to create your own printer file.

Output queues

Output queues are objects, defined to the system, that contain queued entries that point to spooled files waiting to be printed. Output queues are created by the user or by the system.

Printing elements

IBM i

- Spooled file
- Output queue
- Printer writer
- Print devices
- Printer files
- Job description
- Workstation description
- User profile
- System values

© Copyright IBM Corporation 2012

Figure 5-3. Printing elements

AS067.0

Notes:

ElementDefinition

Spooled file: This is a file that holds output data waiting to be printed. The spool file is created by IBM i (OS/400), and, there is usually only one such file in QSPL library. It is the spool file MEMBER that is added when a printout is generated from a printer file. A spooled file member is created by an i (OS/400) system program, an application program or the Print key being pressed.

Output queue: An output queue is an i object that contains entries pointing to spooled files waiting to be printed. Output queues can receive spooled files from more than one application program and from more than one user.

Printer writer: This is a function of the operating system that sends the spooled file from an output queue to a printer. In most cases, the application program sends the spooled file to an output queue first. Then the printer writer program sends it to a printer.

Print devices: Printer devices are the physical printers that can be attached to the i system. Print devices (printers) should not be confused with the printer writer program or printer files.

Printer files: Printer files describe how the system is to operate on data that passes between your program and a printer. A printer file has many parameters. The spooling parameter (**SPOOL**) determines if your output goes to an output queue or directly to a printer. The device (**DEV**) parameter is the name of the printer your output is printed on. The output queue (**OUTQ**) parameter is the name of the output queue your spooled files are sent to.

Job description: A job description is a system object that defines how a job is to be processed.

A job description has many parameters. Printer device (**PRTDEV**) and output queue (**OUTQ**) are the two parameters that help determine where your output will go.

Workstation description: The workstation description contains information collected from the device description for the display. Two of the device description parameters, printer device (**PRTDEV**) and output queue (**OUTQ**), help determine where your output will go.

User profile: This object with a unique name contains the user's password, the list of special authorities assigned to a user and the objects the user owns. A user profile has many parameters. Printer device (**PRTDEV**) and output queue (**OUTQ**) are the two parameters that help determine where your output will go.

System values: System values control information for the operation of certain parts of the system. System administrators maintain system values to define the overall system working environment. The system value most important to printing is the default system printer.

Elements that control printing

IBM i

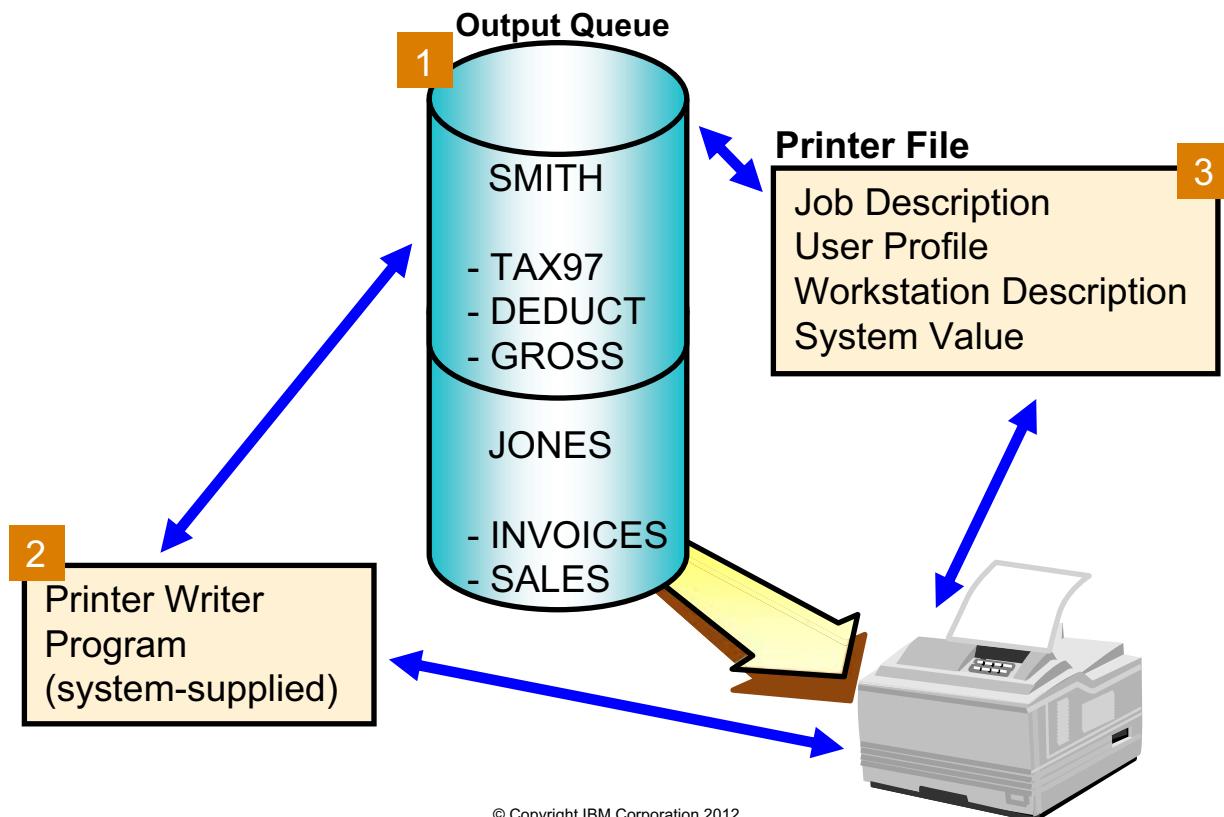


Figure 5-4. Elements that control printing

AS067.0

Notes:

Moving your printed output from your program to a specific printer device involves many different elements.

This visual illustrates how these elements that control printing are interrelated. To produce printed output, the attributes that are common to all the elements (output queues and printer devices) must be correctly matched.

Spoooled files and output queues are identified by 1 in the visual. Your program writes report (printed) output as spool files to an OUTQ.

Elements that control or direct the printing activity are:

- The printer writer program is identified by 2 in the visual. The print writer is the software that starts or stops the systems interface to a specific printer device. When the printer is started, spool files may be moved from the OUTQ to the print device.
- Printer files, job descriptions, user profiles, workstation descriptions, and system values are identified by 3 in the visual. The objects control how the output is printed, what OUTQ is used and so on.

For those who are unfamiliar with SPOOLing, here are a few benefits of using SPOOLing:

- Releases the application from the (slow) printer device
- Drives the physical printing by a separate batch job (the spool writer)
- Allows better utilization of resources
- Allows the same device to service multiple users through the output queue

Which OUTQ?

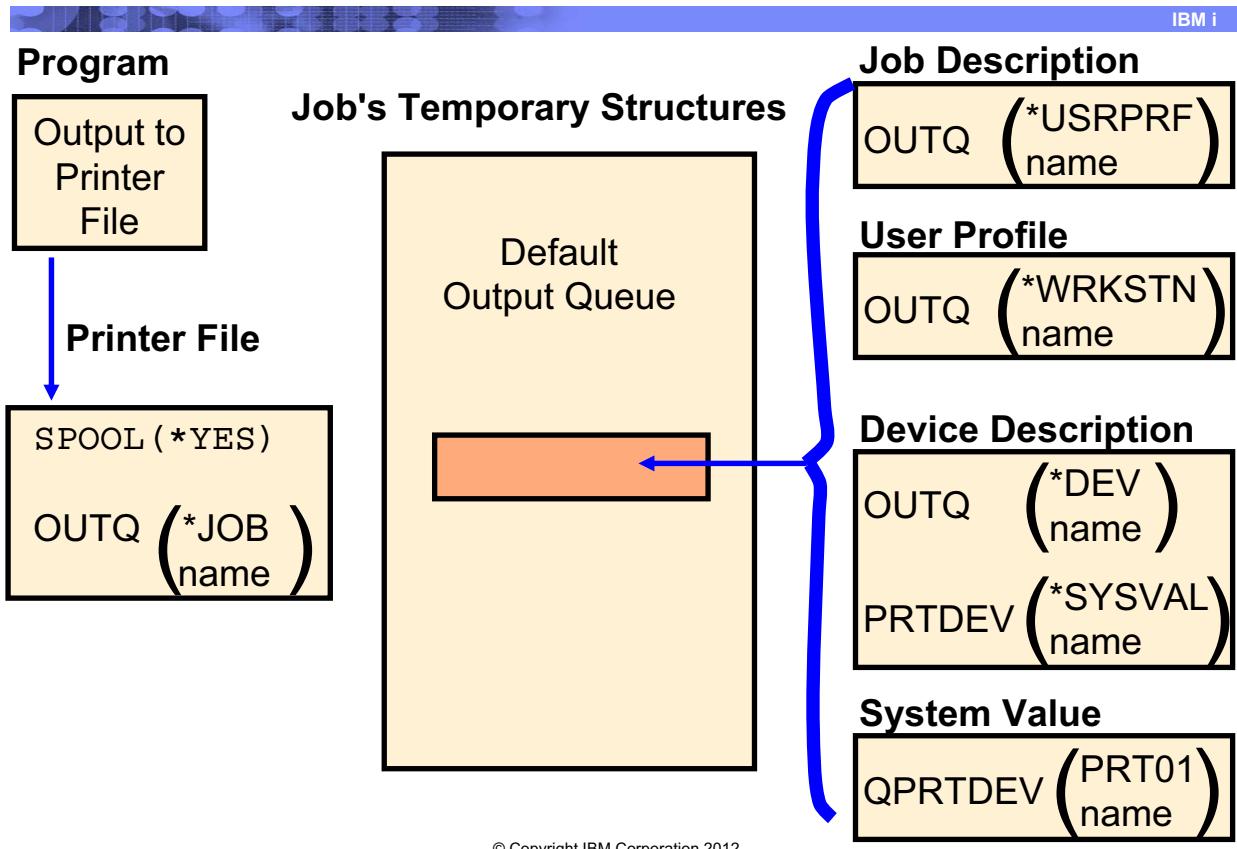


Figure 5-5. Which OUTQ?

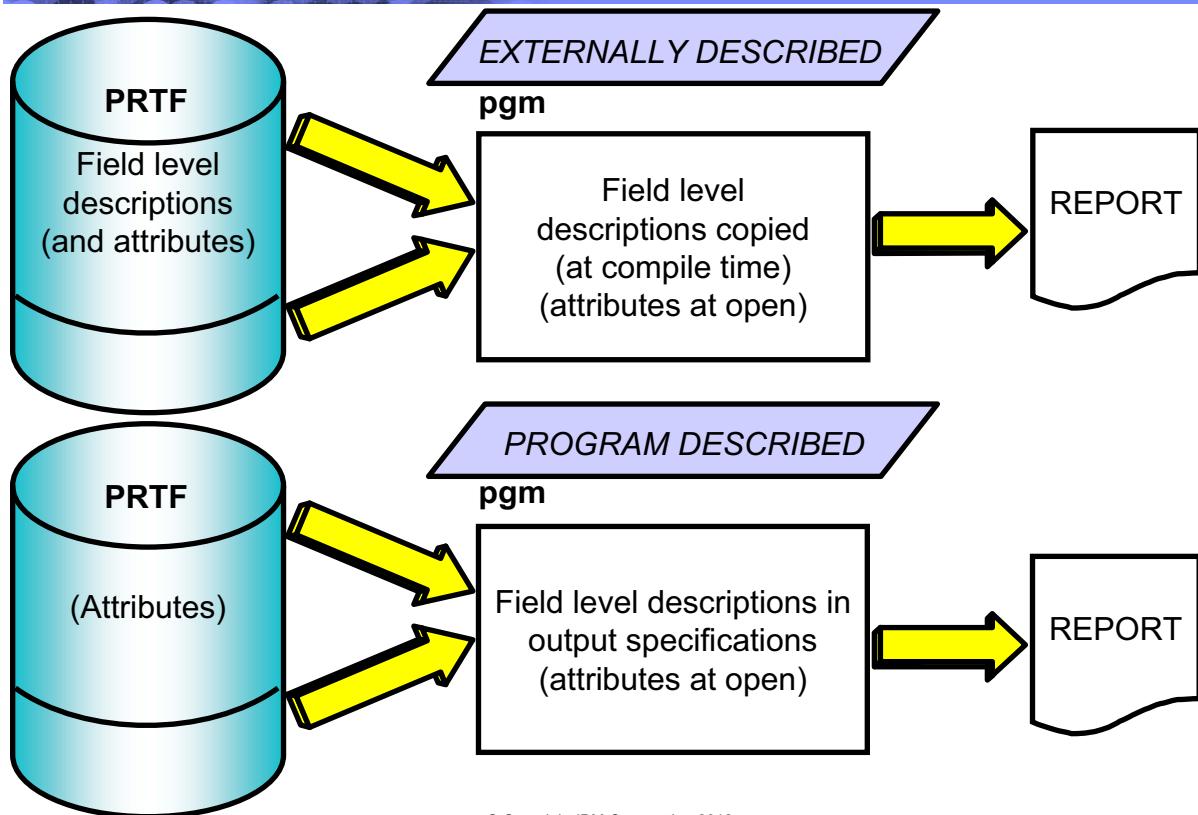
AS067.0

Notes:

The default output queue for a job is determined at job start. If an OUTQ is specified explicitly at any levels, then it is the OUTQ to be used.

Using a printer file

IBM i



© Copyright IBM Corporation 2012

Figure 5-6. Using a printer file

AS067.0

Notes:

A **printer file** describes how the system is to operate on data as it passes between your application program and a printer. A printer file must be specified in your RPG IV program. On other systems, you might code a physical printer device rather than a printer file. On the i, the printer file is the interface between your RPG IV program and the physical printer devices.

In its simplest form, a printer file is a definition of basic formatting attributes that you want to use in your report. In many cases, you simply use the i system-supplied printer files, QSYSPRT or QPRINT. If you want to define your report formatting attributes specifically, you might create a printer device file of your own or, more likely, use a generic printer file created in your enterprise to handle a family of similar printer devices.

The printer file is the filename for the F-spec that defines your printer output. In your program, the actual definition of the report format can be accomplished in one of two ways:

1. Program-described printer files

Program-described printer files rely on the high-level language program to define records and fields to be printed.

2. Externally described printer files

Externally described printer files use data description specification (DDS) rather than the high-level language to define records and field to be printed. You can specify the syntax and attributes of the report together in an externally described printer file. In this case, field description details are copied into the program from the printer file at program compile time. If you use DDS, you specify the name of your DDS source file in the **SRCFILE** parameter of the Create Printer File (CRTPRTF) command. DDS gives the application programmer better control over how the printed output is formatted and printed.

Here is an example that shows you a few of the parameters included in a printer file for one of the PRTF commands:

Change Printer File (CHGPRTF)

Type choices, press Enter.

```
File . . . . . > APPDSC01S      Name, generic*, *ALL
Library . . . . . > AS06V7LIB Name, *LIBL, *ALL, *ALL
Device:
  Printer . . . . . *JOB          Name, *SAME, *JOB, *SYS
  Printer device type . . . . . *SCS          *SAME, *SCS, *IPDS, *LI
Page size:
  Length--lines per page . . . . 66           .001-255.000, *SAME
  Width--positions per line . . . 132          .001-378.000, *SAME
  Measurement method . . . . . *ROWCOL        *SAME, *ROWCOL, *UOM
  Lines per inch . . . . . . . . 6             *SAME, 6, 3, 4, 7.5, 7,
  Characters per inch . . . . . . 10            *SAME, 10, 5, 12, 13.3,
  Overflow line number . . . . . . 60            1-255, *SAME
  Record format level check . . . *YES         *SAME, *YES, *NO
Text 'description' . . . . . > 'Lost discount summary report'
:
:
:
:
```

Whenever your program outputs a printed report, a printer file is referenced that describes the characteristics of the format of your output. Your F-spec must reference either an IBM-supplied printer file (like QSYSPRT or QPRINT) or a printer file that you or someone else in your enterprise has created. When using the IBM-supplied printer files, you must specify the syntax of the report in output specs.

Following is a list of the IBM-supplied printer files:

QSYSPRT: A program-described printer file in library QSYS.

QPRINT: Default spooled output printer file for normal print.

QPRINTS: Default spooled output printer file for special forms.

QPRINT2: Default spooled output printer file for two copy output.

QPSPLPRT: Default spooled output printer file for the spooling subsystem.

Reference:

Printer Device Programming, Chapter 2, Printer File Support

Externally described printer files

IBM i

- Maintenance can be simpler
- Program-managed printer control
- More likely to use standardized naming conventions

© Copyright IBM Corporation 2012

Figure 5-7. Externally described printer files

AS067.0

Notes:

RPG IV programs include less code (no output specifications for reports) if externally described printer files are used. Using externally described printer files makes your code more modular and, therefore, it can be easier to maintain. Also, if report formats must be altered, maintain the printer file DDS rather than the program that produces the specific report. Finally, if more than one program uses the same printer file, there is no need to duplicate coding.

With externally described printer files, you control printed output from your calculation specs.

Printer file characteristics

IBM i

```

FILE ..... File-level keywords
          apply to all Fields in
          all Formats

---
| FORMAT ..... Format-level keywords
|           apply to all Fields in
|           Format
|
| FIELD
| FIELD
| FIELD
---
---
| FORMAT
|
| FIELD ..... Field-level keywords
|
|   - constant (for example, Headings)
|   FIELD - variable (from program - for example, CustNo)
|   - system (for example, System date)
---
---
| FORMAT
|
|
---
```

© Copyright IBM Corporation 2012

Figure 5-8. Printer file characteristics

AS067.0

Notes:

- Smallest item to print is FIELD.
- Field types: Constant, variable, and system.
- Fields grouped together into a layout of FORMAT.
- Format can be a single line or entire page.
- Printer file can contain many formats (max 1024).
- Keywords specify additional print options:
 - UNDERLINE
 - HIGHLIGHT
 - FONT change
- Keywords defined at field, format, and file levels.

Printer file DDS

IBM i

FIG79.PRTF

Line 1	Column 1	Replace	Browse
000001	A* File Level		
000002	A		REF (VENDOR_PP)
000003	A* Record Level		
000004	A R VNADD_FMT		
000005	A		SKIPB (001)
000006	A		TEXT ('VENDOR ADDRESS FORMAT')
000007	A		FONT (222)
000008	A* Field Level		
000009	A		55 'VENDOR ADDRESS' UNDERLINE
000010	A		100DATE EDTCDE (Y)
000011	A		110TIME
000012	A		125 PAGNBR EDTCDE (J)
000013	A		SPACEA (1)
000014	A		
000015	A		2 'VENDOR:'
000016	A VNDNBR R		14EDTCDE (Z)
000017	A VNDNAME R		30SPACEA (1)
000018	A VNDCITY R		30SPACEA (1)

© Copyright IBM Corporation 2012

Figure 5-9. Printer file DDS

AS067.0

Notes:

File-level keywords are entered before the record format name. The file level is terminated when the record format name is processed.

Record-level keywords start on the same line as the record format name. The record level is terminated by the first field or constant is named.

Field-level keywords start on the same line as the field with which they are associated and are terminated when the next field or constant is specified.

Keywords and parameter values

- Code all DDS entries in uppercase except character literals enclosed in apostrophes.
- Code keywords on the same (or subsequent) line as the entry with which they are associated.
- Separate multiple keywords with at least one blank. Parameter values for keywords must be enclosed in parentheses following the keyword.

- Use apostrophes to enclose character values. Numeric values appear without apostrophes.
- Use a minus (-) sign or a plus (+) sign as a continuation character when a keyword and its value does not fit on a single line. The sign must be the last non-blank character in the functions field.

Reference:

IBM i Information Center: DDS - Keywords for Printer Files

Printer DDS keywords

IBM i

ALIAS	●	HIGHLIGHT	UNDERLINE
CHRID	● ●	LPI	
● CHRSIZ		PAGNBR	
● CPI	● ● ●	REF	
DATE		REFFLD	
● DFT	● ● ● ● ●	SKIPA	
● ● DRAWER	● ● ● ● ●	SKIPB	
EDTCDE	●	SPACEA	
EDTWRD	●	SPACEB	
● FONT	●	TEXT	

blank - Field level

● ● ● - File level

● - Field and record level

● ● ● ● - File and record level

● ● - Record level

● ● ● ● ● - File, field and record level

© Copyright IBM Corporation 2012

Figure 5-10. Printer DDS keywords

AS067.0

Notes:

Many of the DDS keywords are self-explanatory. We do not explain all of them in detail in class as you need to use the IBM Information Center *DDS Reference* until they become familiar to you.

PRTF DDS

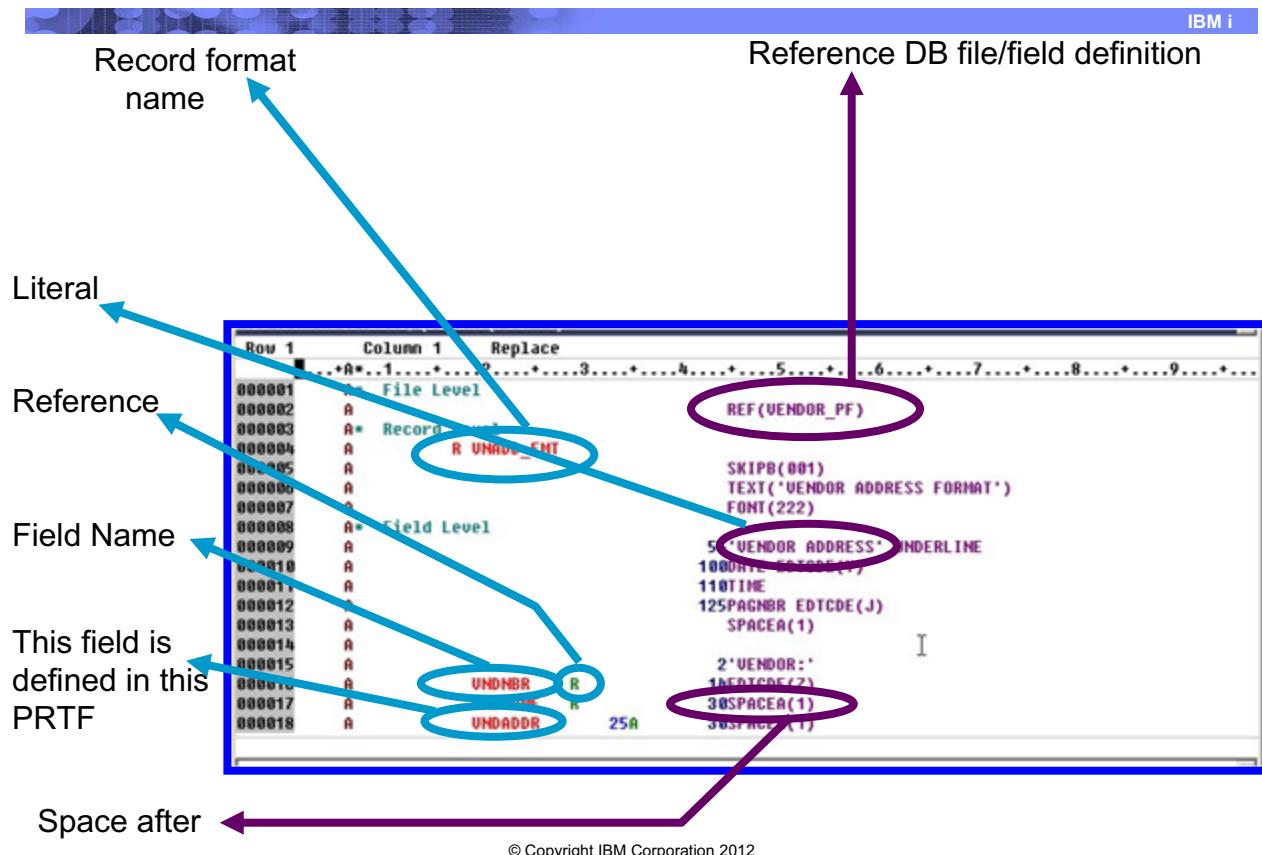


Figure 5-11. PRTF DDS

AS067.0

Notes:

This visual introduces the format of DDS that you use to define a report format (print file). More examples follow but for now, you should be aware that:

- Conditioning (Pos 7 - 16) indicators 01 to 99 may be used to determine record or field output. Use AND / OR for more than three indicators.
- Type of Name (Pos 17) R indicates name of a record format, blank indicates a field name.
- Name (Pos 19 - 28) used to name a record format or a field.
- Reference (Pos 29) R indicates a copy of field attributes from a previous definition is requested (works in conjunction with REF and REFFLD keywords), blank requires further field definition (that is, length, data type and decimal positions, if needed).
- Length (Pos 30 - 34) designates length of a named field that is not referenced.
- Data Type (Pos 35) specifies the data type associated with the field:
 - S: Zoned decimal

- A: Character
- F: Floating point
- Decimal Positions (Pos 36 - 37) define decimal placement within zoned decimal field.
- Usage (Pos 38) defines field as output field, blank or O.
- Location (Pos 39 - 44) positions the beginning of a defined field.
 - Line (pos 39 - 41) specifies the line on the page in which a field begins.
 - Position (pos 42 - 44) start of the field.
- Keywords (Pos 45 - 80) literal constants or keywords are specified.

Output report: Example

...+....1....+....2....+....3....+....4....+....5....+....6....+....7..						IBM i
5/02/00 Purchase Items Listing						
Page: 1						
<hr/>						
Item No Description Qty On Hand Qty On Order Tot Avail						
20001	Telephone, one line	10	6	16		
20002	Telephone, two line	5	12	17		
20003	Speaker Telephone	6	8	<u>14</u>		
20004	Telephone Extension Cord	25	10	35		
:						
20026	Three Ring Binders	10	108	118		
20027	Three hole punch	149	2	151		
<hr/>						
5/02/00	Purchase Items Listing					
Page: 2						
<hr/>						
Item No Description Qty On Hand Qty On Order Tot Avai						
20028	Desk picture frame 5 X 7	47	120	16		
:						
20049	3 hole lined paper	10	4	<u>14</u>		
20050	Heavy duty stapler	5	0	<u>5</u>		
*** End of Listing ***						

© Copyright IBM Corporation 2012

Figure 5-12. Output report: Example

AS067.0

Notes:

This is a Purchase Items Listing sample report. It lists the item number, description, quantity on hand, quantity on order, and total available for various item records from the item data file. If the total quantity available is below 15, the total available field is underlined. Let's look at the DDS that defines the printer file used for this example.

PRTF: DDS example

IBM i

```
*ITEMLIST.PRTF
Line 1 Close column 1 Replace 13 changes
...+A.....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000100 ** ITEMLIST - member type PRTF in QDDSSRC - List of Purchase Items
000200 A REF(ITEM_PF)
000300 A INDARA [1]
000500 ** Page heading
000600 A R HEADING SKIPB(6) SPACEA(1) [2]
000700 A 5DATE EDTCDE(Y) [3]
000800 A 30'Purchase Items Listing'
000900 A UNDERLINE [4]
001000 A 65'Page:'
001100 A +1PAGNBR EDTCDE(Z) [5]
001200 A 5'Item No' SPACEB(2)
001300 A 13'Description'
001400 A 39'Qty On Hand'
001500 A 52'Qty On Order'
001600 A 66'Tot Avail'
001800 ** Item detail
001900 A R DETAIL SPACEB(1)
002000 A ITMNBR R 5
002100 A ITMDESCR R 13
002200 A ITMQTYOH R 41EDTCDE(J)
002300 A ITMQTYOO R 55EDTCDE(J)
002400 A TOTQTYAVL 8 0 65EDTCDE(J)
002500 A 43 UNDERLINE [8]
002700 ** Report ending
002800 A R FOOTING SPACEB(2)
002900 A 30**** End of Listing ****
```

© Copyright IBM Corporation 2012

Figure 5-13. PRTF: DDS example

AS067.0

Notes:

Notice that there are separate record formats for the data that is heading-related, record (item detail)-related and footing-related. We could add additional records as needed for subtotals and totals.

For now, let us focus on several features of this example:

1. We reference the definition of fields already defined in the `ITEM_PF` file in the database. Also notice the `INDARA` keyword. We use it with our PRTFS.
2. For the heading record, we specify skipping six lines prior to printing the line followed by one blank line afterwards.
3. A number of special keywords are available, such as `DATE`, `TIME`, and `PAGNBR`. These variables are automatically defined in DDS and do not have to be referenced in your RPG IV program. The system automatically supplies date and time values, and, in the case of page numbering, automatically increments its value.
4. Keywords may be used to make a report more readable by specifying `underline`, for example.

5. Relative positioning of data is supported. In this case the field will be started one space after the previous field.
6. Editing is supported in DDS in two ways:
 - EDTCDE supports many edit codes that are explained in the *DDS Reference Manual*. Some of the functions performed by edit codes are:
 - Leading zeros are suppressed.
 - The field can be punctuated with commas and periods to show decimal position and to group digits by threes.
 - Negative values can be displayed with a minus sign or CR to the right.
 - Zero values can be displayed as zero or blanks.
 - Asterisks can be displayed to the left of significant digits to provide asterisk protection.
 - A currency symbol (corresponding to the system value QCURSYM) can be displayed immediately to the left of the significant digit that is farthest to the left (called floating-currency symbol). For fixed-currency symbols, use the `EDTWRD` keyword.
 - `EDTWRD` allows you to create your own edit mask in DDS.
1. The field **TOTQTYAVL** is defined externally in the printer file and is available to your RPG IV program without any further definition.
2. We have the UNDERLINE attribute controlled by an indicator, 43. Because we write RPG IV programs with named indicators only, you must code the `INDARA` keyword at the file level. This keyword enables us to map named indicators to numbered indicators.

Finally, the member type for a DDS described printer file is PRTF. When compiling the member, you may use option **14** in PDM, or you could execute the CRTPRTF command as follows on the next page:

Create Printer File (CRTPRTF)

Type choices, press Enter.

```

File . . . . . > ITEMLIST      Name
  Library . . . . . > AS06V7LIB  Name, *CURLIB
Source file . . . . . > QDDSSRC   Name, *NONE
  Library . . . . . > AS06V7LIB  Name, *LIBL, *CURLIB
Source member . . . . . > ITEMLIST  Name, *FILE
Generation severity level . . . . 20    0-30
Flagging severity level . . . . . 0     0-30
Device:
  Printer . . . . . *JOB        Name, *JOB, *SYSVAL
Printer device type . . . . . *SCS       *SCS, *IPDS, *LINE...
Text 'description' . . . . . *SRCMBRTXT
:
:
:
```

Reference:

IBM i Information Center - DDS: Printer Files, Keyword entries for printer files

PRTF RPG IV program: Itemlist

```

000100 .HKeywords*****+
000200 H ExprOpts(*ResDecPos)
000300
000400 FItem_PF IF E      K Disk
000500 FItemList 0 E      Printer OFInd(Overflow)
000600 F
000700 D PrtIndicators DS
000800 D LowAvailQty      43   43H
000900
001000 /FREE
001100   Read Item_PF; // Read first record
001200
001300   IF NOT %Eof (Item_PF);
001400     // Print headings on first page if not EOF
001500     Write Heading;
001600   Endif;
001700
001800 DoW NOT %Eof (Item_PF);
001900
002000   IF Overflow;
002100     // Page overflow
002200     Write Heading;
002300     Overflow = *OFF;
002400   Endif;
002500
002600   // Process record for printing
002700   TotQtyAvl = ItmQtyOH + ItmQtyOO;
002800   LowAvailQty = (TotQtyAvl < 15); // Below min?
002900   Write Detail;
003000
003100   Read Item_PF;
003200 EndDo;
003300
003400   // End of file reached
003500 Write Footing;
003600 *INLR = *on;
003700 /END-FREE

```

© Copyright IBM Corporation 2012

Figure 5-14. PRTF RPG IV program: Itemlist

AS067.0

Notes:

This program, together with the PRTF in the previous visual, produces the list of items that look similar to the report that follows these notes.

Note the highlighted points in the program:

1. The printer file name matches the name of PRTF we coded using DDS and then created.
2. As we did with the earlier printer programs, we have a DoW loop that checks for End of File using %Eof. %Eof on (=1) when we reach EOF of the `Item_PF` data file.
3. The Overflow indicator is used to indicate overflow. It is tied to PRTF on the F-spec and is turned on by the system when it senses overflow (page length and overflow line are set in CRTPRTF). Overflow is tested in an IF statement.
4. Indicator 43 (numeric) was referenced in the DDS to be set to true if the Total Quantity Available is less than 15. Note that *in43 is defined in the Printer File DDS. The quantity will be underlined in this situation. Since we use only named indicators in RPG IV, we use the coding technique shown to map each and every indicator defined in a PRTF.

What we have coded in the D-spec is called a data structure. We spend more time explaining data structures in AS07/S6198. For now, just use this technique when working with PRTFs that have indicators.

5. *in43 is mapped to the named indicator, LowAvailQty. LowAvailQty and indicator 43 are now the same.
6. Using an expression, we set LowAvailQty, and, indicator 43, to true or false based on the result of the expression.

Edit codes for printed output



Commas	Zero Balances to Print	No Sign	CR	—	FLT		
Yes	Yes	1	A	J	N	Y=Date	5-9 = User Defined
Yes	No	2	B	K	O		
No	Yes	3	C	L	P	Z=Remove sign from numeric	
No	No	4	D	M	Q		

© Copyright IBM Corporation 2012

Figure 5-15. Edit codes for printed output

AS067.0

Notes:

There are a number of edit codes that you can use in your PRTF DDS. These codes handle many of the most common situations that in other languages might require you to create your own edit code or edit word.

User-defined edit codes are created by using the create edit description (CRTEDTD) command. Edit codes 5 through 9 correspond to IBM-supplied edit code descriptions QEDIT5, QEDIT6, QEDIT7, QEDIT8 and QEDIT9 in the QSYS library. Use the WRKEDTD *ALL command in order to see the detail of the supplied descriptions. To replace an edit code, make sure that the IBM-supplied version is first deleted by using the delete edit description (DLTEDTD) command.

The Y edit code inserts slashes (/) between the month day and year and suppresses the leftmost zero of a date field.

The Z edit code removes the sign (+ or -) from a numeric field and suppresses leading zeros.

The X edit code originally converted signed values to unsigned values without suppressing the leading zeros. The i now does this automatically.

Edit examples: External print file

IBM i

```

A      R SAMPLES
A          1 1TIME
A* Current time presented HH:MM:SS
A      ORDDAT      6 0 5 1EDTCDE(Y)
A* When ORDDAT = 111795, prints as 11/17/95
A      ORDSUM      8 2 10 50EDTCDE(1 $)
A* When ORDSUM = 00150000, prints as $1,500.00
A      BALANCE      7 2 15 50EDTCDE(J *)
A* When BALANCE = Packed '00000250D', it prints as ***25.00-
A      SSNUM       9 0 12 50EDTWRD('0 - - ')
A* When SSNUM = 234513067, prints as 234-51-3067
A          60124'PAGE'
A          +1PAGNBR EDTCDE(Z)
A* PAGNBR - current page number printed, zero suppressed

```

© Copyright IBM Corporation 2012

Figure 5-16. Edit examples: External print file

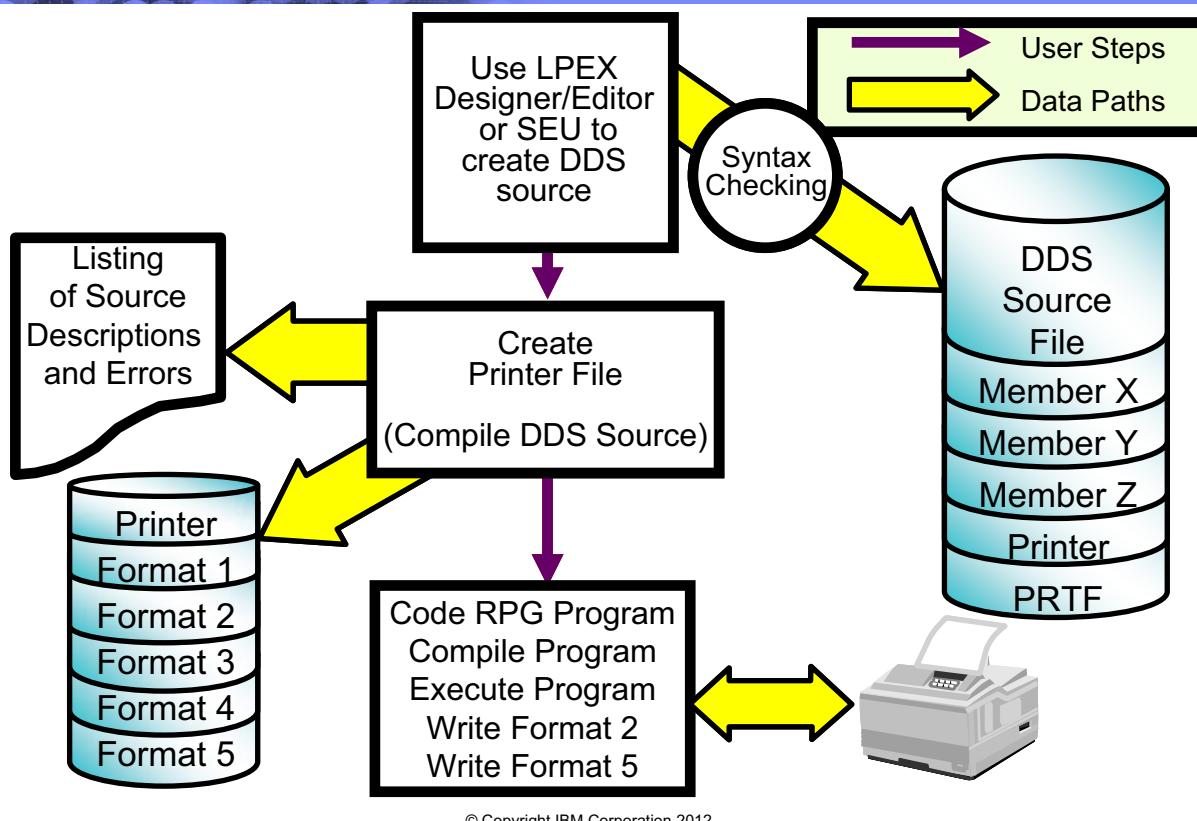
AS067.0

Notes:

This DDS sample coding of a print file shows other ways to position fields on a report. Line numbers can be used instead of the **SPACE** and **SKIP** keywords. You may use one method or the other, but not both in the same record format.

Creating, storing, and executing printer formats

IBM i



© Copyright IBM Corporation 2012

Figure 5-17. Creating, storing, and executing printer formats

AS067.0

Notes:

DDS source members for printer files are specified as type PRTF. When keying the source statements, the prompt DP within SEU provides the proper format.

CRTPRTF is the CL command that compiles or creates the printer file object. There are many parameters available with this command. These parameters correspond to various printer file attributes. Use the attributes appropriate to your needs.

Definition of reports

IBM i

- Report layout externally described in PRTF using DDS:
 - Output specifications in program generated for you
 - You use and tailor your own printer file
 - Can use same PRTF in multiple programs
- Report layout program described in the RPG IV program output specs
 - Use IBM-supplied (generic) printer file

OR

- Create your own printer file shell of just attributes using CRTPRTF
- Specify printer features inaccessible through RPG IV in the PRTF

© Copyright IBM Corporation 2012

Figure 5-18. Definition of reports

AS067.0

Notes:

This visual summarizes your alternatives when writing programs to generate reports. You must always use or reference a print file. If you choose program described report definition, you can use an IBM-supplied printer file and layout your report specifications in RPG output specifications.

You can create a PRTF of just attributes that is complemented by field and record definition of the report layout in your output specifications.

An externally described print file that is written in DDS requires no report related output specs. Several options are available to you when you design reports using DDS (externally described):

- Design using the DDS designer (print) tool of the Rational Developer for Power Systems Software
- Design with screen design aid (and then modify the DDS)
- Design with report layout utility
- Design using a third party tool, such as Gumbo's RDA

Machine exercise: Printing from an RPG IV program

IBM i



© Copyright IBM Corporation 2012

Figure 5-19. Machine exercise: Printing from an RPG IV program

AS067.0

Notes:

Perform the printing from an RPG IV program machine exercise.

Checkpoint

IBM i

1. True or False: Exact line numbers can be used to position a particular field or constant on the report.
2. Which of the following DDS facilities are available when defining printer files?
 - a. Continuation lines
 - b. Edit codes and words edit codes and words
 - c. Field definition referencing
 - d. All of the above
3. The three levels at which DDS printer file keywords can be specified are:

© Copyright IBM Corporation 2012

Figure 5-20. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Use DDS to define the layout of a report
- Create a printer file (PRTF)
- Write an RPG IV program that prints a report using a printer file defined in DDS

© Copyright IBM Corporation 2012

Figure 5-21. Unit summary

AS067.0

Notes:

Unit 6. Using the debugger

What this unit is about

This unit describes the use of the source debugger that is available with ILE. Also mentioned is the debugger packaged with the Rational Developer for Power PC toolset.

What you should be able to do

After completing this unit, you should be able to:

- Describe the benefits of source view and listing view debugging
- Use compile options to specify desired level of debugging
- Use the source debugger to:
 - Set breakpoints
 - Display and change values of program variables
 - Step through the execution of an RPG IV program

How you will check your progress

- Checkpoint questions
- Machine exercise: Given an RPG IV source member, compile it to enable debugging with source and then correct the errors in the program.
- During this exercise, you:
 - Create a program that can be debugged using source code
 - Start the ILE source debugger
 - Set breakpoints
 - Display data
 - Change data

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe the benefits of source view and listing view debugging
- Use compile options to specify desired level of debugging
- Use the source debugger to:
 - Set breakpoints
 - Display and change values of program variables
 - Step through the execution of an RPG IV program

© Copyright IBM Corporation 2012

Figure 6-1. Unit objectives

AS067.0

Notes:

Source view debugger

IBM i

- IBM i debugger for ILE programs
- View source on display in debug mode:
 - Add or remove breakpoints based on cursor position
 - Step through source statements
 - Display values of program variables
 - Change values of program variables
 - Watch the value of a variable change as program executes
- Includes debug commands to:
 - Display values of variables, structures and expressions
 - Change values of variables, structures and expressions
 - Set a watch for a variable
 - Search for strings in source code
 - Set up shorthand notation to reduce keying
 - Navigate through the source code

© Copyright IBM Corporation 2012

Figure 6-2. Source view debugger

AS067.0

Notes:

The i has a number of functions designed to support your application development efforts. One of them is the ILE source debugger. We discuss it and how it can be used in this topic.

Beginning a debug session

IBM i

- STRDBG command works for ILE programs as well as OPM:
 - Programs listed may be ILE, OPM, or a combination
 - Specific parameters for source debugging
- Source for the first module is displayed
- Add any breakpoints desired on the source display.
- Then, call the first program:
 - Exit module source and call from CL command line, or
 - Use function key to get command line on source screen and call

F11 Display variable
F14 Work with modules list

© Copyright IBM Corporation 2012

Figure 6-3. Beginning a debug session

AS067.0

Notes:

Most debug functions are not active unless the program is executing. STRDBG does not begin execution. It simply sets up the debugging environment.

When you first use the ILE debugger, what happens might not be what you expect. When you see the display module source screen after having entered the STRDBG command, you have not called the program yet. Thus, the functions to step or to display variables or to watch variables do not work at this point.

It is necessary to call the program after setting any breakpoints and begin actual execution of the module before these features can be used.

When execution is begun, the display module source screen will appear and the source will be positioned at the first breakpoint, whose line is highlighted on the screen. More breakpoints can be added, of course, at any time program execution is halted.

Compiling a program for debug

IBM i

Create Bound RPG Program (CRTBNDRPG)

Type choices, press Enter.

Program	> <u>IITEMLIST</u>	Name, *CTLSSPEC
Library	> <u>AS06V6LIB</u>	Name, *CURLIB
Source file	<u>QRPGLESRC</u>	Name, QRPGLESRC
Library	<u>*LIBL</u>	Name, *LIBL, *CURLIB
Source member	<u>*PGM</u>	Name, *PGM
Source stream file		
Generation severity level	<u>10</u>	0-20
Text 'description'	<u>*SRCMBRTXT</u>	
Default activation group	<u>*YES</u>	*YES. *NO
Debugging views	> <u>*ALL</u>	*STMT, *SOURCE, *LIST...
Debug encryption key	<u>*NONE</u>	
Output	<u>*PRINT</u>	*PRINT, *NONE
Optimization level	<u>*NONE</u>	*NONE, *BASIC, *FULL
Source listing indentation	<u>*NONE</u>	Character value, *NONE
Type conversion options	<u>*NONE</u>	*NONE, *DATETIME, *GRAPHIC...
+ for more values		

© Copyright IBM Corporation 2012

Figure 6-4. Compiling a program for debug

AS067.0

Notes:

To use the source view part of the debugger, the programs or modules must be compiled specifying source, listing or copy for the debugging view parameter. None of these is the default option. By default, RPG IV modules will have only statement level debug available.

Note the debug view parameter in the CRTBNDRPG command. If you leave this parameter set to its default, *STMT, you are to specify the source line number but no source code is displayed. This option makes debugging difficult. To get the most function, specify *ALL. This will include source, listing and copy views of your source during debug.

As you see during the exercise, you usually want to change the default on this parameter from *STMT to *ALL to see the detail, you will need during a debug session.

Using the ILE debugger

IBM i

- STRDBG PGM (RPG4PGM) UPDPROD (*NO/*YES):
 - View source
 - Set breakpoints
- Call PGM (RPG4PGM)
 - View source, set/clear breakpoints, display/change variables, and so forth
- ENDDBG
- Correct source, recompile
- Test
- Repeat process until satisfied

© Copyright IBM Corporation 2012

Figure 6-5. Using the ILE debugger

AS067.0

Notes:

This is the process you would follow to use the ILE debugger to help determine the cause of any program malfunctions.

STRDBG begins the debug process. Assuming you compiled your program with a DBGVIEW that allows you to view the source, you can set breakpoints as desired in your program. When you call your program, you work with program variables and set watch conditions.

Example of source debugging

The screenshot shows a window titled "Session A - [24 x 80]" with the "Source" tab selected. The menu bar includes File, Edit, View, Communication, Actions, Window, and Help. Below the menu is a toolbar with various icons. The source code listed is:

```

Program:    ITEMLIST      Library:   AS06V6LIB      Module:    ITEMLIST
1           H ExprOpts(*ResDecPos)
2
3           FItem_PF  IF   E          K Disk
4           FItemList 0   E          Printer Oflind(Overflow)
5           F                           IndDS(PrtIndicators)
6
7           D PrtIndicators DS
8           D LowAvailQty        43     43N
9
10          /FREE
11          Read Item_PF; // Read first record
12
13          If NOT %Eof (Item_PF);
14          // Print headings on first page if not EOF
15          Write Heading;

```

A "More..." button is visible at the bottom right of the code area. Below the code, a debug command line shows "Debug . . ." followed by a cursor. A key legend at the bottom left lists F3 through F24 with their respective functions. The status bar at the bottom right shows the date "20/016".

Figure 6-6. Example of source debugging

AS067.0

Notes:

When you issue the command STRDBG for the ILE program, ITEMIST, the source code is displayed.

Here are some things that you can do to set up your debug session for this (or any specific program):

- Position cursor on a line and press **F6** to add or remove a breakpoint.
- Position cursor on a variable name and press **F11** to see its value.
- Press **F10** to step through execution of the code one source line at a time or type `step` on the debug command line.

This RPG module was created with DBGVIEW(*ALL). By default, STRDBG will display the module source for the program entry procedure (the main entry point for this ILE program. Remember, it might have multiple procedures.)

You can see a **debug command line**. You do not enter system commands on this line. You can only enter special ILE debug commands that are different from the OPM ADDBKPs,

DSPPGMVAR, and so on. Note that the OPM debug commands, such as ADDBKP and DSPPGMVAR are not supported by the ILE source debugger.

Typically, to set a breakpoint, position your cursor on a line on the screen and press **F6**. The line number highlights on the screen to indicate a breakpoint setting. Alternatively, you could use the **BREAK** command to set breakpoints by statement number (the one listed on the far left).

Positioning your cursor on (or near) a variable name, such as **TotQtyAvail** or **Overflow** and pressing **F11** will display the variable's value at the bottom of the screen.

The **F10** key steps through execution of the code 1 source line at a time.

For your reference, here is a complete list of the debugger function keys and a description of their use:

- **F3** = End program
- **F6** = Add/Clear breakpoint

Position the cursor at line 20 and press **F6** to set a breakpoint or type **break 26** on the debug command line.

Position the cursor at line 26 and press **F6** to remove the breakpoint or type **clear 26** on the debug command line.

- **F5** = Refresh
- **F9** = Retrieve

Press **F9** to retrieve a previously entered debug command.

- **F10** = Step
- **F11** = Display variable

Position the cursor under the variable and press **F11** or type **eval TotQtyAvail** to display contents of the variable.

- **F12** = Resume
- **F13** = Work with module breakpoints
- **F14** = Work with module list
- **F15** = Select view
- **F16** = Repeat find
- **F17** = Watch variable
- **F18** = Work with watch
- **F19** = Scroll left
- **F20** = Scroll right
- **F21** = Command entry
- **F22** = Step into
- **F23** = Display output (not commonly used for RPG IV)
- **F24** = More keys

Using ILE debug commands

IBM i

BREAK: Set breakpoints

```
BREAK 40
```

```
BREAK 42 WHEN *IN02 = '1'
```

*Conditional breakpoints
use
WHEN*

CLEAR: Remove specific or all breakpoints

```
CLEAR 40
```

```
CLEAR PGM
```

STEP: Execute one or more statements and stop

```
STEP (executes one statement and stops)
```

```
STEP 10 (executes 10 statements and stops)
```

```
STEP INTO (steps into the source code for a module and stops)
```

```
STEP OVER (executes the code in another module, but steps over the  
code)
```

© Copyright IBM Corporation 2012

Figure 6-7. Using ILE debug commands

AS067.0

Notes:

Note that BREAK, CLEAR and STEP can be used with ILE modules that are compiled without source view capability.

The debug commands used to set and clear breakpoints and to step through can be used whether the module has been compiled with the source or listing level debug view. The BREAK command is the easiest way to set a conditional breakpoint. Another option is to use **F13**, get a list of existing breakpoints and add another to the list specifying a conditional expression.

Using the debug EVAL command

IBM i

- EVAL: Display or change the value of:

- Variables
- Expressions
- Records
- Data structures or arrays

```
>      EVAL Custno
          CUSTNO = '0010500'
>      EVAL      *in(1..9)
          *IN OF *ind_id(1) = '0'
          *IN OF *ind_id(2) = '0'
          *IN OF *ind_id(3) = '0'
          *IN OF *ind_id(4) = '1'
          *IN OF *ind_id(5) = '0'
          *IN OF *ind_id(6) = '0'
          *IN OF *ind_id(7) = '0'
          *IN OF *ind_id(8) = '0'
          *IN OF *ind_id(9) = '0'
>      EVAL      *in03 = '1'
          *in03 = '1'
```

© Copyright IBM Corporation 2012

Figure 6-8. Using the debug EVAL command

AS067.0

Notes:

The ILE debugger's EVAL command can be entered on the command line on the **Display Module Source Screen**. The results of the Eval command appear at the bottom of the debug display.

If the requested value cannot be shown in a single line, a separate display, **Evaluate Expressions**, is shown in a format similar to what you see in the visual.

EVAL is used both to display and to change the values of variables.

EVAL has a number of uses as a debug command:

- You can use the EVAL debug command to display a field in character or hexadecimal format. To display a variable:

```
EVAL field-name: c number-of-characters
EVAL field-name: x number-of-bytes
```

Field-name is the name of the field that you want to display. The *c* or *x* specifies either character or hex format. *Number of characters* or *bytes* specifies the length to display.

- The following special debugger built-in functions are available while using the ILE source debugger:

%SUBSTR: Substring a string field.

%ADDR: Retrieve the address of a field.

%INDEX: Change the index of a table or multiple-occurrence data structure.

%VAR: Identifies the specified parameter as a variable. Use the %VAR debug built-in function when the variable name conflicts with any of the debug command names. For example, EVAL %VAR (EVAL) can be used to evaluate a variable named EVAL, whereas EVAL EVAL would be a syntax error.

Reference:

ILE RPG Programmers Guide, Debugging Programs and Handling Exceptions

Watching a variable

The screenshot shows the IBM i debugger interface with the title "Session A - [24 x 80]". The menu bar includes File, Edit, View, Communication, Actions, Window, and Help. Below the menu is a toolbar with icons for Display, Module, and Source. The main window displays the following RPG IV program code:

```

Program: DEBUGDEMO Library: AS06V6LIB Module: DEBUGDEMO
1 // Declare variables
2 D Count      S          3P 0 Inz(1)
3 D Sum        S          5P 0
4
5 // Loop 5 times and accumulate
6 /FREE
7   Dow Count <= 5;
8     Sum = Sum + Count;
9     Count = Count + 1;
10    Enddo;
11
12 // End program - what are the values of Sum and Count ?
13 *InLR = *ON;
14 Return;
15 /END-FREE

```

A "Bottom" status bar at the bottom of the code window shows "Debug . . . watch sum". Below the code window is a keyboard legend:

F3=End program	F6=Add/Clear breakpoint	F10=Step	F11=Display variable
F12=Resume	F17=Watch variable	F18=Work with watch	F24=More keys

The status bar at the bottom of the screen shows "I902 - Session successfully started" and the date "20/025".

Figure 6-9. Watching a variable

AS067.0

Notes:

A **watch** is used to monitor the value of an expression or a variable for any changes in value while your program runs. Setting watch conditions is similar to setting conditional breakpoints, with one important difference. Watch conditions *stop the program* as soon as the value of a watched expression or variable changes from its current value.

Conditional job breakpoints stop the program only if a variable changes to the value specified in the condition.

The debugger watches an expression or a variable through the contents of a storage address, computed at the time the watch condition is set. When the content at the storage address is changed from the value it had when the watch condition was set or when the last watch condition occurred, the program stops. The characteristics of a watch are:

- Watches are monitored system-wide, with a maximum number of 256 watches that can be active simultaneously. This number includes watches set by the system.
- Watch conditions can only be set after a program has been called in debug mode.

A watch of a variable or expression can be set by issuing the `watch` command on the debug command line (as shown in the visual) or by placing the cursor under the variable to be watched and pressing **F17**.

Do not use DSPLY or printer files to debug your program. Use the source-level debugger and step through it to see how the program is behaving. If you do not know how or where a variable is changing, use WATCH in the debugger. It shows you where the variable was changed even if it was not in the same module that owns the variable.

More debug commands

FIND: Searches for a line number or string

Command entered:

```
FIND '*INLR'
```

Response:

(Source display positioned to line 57)

ATTR: Returns size and type of variables

ATTR CustNo

Response:

TYPE = FIXED LENGTH STRING, LENGTH = 7 BYTES

EQUATE: Assigns short names

DISPLAY EQUATE: Displays any EQUATE abbreviation

© Copyright IBM Corporation 2012

Figure 6-10. More debug commands

AS067.0

Notes:

The FIND command helps you navigate through large source members. FIND can also be used to search through the debug history you get when you press **enter** without entering a debug command.

The Equate command can be used to assign an abbreviated name to a variable for shorthand use.

There is no need to page back and forth to find where a variable has been defined. Use the ATTR command to find out its attributes.

Source navigation

IBM i

	UP X, DOWN X	Move up or down X lines
	LEFTn, RIGHTn	Move left or right n columns
	TOP, BOTTOM	Move to the top or bottom of the source
-		
	NEXT, PREVIOUS	Move to the next or previous breakpoint
	FIND	Search forward or backward for a line number, string or text

© Copyright IBM Corporation 2012

Figure 6-11. Source navigation

AS067.0

Notes:

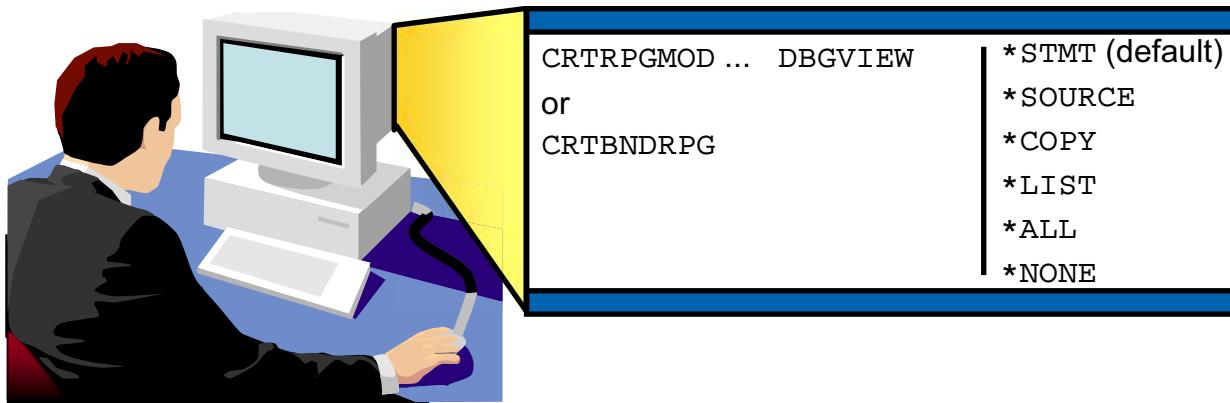
Simple navigation tools to move you through the module source screens.

Note that Left and Right can also be done using function keys (**F19** and **F20**).

Debug views

IBM i

- Programmer chooses debugging views:
 - Statement number only
 - Source from original source file
 - Compiler-generated listing
 - No debugging capability



© Copyright IBM Corporation 2012

Figure 6-12. Debug views

AS067.0

Notes:

When the *NONE option for debug view is specified, the DUMP operation code is *not* supported.

Note that debug data can be removed with CHGMOD or CHGPGM. Debug data can be removed without removing the ability to retranslate the program.

If source or copy view is chosen, the source member must be available at the time of the debug session.

The *LIST option saves a copy of the source listing report with the program object.

Note also that the storage requirements for a module or program vary somewhat depending on the type of debug data included with it. The following values for the DBGVIEW parameter are listed in increasing order based on their effect on secondary storage requirements:

1. *NONE
2. *STMT

3. *SOURCE
4. *COPY
5. *LIST
6. *ALL

If you are debugging a program that was created from source in a different library, or the source member is named differently from the *PGM object, you might have to OVRDBF FILE (QRPGLESRC) or use the QLICOBJD API to work with the source. The debugger does not find the source on its own unless the name and library match where the program object is located. You cannot specify where the source member is located in the STRDBG command.

Note that if you step on a READ or WRITE, you normally start stepping through I-specs and O-specs. If the file is externally described and you have specified a source view, the debugger repeatedly steps on the F spec for the file. To avoid this problem, you can compile with the program with H spec option OPTIONS(*NODEBUGIO). I and O specs are not debugged.

Machine exercise: Debugging an RPG IV program

IBM i



© Copyright IBM Corporation 2012

Figure 6-13. Machine exercise: Debugging an RPG IV program

AS067.0

Notes:

Perform the debugging an RPG IV program machine exercise.

Source debugger summary

IBM i

- Source view debugging
- Simple setting of breakpoints
- Simple display of variable values
- Navigation commands
- Step option
- Evaluation of expressions
- BIFs can be used with debugger

© Copyright IBM Corporation 2012

Figure 6-14. Source debugger summary

AS067.0

Notes:

This summarizes the ILE debugger.

But, what about OPM programs?

To debug OPM RPG/400 programs:

- Recompile OPM programs with desired DBGVIEW
- Specify *SRCDBG as the OPTION parameter (source listing options) of CRTBNDRPG
- Specify *YES for the OPMSRC parameter of the STRDBG command

This means that the ILE source debugger can be used as your single debug native i debugging tool.

And, there are PC tools as well. The **integrated debugger** is IBM's PC-based debugger.

With a single user interface, you can debug RPG IV (and other) programs from your Windows PC.

The i debug engine uses API (Application Program Interface) calls to the system debugger to obtain all necessary information.

Your application still executes on the i host.

Features of integrated debug

IBM i

- The debugger client user interface allows you to:
 - View source code or compiler listings
 - Control program execution
 - Set, change, delete, enable, and disable line breakpoints
 - Set watch breakpoints to make your program stop when a variable changes.
 - Step through code one line at a time.
 - Display variables and their values
 - Display programs, service programs, and ILE modules
- Same functions as STRDBG, but:
 - Multiple views in single window
 - Can see more
 - Easier to track program flow

© Copyright IBM Corporation 2012

Figure 6-15. Features of integrated debug

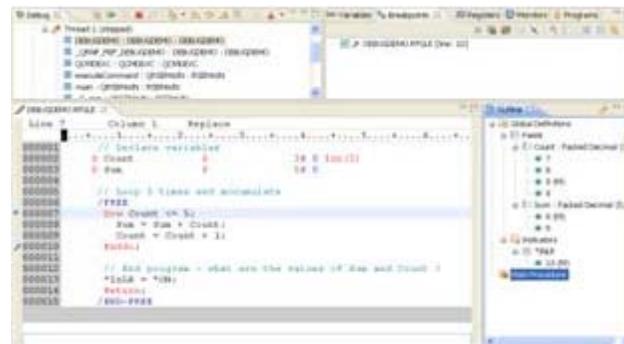
AS067.0

Notes:

Starting the debug server

IBM i

- Can be started from RSE
- On the i start the debug server (STRDBGSRV)
- **Note:** If the debug server is not started, an error message appears when you try to debug from your PC.
- To debug interactive programs, you need a 5250 session.
- TCP/IP must be configured on workstation and the i.



© Copyright IBM Corporation 2012

Figure 6-16. Starting the debug server

AS067.0

Notes:

A message will indicate that the debug server is not started.

To start the debug server from Remote Systems Explorer (RSE), locate the **Objects** icon under the IBM i of choice and right-click **Objects > Remote Servers > Debug > Start**.

As an alternative, the STRDBGSRV command could be issued on the IBM i. Once started, the debug server remains active until it is explicitly ended (ENDDBSVR) or the IBM i is re-IPLed.

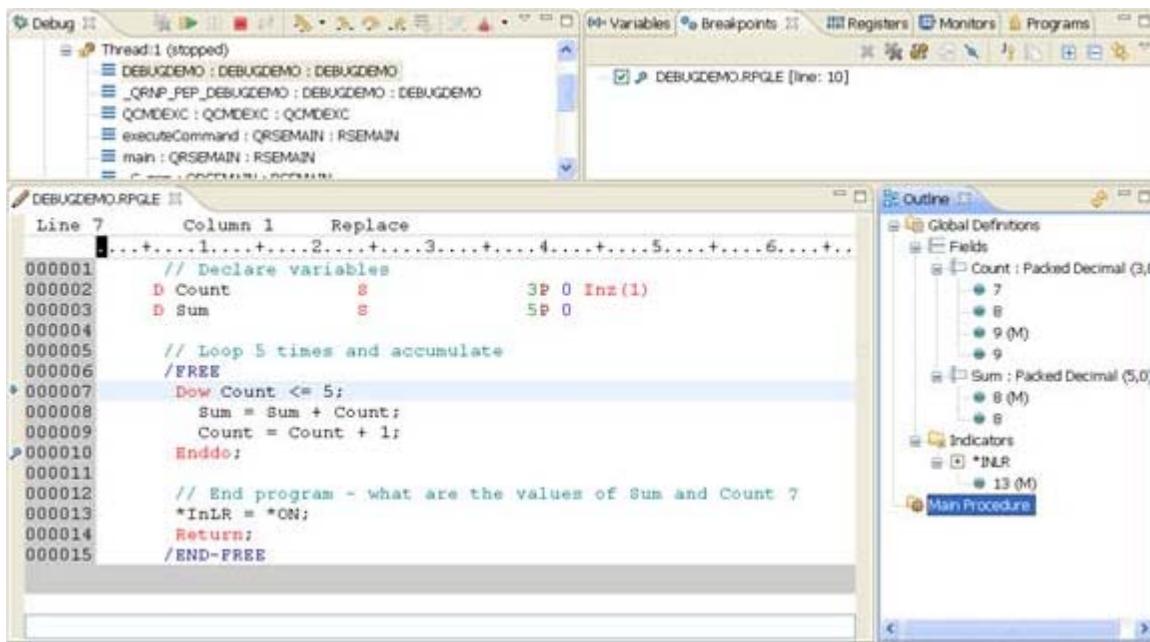
The debug perspective should open when debug of a program is requested.

If the Remote Systems Explorer communications server has been shut down and needs to be restarted, go to your 5250 emulator session and restart the Remote System Explorer communications server by following the instructions in the message.

Example of integrated debug

IBM i

- Integrated PC workbench, component of RD for Power
- Views show variables, source flow, and program structure.



© Copyright IBM Corporation 2012

Figure 6-17. Example of integrated debug

AS067.0

Notes:

This example illustrates the integrated debug tool. This tool is a component of the Rational Developer for Power Workbench.

Integrated debug provides similar facilities to the debug tools we have already discussed.

From the RSE/LPEX Editor view, find the program from the listed libraries in your library list. Right-click it, then select the **Debug (Service Entry) > Set Source Entry Point** icon.

Tools

IBM i

- DEBUG on H-specification
- DUMP opcode
- STRSRVJOB
- Joblog

© Copyright IBM Corporation 2012

Figure 6-18. Tools

AS067.0

Notes:

Now that we have discussed the ILE source debugger, we cover some other tools that are available to assist you to determine the answer to the question, “What caused my program to fail?”

Debug and dump

IBM i

```

Line 1      Column 1    Replace
000001      Debug(*yes)
000002
000003      // Declare variables
000005      D Count      S          3P 0 Inz(1)
000006      D Sum       S          5P 0
000007
000008      // Loop 5 times and accumulate
000009      /FREE
000010      Dow Count <= 5;
000011      Sum = Sum + Count;
000012      Count = Count + 1;
000013      Enddo;
000014
000015      Dump;
000016      // End program - what are the values of Sum and Count ?
000017      *InLR = *ON;
000018      Return;
000019      /END-FREE

```

DUMP(A) can also be used

© Copyright IBM Corporation 2012

Figure 6-19. Debug and dump

AS067.0

Notes:

The **DEBUG** keyword determines whether DUMP operations are performed. DUMP operations are performed when **DEBUG** or **DEBUG (*YES)** is specified on the H-spec.

The DUMP operation provides a dump of all data (fields, files, indicators, and so forth) defined in the program.

The DUMP operation is performed if the **DEBUG** keyword is specified on the control specification. If the **DEBUG** keyword is not specified, the DUMP operation is checked for errors and the statement is printed on the listing, but the DUMP operation is not processed. If the **DUMP(A)** operation is used, a dump of all data fields, files, indicators, and so forth occurs even if **DEBUG (*NO)** or no **DEBUG** keyword is specified. The operation extender means that a dump is always performed, regardless of the value of the **DEBUG** keyword.

For DUMP to work, the **DBGVIEW(*NONE)** compiler option must *not* be specified when compiling as observability information required for DUMP would not be included in the ***PGM** or ***MOD** object.

When the OPTIMIZE(*FULL) compiler option is selected, the field values shown in the dump might not reflect the actual content due to the effects of optimization.

Reference:

ILE RPG Language Reference, Operation Codes Detail

Sample DUMP output

IBM i

```

ILE RPG FORMATTED DUMP
Program Status Area:
Procedure Name . . . . . : DEBUGDEMO
Program Name . . . . . : DEBUGDEMO
  Library . . . . . : AS06001
Module Name. . . . . : DEBUGDEMO
Program Status . . . . . : 00000
Previous Status . . . . . : 00000
Statement in Error . . . . . : 00000000
RPG Routine . . . . . : *DETC
Number of Parameters . . . . . :
Message Type . . . . . :
Additional Message Info . . . . . :
Message Data . . . . . :
Status that caused RNX9001 . . . . . :

Internal Indicators:
LR '0'    MR '0'    RT '0'    1P '0'
NAME          ATTRIBUTES           VALUE
COUNT         PACKED(3,0)          004.            '004F'X
SUM          PACKED(5,0)          00006.          '00006F'X

```

© Copyright IBM Corporation 2012

Figure 6-20. Sample DUMP output

AS067.0

Notes:

This is a sample page of a dump that is generated as the result of executing the DUMP opcode.

In your programs, you would determine where to place the DUMP opcode in your logic in order to assist you in the debugging process.

STRSRVJOB and TRCJOB

IBM i

- STRSRVJOB
- TRCJOB *ON/*OFF/*END

© Copyright IBM Corporation 2012

Figure 6-21. STRSRVJOB and TRCJOB

AS067.0

Notes:

The Start Service Job (STRSRVJOB) command starts the remote service operation for a specified job (other than the job issuing the command) so that other service commands can be entered to service the specified job. Dump, debug, and trace commands can be run in that job until service operation ends. This command is often used to debug batch jobs. Service operation continues until the End Service Job (ENDSRVJOB) command is run.

The Trace Job (TRCJOB) command, which functions with both OPM and ILE programs, controls traces of original program model (OPM) programs and Integrated Language Environment (ILE) procedure calls and returns that occur in the current job or in the job being serviced as a result of the Start Service Job (STRSRVJOB) command directed to that job. This command, which sets a trace on or off, can trace module flow, IBM i (i5/OS, OS/400) system data acquisition (including CL command traces), or both.

As the trace records are collected, they are stored in an internal trace storage area. When the trace is ended, the trace records can be written to a spooled printer file, QPSRVTRC. The trace records can also be directed to a database output file.

If the start service job (STRSRVJOB) command is entered before the TRCJOB command, the job that is traced is the one identified by the STRSRVJOB command. The trace output from the serviced job is returned to the servicing job after the trace is set off or after the serviced job has ended. Debugging a batch job can be a very delicate task. Specific steps have to be followed in sequence in order to use the source debug facilities:

1. Submit job to run in batch with HOLD(*YES).
2. Use WRKSBMJOB to obtain information about the job you want to debug, that is:
 - Job ID
 - User name
 - Job number
3. Enter the command STRSRVJOB for the job above using the information that you retrieved using WRKSBMJOB.
4. Enter the STRDBG command for the program that the batch job is running. The source view of the program is displayed.
5. Press **F21** to obtain a command line.
6. Type WRKSBMJOB from the command line to release the batch job. A system message will appear allowing you to press **F10** to enter debug commands or enter to run the program.
7. Press **F10**. A command entry screen displays.
8. Use DSPMODSRC from the command entry display and set breakpoints where desired.
9. Exit DSPMODSRC (**F3**).
10. Exit the command entry screen (**F12**). The system message from step 6 will display again.
11. Press **Enter** to run the program.
12. Program now halts at breakpoint.
13. Debug program as normal.
14. When the program or job has completed execution, close out the process with ENDDBG and ENDSRVJOB.

Using the joblog as a debugging tool (1 of 2)

IBM i

© Copyright IBM Corporation 2012

Figure 6-22. Using the joblog as a debugging tool (1 of 2)

AS067.0

Notes:

Often, you receive a message that indicates a problem. You have the option of asking for help.

By pressing **F1**, you can receive help information that can lead you to determine the cause of the problem. In this example, we know that the system had a problem opening a file.

Using the joblog as a debugging tool (2 of 2)

IBM i

3 Joblog

4> CALL OL86V4LIB/BFRSUBST

File BFDSUBST in library *LIBL not found or inline data file missing.

Error message CPF4101 appeared during OPEN for file BFDSUBST.

Function check. RNX1216 unmonitored by BFRSUBST at statement 000100001,
instruction X'0000'.

Error message CPF4101 appeared during OPEN for file BFDSUBST (C S D F).

Error message CPF4101 appeared during OPEN for file BFDSUBST (C S D F).

© Copyright IBM Corporation 2012

Figure 6-23. Using the joblog as a debugging tool (2 of 2)

AS067.0

Notes:

In the Help screen, when we press **F10 Display messages in job log**, we find the reason why the system was unable to open the file. It could not find it.

Can you think of any reason why the system would not be able to find a data file?

Checkpoint

IBM i

1. True or False: If all the shipped system defaults are used when creating RPG IV modules and programs, the programs will be able to use source view debugging.
2. To change the value of a variable during a source or listing view debug session, you would:
 - a. Press **F11** and key the new value over the previous one
 - b. Use CHGPGMVAR command from the debug command line
 - c. Press **F21** to get a system command line and key CHGPGMVAR
 - d. Use the EVAL command on the debug command line
3. Which of the following parameter values for debug view (DBGVIEW) would normally result in the largest total program or module size?
 - a. *LIST
 - b. *COPY
 - c. *SOURCE
 - d. *STMT
 - e. *ALL

© Copyright IBM Corporation 2012

Figure 6-24. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe the benefits of source view and listing view debugging
- Use compile options to specify desired level of debugging
- Use the source debugger to:
 - Set breakpoints
 - Display and change values of program variables
 - Step through the execution of an RPG IV program

© Copyright IBM Corporation 2012

Figure 6-25. Unit summary

AS067.0

Notes:

Perform the ILE source debugger exercise.

Unit 7. Structured programming and subroutines

What this unit is about

This unit describes how to write structured RPG code. It discusses how to build and control loops, how to condition execution of RPG IV code, and how to write and use subroutines within an RPG IV program.

This unit further describes the use of If, Select/When/Other, and Iter/Leave in loops. You also learn how to execute subroutines using ExSr.

What you should be able to do

After completing this unit, you should be able to:

- Write structured RPG IV code
- Write RPG IV code using conditional IF, DOW, DOU and FOR opcodes
- Write RPG IV code using conditional SELECT groups
- For a nested group, determine the number of nesting levels
- Code a subroutine in an RPG IV program

How you will check your progress

- Checkpoint questions
- Machine exercise

In this exercise, the students are given a program that they modify to produce a more structured program by coding subroutines.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Write structured RPG IV code
- Write RPG IV code using conditional IF, DOW, DOU, and FOR opcodes
- Write RPG IV code using conditional SELECT groups
- For a nested group, determine the number of nesting levels
- Code a subroutine in an RPG IV program

© Copyright IBM Corporation 2012

Figure 7-1. Unit objectives

AS067.0

Notes:

Introducing structured programming

The screenshot shows four separate code snippets in the Integrated Problem Editor:

- Example 1:** A subroutine named MySub. It contains an external call to ExSR MySub, a BEGINSR block with a message, and an ENDSR block.
- Example 2:** An IF-ELSE-ENDIF block. It checks if X equals Y. If true, it calls ExSR XeqY; otherwise, it calls ExSR XneY.
- Example 3:** A SELECT group. It has a WHEN clause for X = Y, an OTHER clause, and an ENDSEL block.
- Example 4:** A DO loop. It loops until X is greater than 12, reads a record from a file form, increments X, and ends the loop.

© Copyright IBM Corporation 2012

Figure 7-2. Introducing structured programming

AS067.0

Notes:

This visual overviews some of the functions available in RPG IV to assist the programmer to write structured programs. Some of the techniques that you can code include:

- IF/ELSE/ELSEIF/ENDIF
- Looping Dow/DoU/FOR
- SELECT groups
- Subroutines

External calls to other programs or procedures are covered in the AS07/S6198 class that follows this one.

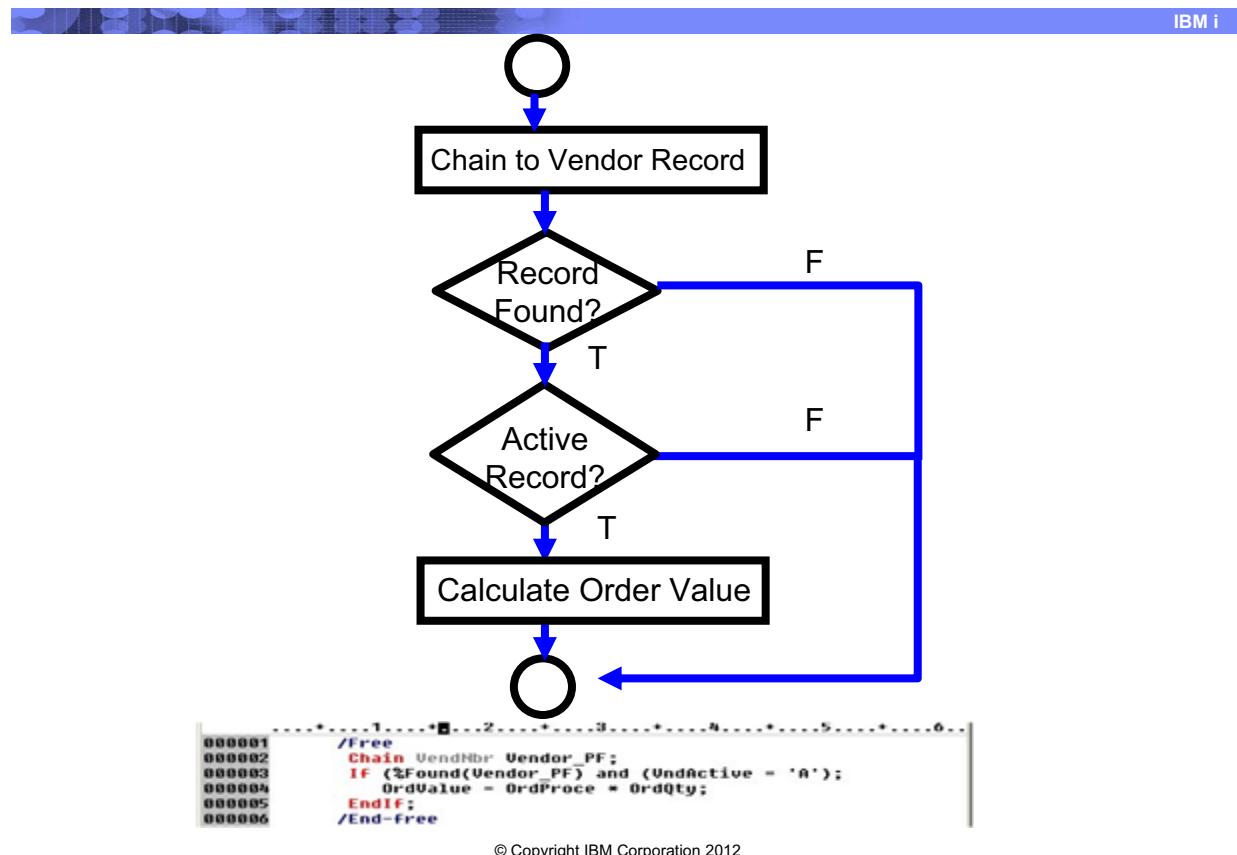
IF

Figure 7-3. IF

AS067.0

Notes:

The IF operation code is one of the opcodes used to condition the flow of your calculations. Note the form of the IF opcode in this example. The result of the expression is either true or false. If the condition is true, the code that follows the IF is executed. Otherwise, the program skips directly to the ENDIF statement.

Each IF group must be terminated with an ENDIF statement. Since IF groups can be nested, the ENDIF not only ensures that you correctly terminate an IF, it also makes your code more easily read and therefore maintained.

In this example, we only want to process active records. Therefore, we condition the IF such that we test for an active record and also that end of file has not been reached.

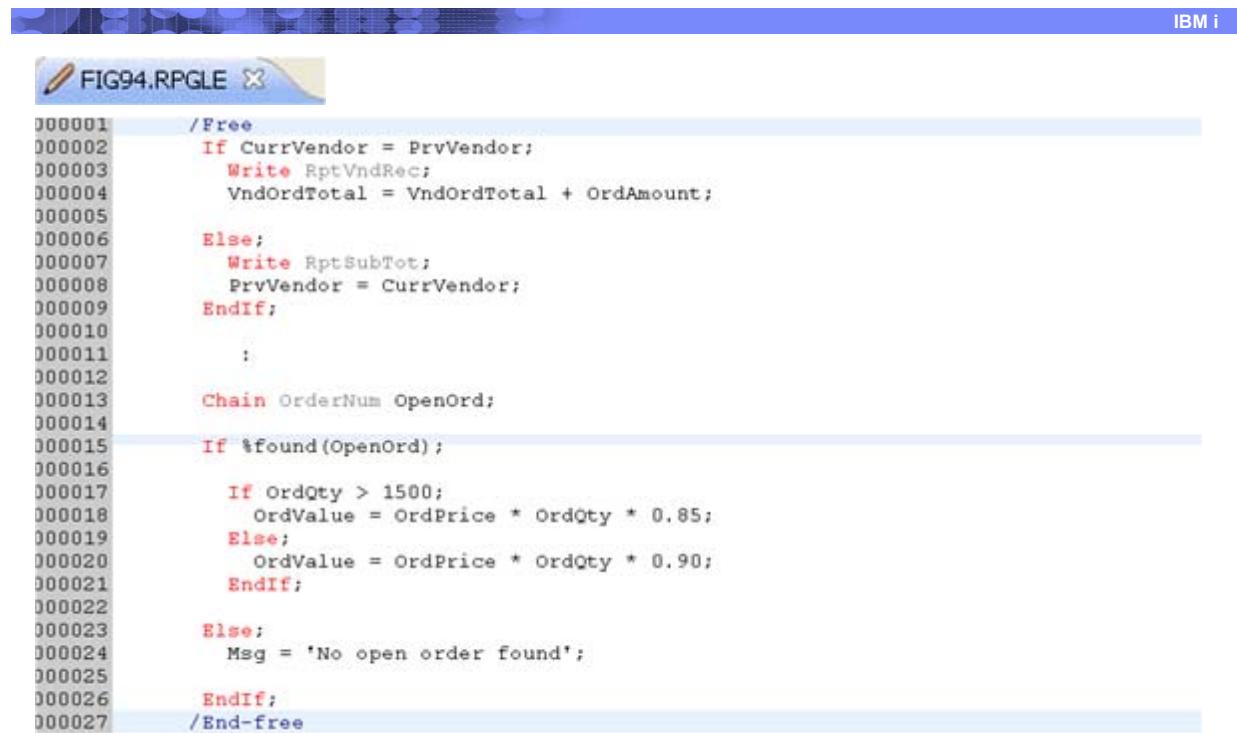
You can see a new opcode (CHAIN) and a new BiF (%found) in this visual.

CHAIN performs a random read of a record using a search argument (in this case VndNbr). This search argument is matched to the record key.

The %found BiF is coded and set to true or false (1 or 0) depending on the whether a record was found in the previous input operation (CHAIN, in this example). Although the

filename parameter is optional, we recommend that you *always* code it to make your code clear and easier to maintain.

IF and ELSE



The screenshot shows an IBM i application window titled 'FIG94.RPGLE'. The code displays the structure of IF and ELSE statement groups in RPGLE. The code includes logic for writing vendor records, calculating totals, and handling open orders based on quantity thresholds.

```

000001      /Free
000002      IF CurrVendor = PrvVendor;
000003          Write RptVndRec;
000004          VndOrdTotal = VndOrdTotal + OrdAmount;
000005
000006      Else;
000007          Write RptSubTot;
000008          PrvVendor = CurrVendor;
000009      EndIf;
000010
000011      :
000012
000013      Chain OrderNum OpenOrd;
000014
000015      If *found(OpenOrd);
000016
000017          If OrdQty > 1500;
000018              OrdValue = OrdPrice * OrdQty * 0.85;
000019          Else;
000020              OrdValue = OrdPrice * OrdQty * 0.90;
000021          EndIf;
000022
000023      Else;
000024          Msg = 'No open order found';
000025
000026      EndIf;
000027  /End-free

```

© Copyright IBM Corporation 2012

Figure 7-4. IF and ELSE

AS067.0

Notes:

This visual is designed to show you the structure of **IF** statement groups:

- The rules for the comparison when using **IF** are the same as you would expect for any logical comparison. The fields must be the same type.
- Each **IF** must be terminated by an **ENDIF**.
- The **ELSE** operation is an optional part of an **IF** group. If the **IF** comparison is met, the calculations before **ELSE** are processed. Otherwise, the calculations following the **ELSE** are processed.

Make your expressions clear

IBM i

```

FIG95.RPGLE X
Line 20      Column 1      Replace      Browse
...../END-FREE.....
000001      D Exit           S           N
000002      D Indicator       S           N
000003
000004      /Free
000005      // Test if indicator is on
000006      If Indicator = *ON;      // These
000007      If Indicator;          // are
000008      If NOT (Indicator = *Off); // equivalent
000009
000010      // Test if indicator is off
000011      If Indicator = *OFF;     // These
000012      If NOT Indicator;       // are
000013      If NOT (Indicator = *ON); // equivalent
000014
000015      // Test if Exit on
000016      If Exit;
000017
000018      // Test if Exit off
000019      If Not Exit;
000020      /End-free

```

© Copyright IBM Corporation 2012

Figure 7-5. Make your expressions clear

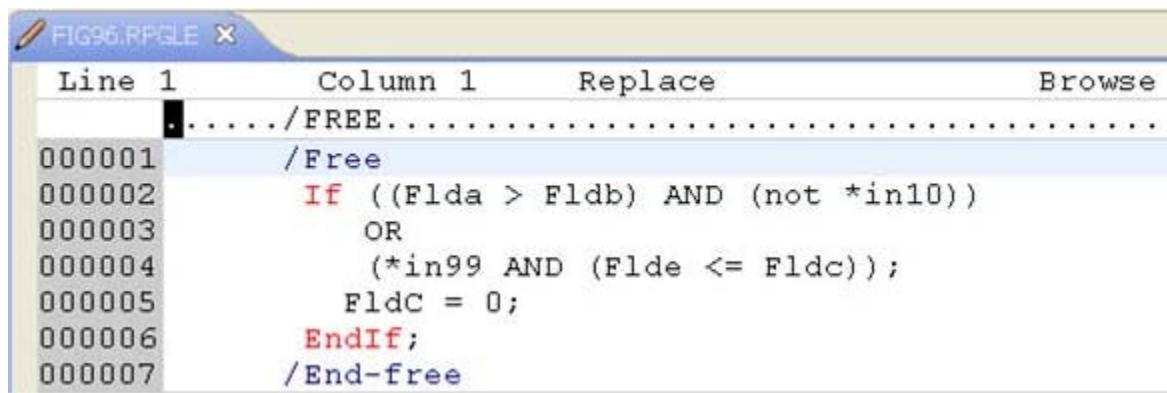
AS067.0

Notes:

Which is more easily understood? We recommend that you always code expressions that explain the condition clearly. Also, you can see that rather than writing **IF Indicator = *on**, it is better to code **IF Indicator**. This is true for testing of ***off**, as well.

AND/OR in expressions

IBM i



The screenshot shows an IBM i RPGLE editor window titled 'FIG96.RPGLE'. The code is as follows:

```
Line 1      Column 1      Replace      Browse
...../FREE.....
000001      /Free
000002      If ((Flida > Flldb) AND (not *in10))
000003          OR
000004          (*in99 AND (Flde <= Fldc));
000005          Fldc = 0;
000006      EndIf;
000007      /End-free
```

© Copyright IBM Corporation 2012

Figure 7-6. AND/OR in expressions

AS067.0

Notes:

This example shows you an example of using AND/OR in expressions.

Note the use of parentheses to clarify how each component of the expression is evaluated. In addition, free format coding enables your code so that it is more easily read.

Nested structures groups

```

FIG77.RPGLE X
Line 1      Column 1      Replace      Browse
...../FREE.....
000001      /Free
000002      DoW Not Exit;
000003
000004      DoU X = 12;
000005      ::
000006      EndDo;
000007
000008      If TrnCode NOT = *blank;
000009
000010      Select;
000011      When TrnCode = 'A';
000012          ExSR Add;
000013      When TrnCode = 'C';
000014          ExSR Change;
000015      When TrnCode = 'D';
000016          ExSR Delete;
000017      Other;
000018          ExSR Error;
000019      EndSL;
000020
000021      Else;
000022      EndIf;
000023
000024      EndDo;
000025      /End-Free
000026

```

© Copyright IBM Corporation 2012

Figure 7-7. Nested structures groups

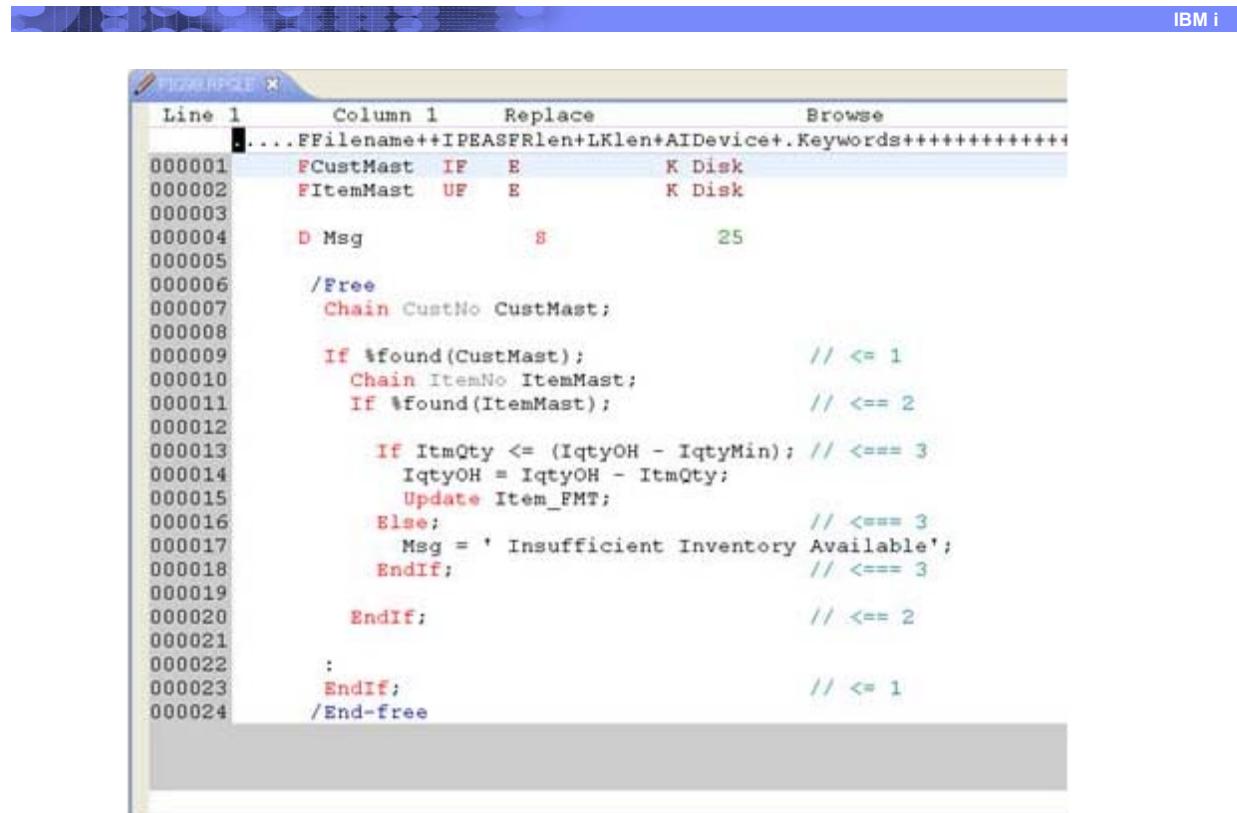
AS067.0

Notes:

If a structured group of various opcodes that can condition calculations contains another complete structured group, together they form a *nested structured group*. Structured groups can be nested to a maximum depth of 100 levels.

The visual is an example of nested structured groups, three levels deep.

Nested IF/ELSE



```

Line 1      Column 1    Replace      Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001    FCustMast  IF   E           K Disk
000002    FItemMast  UF   E           K Disk
000003
000004    D Msg          S           25
000005
000006    /Free
000007    Chain CustNo CustMast;
000008
000009    If %found(CustMast);           // <= 1
000010    Chain ItemNo ItemMast;
000011    If %found(ItemMast);           // <= 2
000012
000013    If IqtyOH <= (IqtyOH - IqtyMin); // <= 3
000014    IqtyOH = IqtyOH - IqtyOH;
000015    Update Item_FMT;
000016    Else;                         // <= 3
000017    Msg = 'Insufficient Inventory Available';
000018    EndIf;                        // <= 3
000019
000020    EndIf;                        // <= 2
000021
000022    :
000023    EndIf;                        // <= 1
000024    /End-free

```

© Copyright IBM Corporation 2012

Figure 7-8. Nested IF/ELSE

AS067.0

Notes:

The 1, 2, and 3 in the visual mark the beginning and end of each group within this nested group.

Notice a new opcode, Update. This opcode replaces a record in a file with changed data. We will see more about Update in the next unit.

SELECT/WHEN/OTHER

IBM i

```

Line 1      Column 1      Replace      Browse
. .... DName++++++ETDsFrom+++To/L+++IDc.Keywords++
000001      D Flag          S           4
000002
000003      /Free
000004      Select;
000005
000006      When RegPay > 600.00;
000007          Flag = 'High';
000008
000009      When RegPay < 100.00;
000010          Flag = 'Low ';
000011      Other;
000012          Flag = 'Avg ';
000013      EndSl;
000014      /End-free

```

© Copyright IBM Corporation 2012

Figure 7-9. SELECT/WHEN/OTHER

AS067.0

Notes:

The select group conditionally processes one of several alternative sequences of operations. It consists of:

- A SELECT statement
- One or more WHEN groups
- An optional OTHER group
- ENDSL statement

Code can be grouped within a SELECT / ENDSL pair. WHEN may be used to condition execution of a SELECT group.

The system compares the value of RegPay for each WHEN statement. If a comparison is satisfied in one of the WHEN statements, the code that follows that WHEN will be executed. Execution of the WHEN group is terminated by the next WHEN, OTHER or ENDSL that is encountered. Any WHEN or OTHER groups that follow the WHEN that satisfied the comparison test will be bypassed. If both WHENs in the example do not satisfy the comparison, the code that follows the OTHER is executed.

IF versus SELECT group

IBM i

```

.....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000001 /Free
000002   If DeptCode = 'A';           // This code is OK
000003     DeptName = 'Accounts';
000004   Else;
000005     If DeptCode = 'S';
000006       DeptName = 'Sales';
000007     Else;
000008       If DeptCode = 'P';
000009         DeptName = 'Production';
000010       Else;
000011         DeptName = 'Unknown';
000012       EndIf;
000013     EndIf;
000014   EndIf;
000015
000016
000017   If DeptCode = 'A';           // Can also be coded like this
000018     DeptName = 'Accounts';
000019   Elseif DeptCode = 'S';
000020     DeptName = 'Sales';
000021   Elseif DeptCode = 'P';
000022     DeptName = 'Production';
000023   Else;
000024     DeptName = 'Unknown';
000025   EndIf;
000026
000027
000028   Select;                  // Or even better like this
000029   When DeptCode = 'A';
000030     DeptName = 'Accounts';
000031   When DeptCode = 'S';
000032     DeptName = 'Sales';
000033   When DeptCode = 'P';
000034     DeptName = 'Production';
000035   Other;
000036     DeptName = 'Unknown';
000037 EndSel;
000038 /End-free

```

© Copyright IBM Corporation 2012

Figure 7-10. IF versus SELECT group

AS067.0

Notes:

This example shows a combination of the use of IF and a SELECT/WHEN/OTHER group. Look at the three examples. The coded function is identical. Which is easiest to understand and hence maintain? Also, notice the coding of the ELSEIF statement within an IF rather than a nested IF statement under an ELSE.

DoWhile and DoUntil

IBM i

```
.....+....1...+....2....+....3....+....4....+....5....+....6....+....7....+...
000001    D Month      $          2   0
000002    D YrTot      $          9   2
000003    D MonSal     $          7   2 Inz(3000)
000004
000005    /Free
000006        Month = 1;
000007        DoW Month <= 12; // Do While - check condition BEFORE executing loop
000008            YrTot = YrTot + MonSal;
000009            Month = Month + 1;
000010        EndDo;           // Loop 12 times
000011
000012        Clear Month;  // Month = 0
000013        Clear YrTot;
000014        DoU Month = 12; // Do Until - check condition AFTER executing loop
000015            YrTot = YrTot + MonSal;
000016            Month = Month + 1;
000017        EndDo;           // Loop 12 times
000018
000019        *InLR = *on;
000020    /End-Free
```

© Copyright IBM Corporation 2012

Figure 7-11. DoWhile and DoUntil

AS067.0

Notes:

DOWHILE and DOUNTIL are opcodes that enable you to code a loop. Iterations can be managed in the loop using a counter or a condition in an expression and can be tested to determine whether looping should continue or be terminated. Several operations can change the normal flow of the loop (ITER and LEAVE). A loop will always begin with DoW or DoU and is ended by an ENDDO.

Use blank lines and comments to separate DO groups to make your code more readable.

Within a DoW or DoU group, there are several other statements that you can use.

The LEAVE operation is used within a DO loop to cause a premature exit of the loop before having completed all operations.

The ITER operation transfers control from within a do group to the ENDDO statement of the do group. It can be used in DO loops to transfer control immediately to the ENDDO statement. ITER causes the next iteration of the loop to be executed.

DoWhile

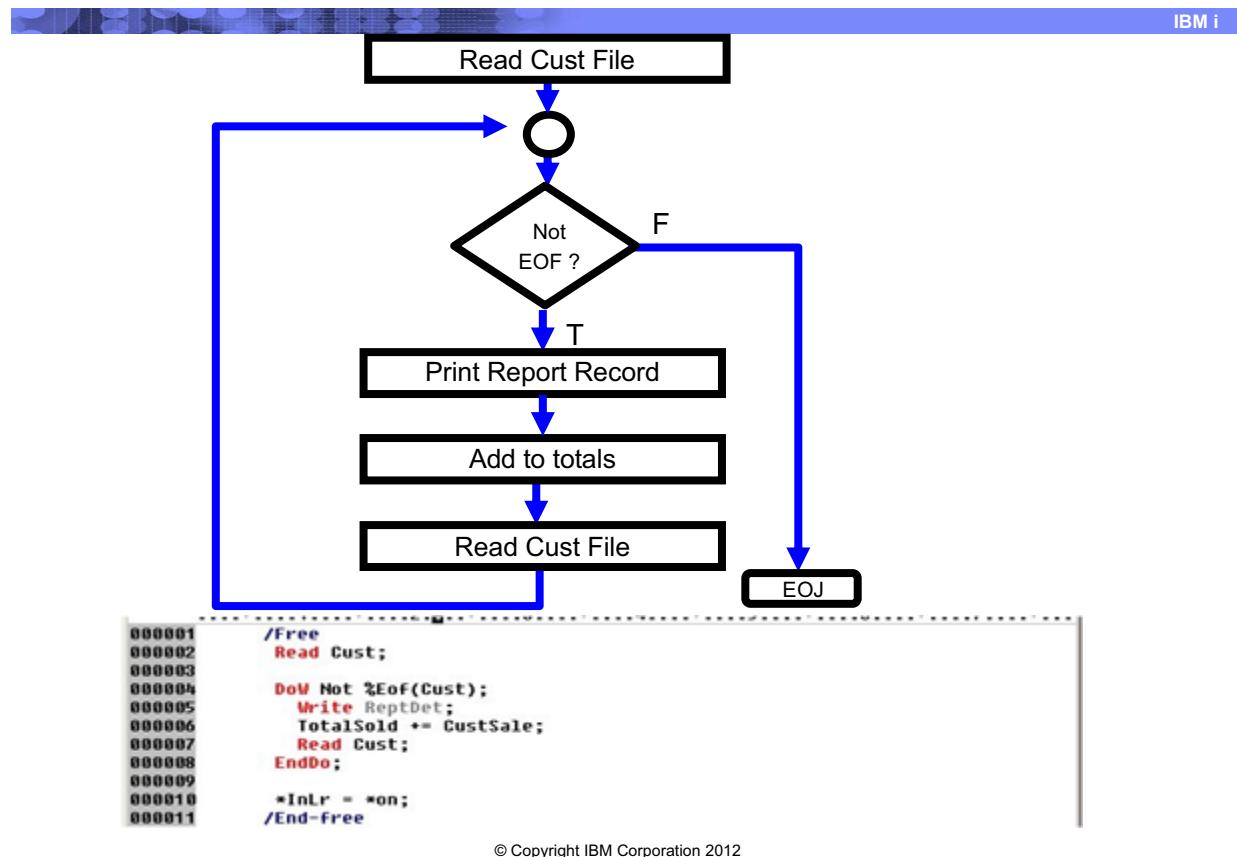


Figure 7-12. DoWhile

AS067.0

Notes:

The DoW operation code precedes a group of operations which you want to process as long as a certain condition exists. An associated ENDDO statement marks the end of the group. A logical condition is expressed by an indicator valued expression (result will be '1' or '0'). The operations controlled by the DoW operation are performed while the expression is true.

The condition is tested before the code within the loop is executed.

In this case, we read the file until we run out of records. As long as we are not at EOF, we want to write a transaction record on the report and accumulate a total.

DoUntil

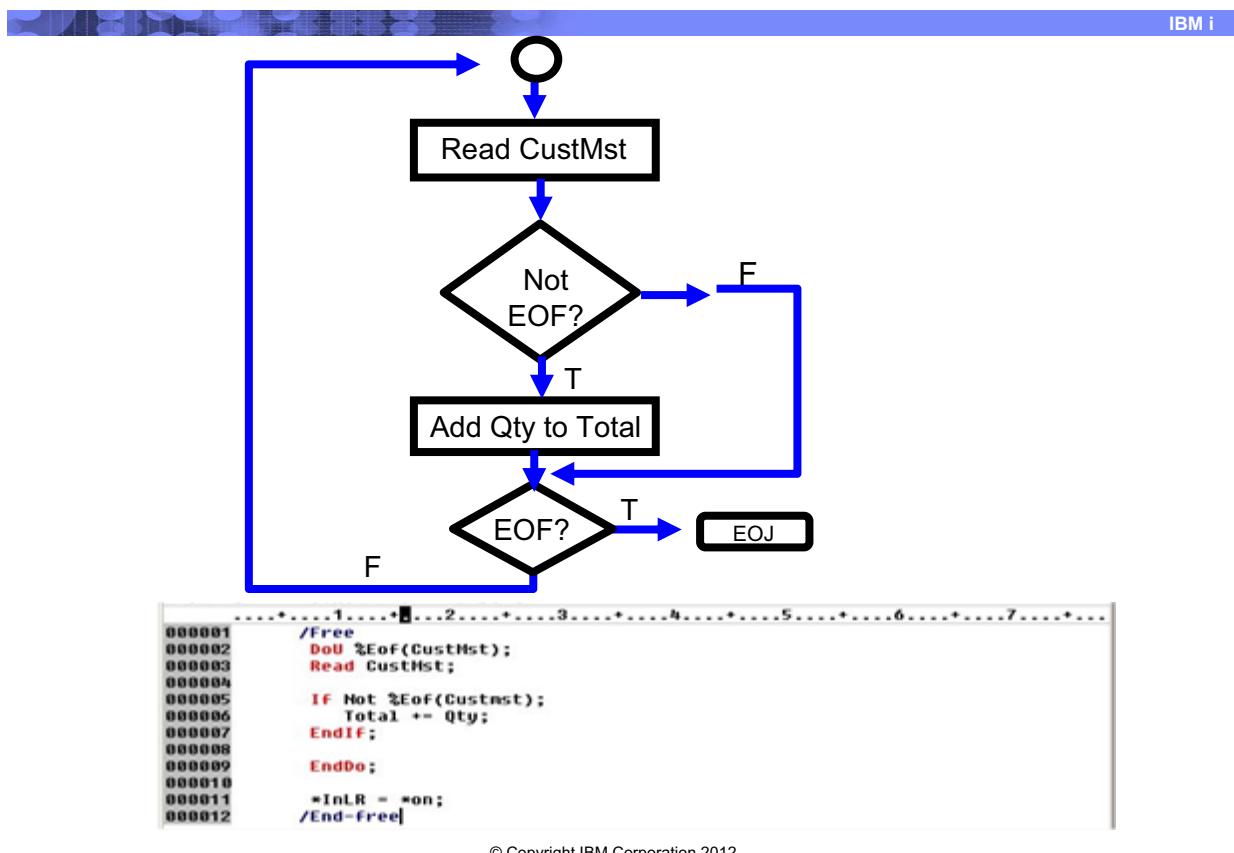


Figure 7-13. DoUntil

AS067.0

Notes:

The DoU operation code precedes a group of operations which you want to execute at least once and possibly more than once. An associated EndDo statement marks the end of the group. A logical condition is expressed by the %Eof BiF. The operations controlled by the DoU loop are performed until the %Eof expression is true.

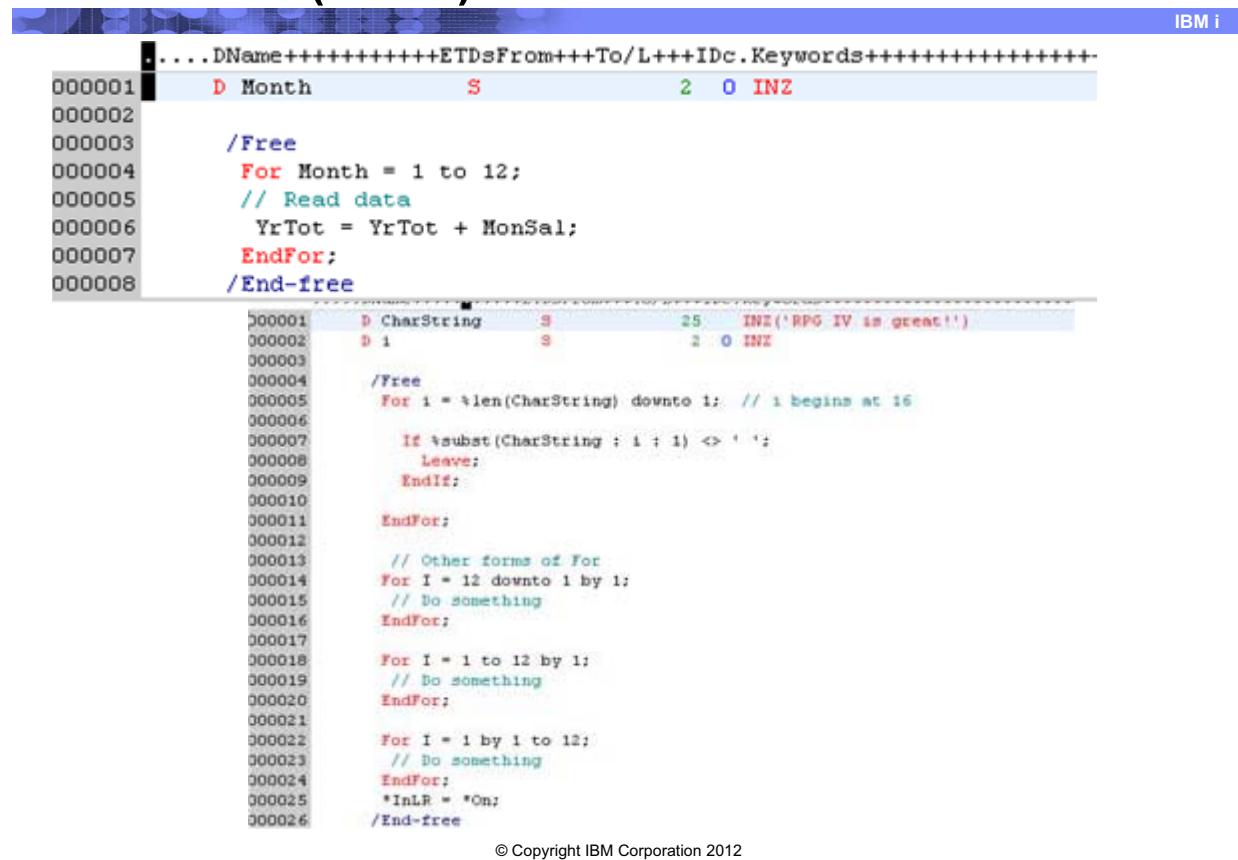
The condition is tested after the DoU group is performed.

In this example, the DO group is executed until we reach end of file of `CustMast`.

Why is the IF statement coded?

The reason we need the IF is because DoU checks the condition at end of the loop. Therefore, at least one iteration is always performed. An empty file condition would not be detected without the IF (or too many iterations would be performed).

For/EndFor (1 of 2)



The screenshot shows an IBM i terminal window with the title bar "IBM i". The code editor displays several examples of RPG IV programming, specifically focusing on the For/EndFor loop construct.

```

....DName++++++ETDsFrom+++To/L+++IDc.Keywords+++++-
000001 D Month      S          2  O INZ
000002
000003 /Free
000004   For Month = 1 to 12;
000005     // Read data
000006     YrTot = YrTot + MonSal;
000007   EndFor;
000008 /End-free

000001 D CharString  S          25  O INZ('RPG IV is great!')
000002 D i            S          2  O INZ
000003
000004 /Free
000005   For i = %len(CharString) downto 1; // i begins at 16
000006     If %subst(CharString : i + 1) <> ' '
000007       Leave;
000008     EndIf;
000009
000010   EndFor;
000011
000012 // Other forms of For
000013 For I = 12 downto 1 by 1;
000014   // Do something
000015 EndFor;
000016
000017 For I = 1 to 12 by 1;
000018   // Do something
000019 EndFor;
000020
000021 For I = 1 by 1 to 12;
000022   // Do something
000023 EndFor;
000024 *InLR = *On;
000025
000026 /End-free

```

© Copyright IBM Corporation 2012

Figure 7-14. For/EndFor (1 of 2)

AS067.0

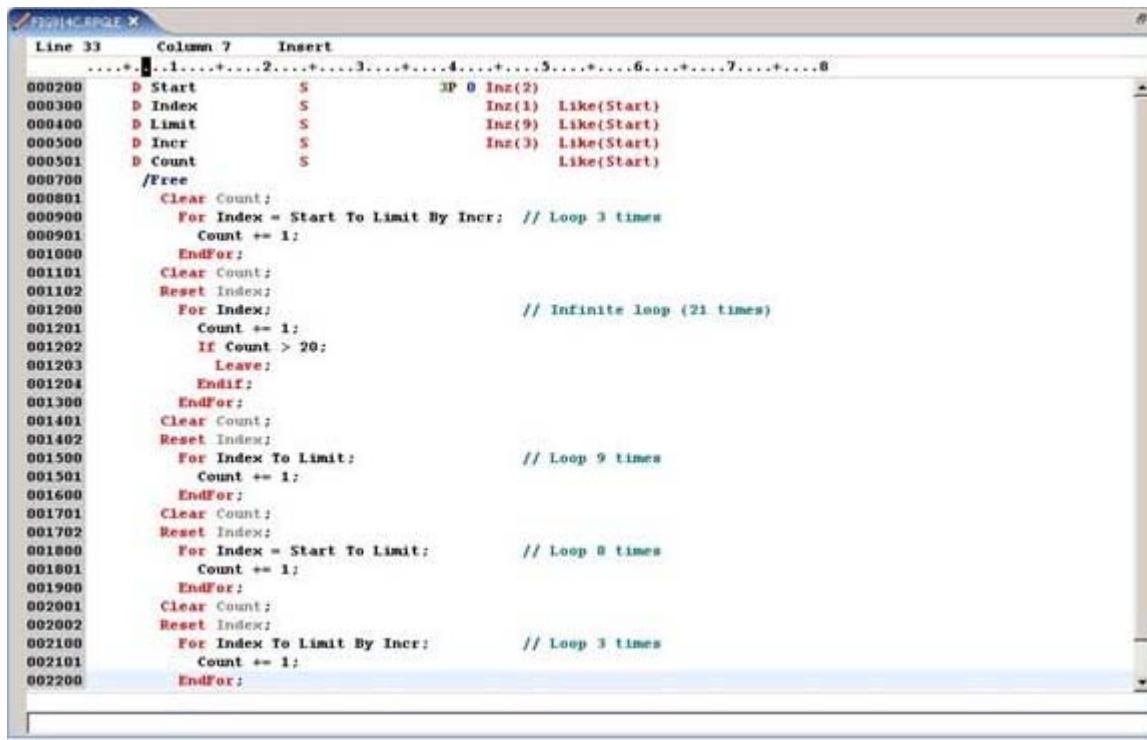
Notes:

For and EndFor is another form of looping. The For / EndFor loop offers you flexibility in how you control the loop.

The visual shows various forms of For. You can increment or decrement a variable by an integer amount.

For/EndFor (2 of 2)

IBM i



```

Line 33    Column 7    Insert
.....+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
000200  D Start      S          IP 0 Inv(2)
000300  D Index       S          Inv(1) Like(Start)
000400  D Limit       S          Inv(9) Like(Start)
000500  D Incr        S          Inv(3) Like(Start)
000501  D Count       S          Like(Start)
000700  /Free
000801  Clear Count;
000900  For Index = Start To Limit By Incr; // Loop 3 times
000901  Count += 1;
001000  EndFor;
001101  Clear Count;
001102  Reset Index;
001200  For Index;           // Infinite loop (21 times)
001201  Count += 1;
001202  If Count > 20;
001203  Leave;
001204  Endif;
001300  EndFor;
001401  Clear Count;
001402  Reset Index;
001500  For Index To Limit; // Loop 9 times
001501  Count += 1;
001600  EndFor;
001701  Clear Count;
001702  Reset Index;
001800  For Index = Start To Limit; // Loop 8 times
001801  Count += 1;
001900  EndFor;
002001  Clear Count;
002002  Reset Index;
002100  For Index To Limit By Incr; // Loop 3 times
002101  Count += 1;
002200  EndFor;

```

© Copyright IBM Corporation 2012

Figure 7-15. For/EndFor (2 of 2)

AS067.0

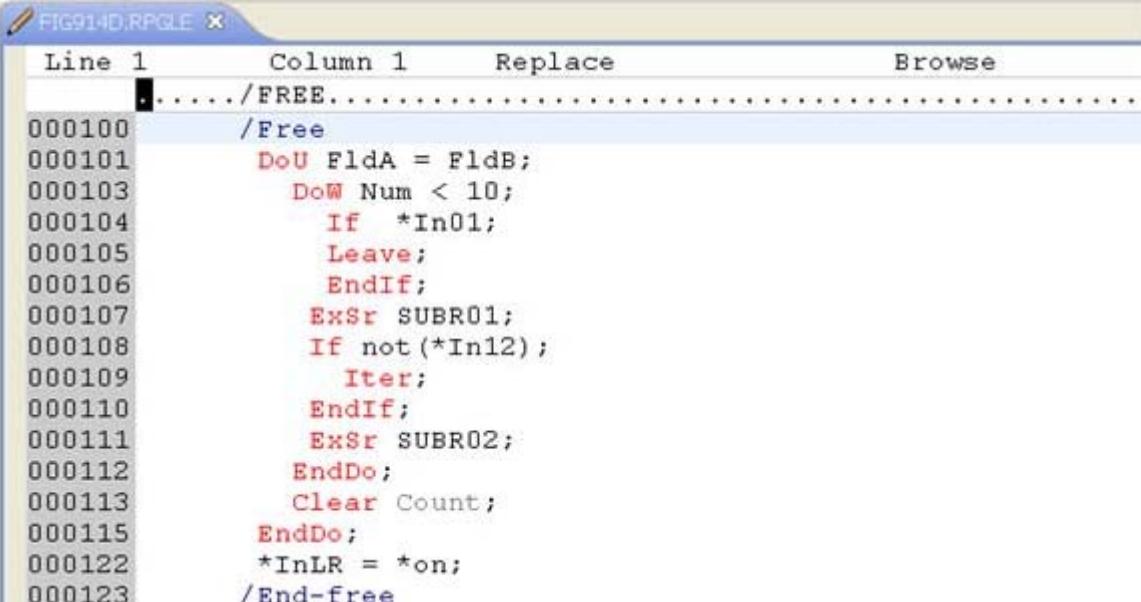
Notes:

The visual illustrates how the For/Endfor operates under different circumstances:

- If no Start value is specified, the Index retains its original value.
- If no Increment is specified, it defaults to 1.
- If no Limit value is specified, an indefinite loop results. (This relies upon some other condition within the loop to terminate.)

LEAVE and ITER

IBM i



```

FIG914D.RPQLX X
Line 1      Column 1      Replace      Browse
...../FREE..... .
000100      /Free
000101      DoU FldA = FldB;
000103          DoW Num < 10;
000104              If *In01;
000105                  Leave;
000106                  EndIf;
000107                  ExSr SUBR01;
000108                  If not (*In12);
000109                      Iter;
000110                      EndIf;
000111                      ExSr SUBR02;
000112                      EndDo;
000113                      Clear Count;
000115                  EndDo;
000122                  *InLR = *on;
000123              /End-free

```

© Copyright IBM Corporation 2012

Figure 7-16. LEAVE and ITER

AS067.0

Notes:

The visual illustrates how the LEAVE/ITER operation codes are used to control loop processing:

- A DoU loop contains a DoW loop within it.
- An IF statement checks indicator 01. If it is ON, the LEAVE operation executes.
- Control transfers *out of* the innermost DoW loop to the Clear instruction, avoiding SUBR01 and SUBR02.
- Alternately, if indicator 01 is not on, SUBR01 is processed and indicator 12 is checked.
- If indicator 12 is off, the ITER operation transfers control to the innermost EndDo of the DoW loop and the loop is evaluated once more.

Subroutines

IBM i

- Modular way of writing your code
- Placed at end of calculations
- Local: Can be executed only within program in which they are located
- Calculations that are used repeatedly in your program
- Can code as separate source member and /COPY into many programs

© Copyright IBM Corporation 2012

Figure 7-17. Subroutines

AS067.0

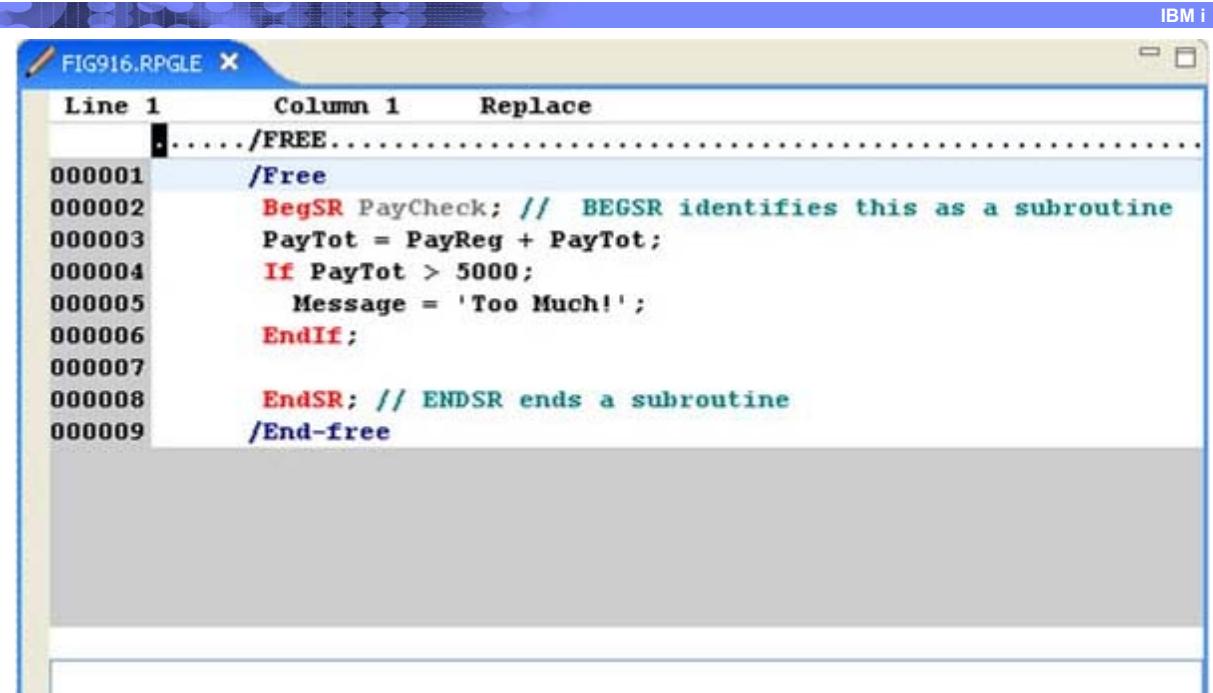
Notes:

Using subroutines allow you to organize your code in a more readable as well as a modular fashion.

We cover how to code a subroutine and then look at the ways they can be executed from within an RPG IV program using the EXSR opcode.

Subroutines are *local* to the program in which they are coded, and they cannot be called from outside that program. If the code has universal application, the subroutine could be coded as an individual source member, and the /COPY directive should be used to include a copy of the subroutine in any program that requires it.

Coding subroutines



```

Line 1      Column 1      Replace
...../FREE.....  

000001      /Free  

000002      BegSR PayCheck; // BEGSR identifies this as a subroutine  

000003      PayTot = PayReg + PayTot;  

000004      If PayTot > 5000;  

000005          Message = 'Too Much!';  

000006      EndIf;  

000007  

000008      EndSR; // ENDSR ends a subroutine  

000009      /End-free

```

© Copyright IBM Corporation 2012

Figure 7-18. Coding subroutines

AS067.0

Notes:

Let us assume that we have some calculations that are used many times within a program.

First, we need a way to name this code so that we can execute it from within the program. Therefore, we use the label **PayCheck** to uniquely identify the start of this subroutine.

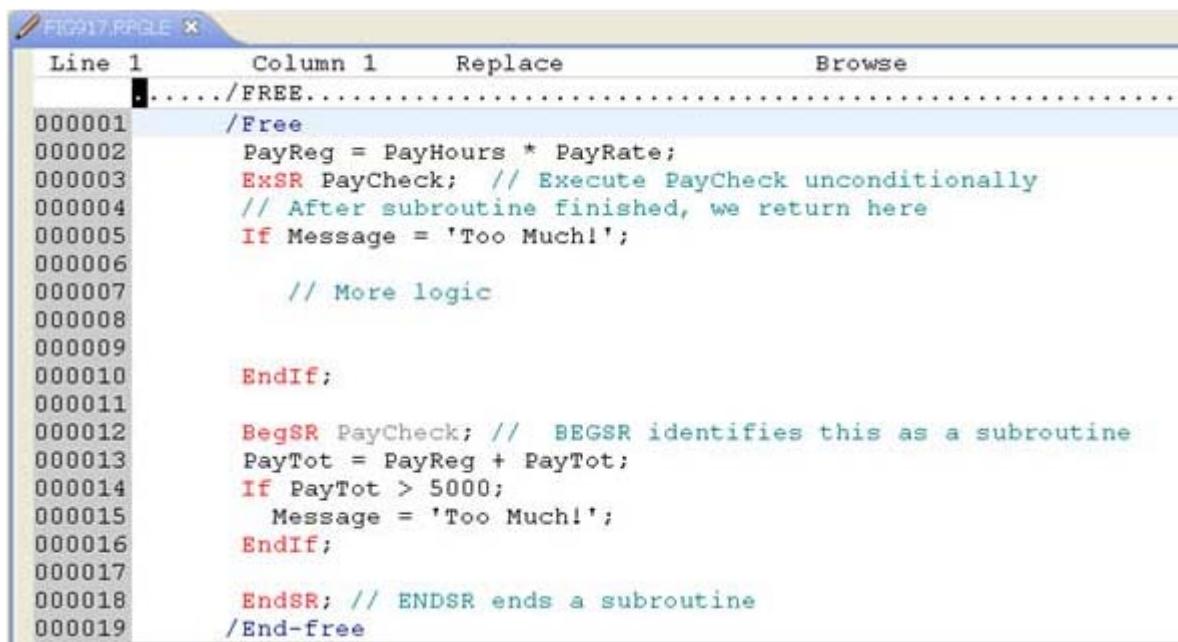
A subroutine is a set of RPG calculations enclosed within BEGSR and ENDSR operation codes. On the BEGSR statement, you must code a subroutine name to uniquely identify this subroutine within this particular program. You may code as many subroutines as you want in a program.

Subroutines require you to organize code that is repeatedly executed in your program at the end of your calculations. Subroutines force you to organize your code in a structured fashion enhancing its readability.

All subroutines within your program must be grouped together at the end of your calculations. You can code only one subroutine between each set of BEGSR and ENDSR opcodes. Subroutines cannot be coded in a nested fashion. However, one subroutine can issue an ExSR to execute another subroutine.

Unconditional execution of a subroutine

IBM i



The screenshot shows a window titled 'FIG017.RPGLE' containing RPGLE source code. The code is as follows:

```

Line 1      Column 1      Replace      Browse
...../FREE.....  

000001      /Free  

000002          PayReg = PayHours * PayRate;  

000003          EXSR PayCheck; // Execute PayCheck unconditionally  

000004          // After subroutine finished, we return here  

000005          If Message = 'Too Much!';  

000006  

000007          // More logic  

000008  

000009  

000010         EndIf;  

000011  

000012         BegSR PayCheck; // BEGSR identifies this as a subroutine  

000013         PayTot = PayReg + PayTot;  

000014         If PayTot > 5000;  

000015             Message = 'Too Much!';  

000016         EndIf;  

000017  

000018         EndSR; // ENDSR ends a subroutine  

000019     /End-free

```

© Copyright IBM Corporation 2012

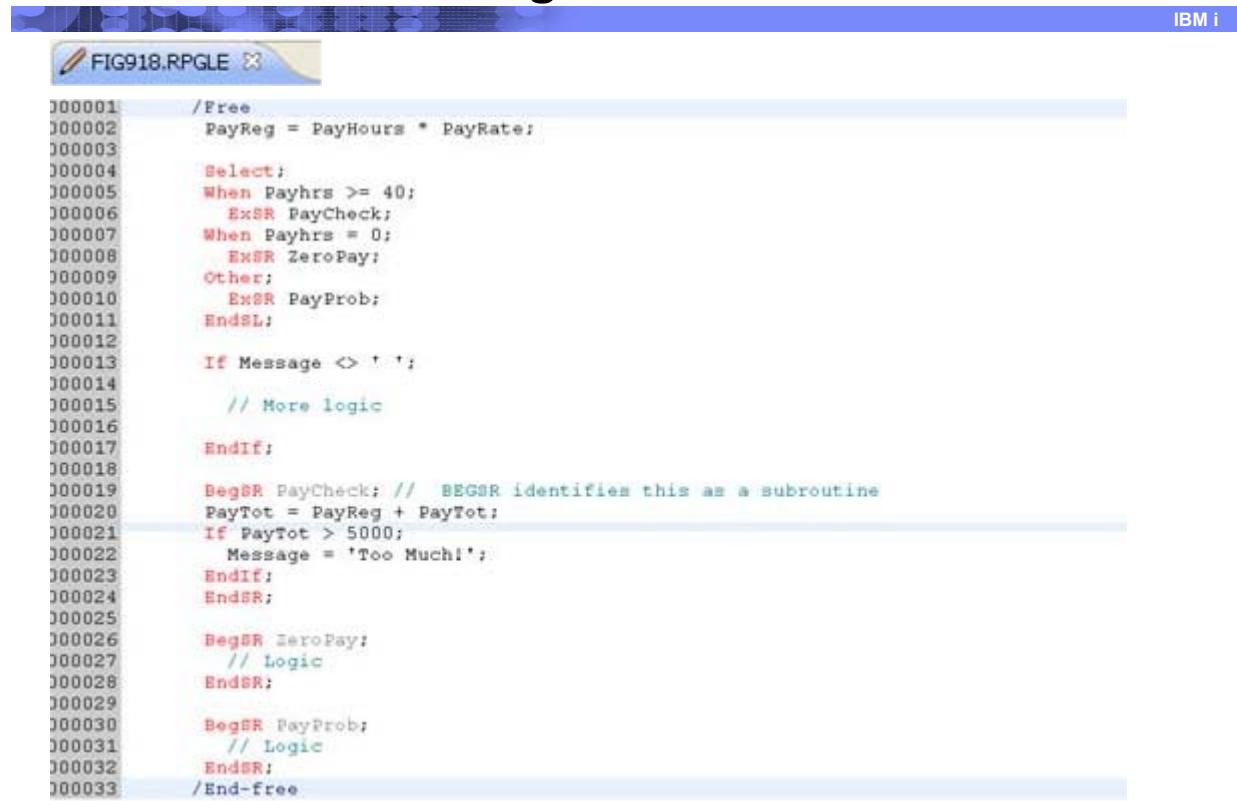
Figure 7-19. Unconditional execution of a subroutine

AS067.0

Notes:

The EXSR opcode causes a branch directly to the named subroutine. Upon completion of the subroutine, your program returns to the calculation specification immediately following the EXSR opcode.

Conditional branching to a subroutine



```

FIG918.RPGLE X
000001 /Free
000002     PayReg = PayHours * PayRate;
000003
000004     Select;
000005     When Payhrs >= 40;
000006         ExSR PayCheck;
000007     When Payhrs = 0;
000008         ExSR ZeroPay;
000009     Other;
000010         ExSR PayProb;
000011     EndSL;
000012
000013     If Message <> ' ';
000014         // More logic
000015
000016     EndIf;
000017
000018     BegSR PayCheck; // BEGSR identifies this as a subroutine
000019     PayTot = PayReg + PayTot;
000020     If PayTot > 5000;
000021         Message = 'Too Much!';
000022     EndIf;
000023     EndSR;
000024
000025     BegSR ZeroPay;
000026     // Logic
000027     EndSR;
000028
000029     BegSR PayProb;
000030     // Logic
000031     EndSR;
000032
000033 /End-free

```

© Copyright IBM Corporation 2012

Figure 7-20. Conditional branching to a subroutine

AS067.0

Notes:

Whereas the EXSR is an unconditional branch to the subroutine, using a SELECT group can condition execution. If a conditional WHEN test is satisfied, execute the subroutine and return and execute the next line of code following the EndSL. If the test fails, simply attempt the next test, until a condition is satisfied or the EndSL is reached. The Other in the visual is executed as a catch-all if all previous When operations were not executed.

Implicit execution of a subroutine

IBM i

```

000001      /Free
000002      PayReg = PayHours * PayRate;
000003
000004      Select;
000005      When Payhrs >= 40;
000006          ExSR PayCheck;
000007      When Payhrs = 0;
000008          ExSR ZeroPay;
000009      Other;
000010          ExSR PayProb;
000011      EndSL;
000012
000013      IF Message <> ' ';
000014
000015          // More logic
000016
000017      EndIF;
000017-----16 data records excluded -----
000034      BegSR *INZSR;
000035          Char1 = 'Name';
000036          Char2 = 'Address';
000037          Numeric1 = 1;
000038          Numeric2 = 2;
000039      EndSR;
000040  /End-free

```

© Copyright IBM Corporation 2012

Figure 7-21. Implicit execution of a subroutine

AS067.0

Notes:

Subroutines can also be executed implicitly if they are identified as *special* routines, such as *INZSR (initialization subroutine).

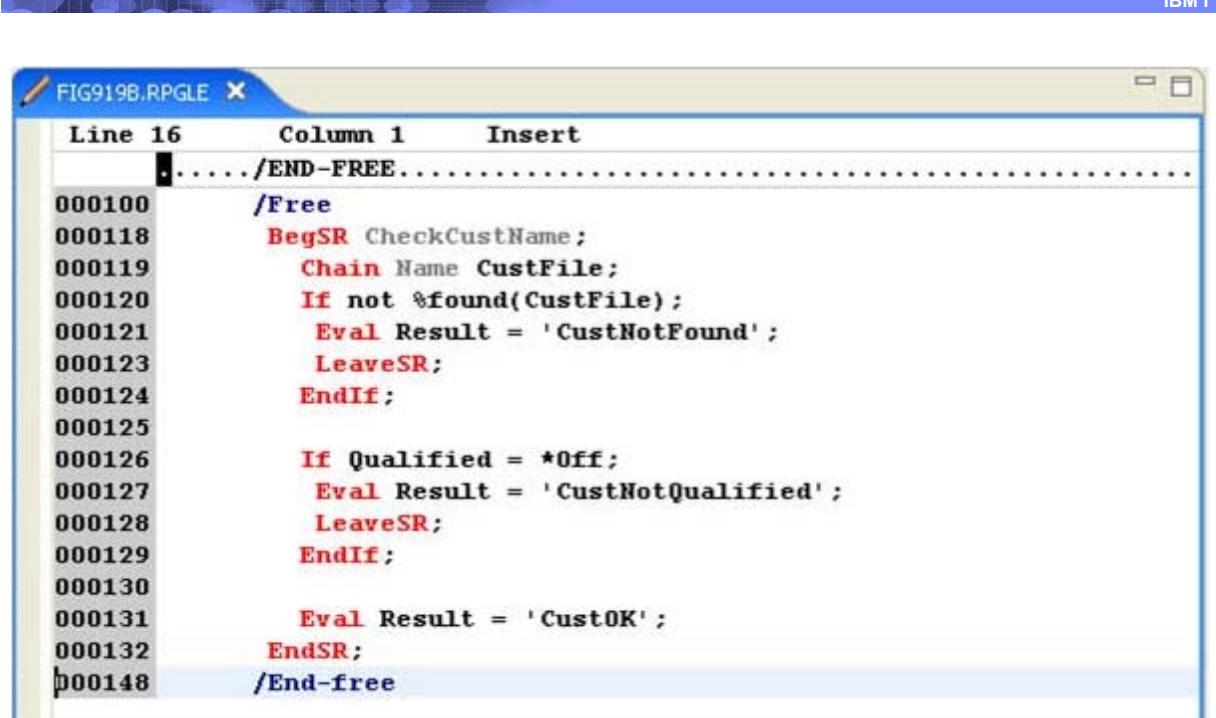
The initialization subroutine allows you to process the calculations in the *INZSR before you do anything else in your program. A specific subroutine that is to be run at program initialization time can be defined by specifying *INZSR as the name of the subroutine beside the BEGSR operation. Only one subroutine can be defined as an initialization subroutine. It is called at the end of the program initialization step of the program cycle. Briefly, this includes initializing fields, opening files, and so forth.

It is not necessary to code the EXSR for the *INZSR subroutine as it is executed by the system automatically (only once). You may execute this subroutine from within your code by issuing an EXSR.

Reference:

ILE RPG Language Reference, Program Cycle

LEAVESR



The screenshot shows an IBM i terminal window titled 'FIG919B.RPGLE X'. The code is displayed in a text editor with syntax highlighting. The code uses line numbers from 000100 to 000148. It starts with a comment '/END-FREE' followed by a series of conditional blocks ('If', 'Else If', 'Else') that check for various conditions like 'CustNotFound', 'CustNotQualified', and 'CustOK'. Each condition leads to a 'LeaveSR' operation. The code ends with an 'EndSR' and '/End-free'.

```

Line 16      Column 1      Insert
...../END-FREE.....
000100      /Free
000118      BegSR CheckCustName;
000119          Chain Name CustFile;
000120          If not %found(CustFile);
000121              Eval Result = 'CustNotFound';
000122                  LeaveSR;
000123              EndIf;
000124
000125
000126          If Qualified = *Off;
000127              Eval Result = 'CustNotQualified';
000128                  LeaveSR;
000129              EndIf;
000130
000131          Eval Result = 'CustOK';
000132      EndSR;
000148      /End-free

```

© Copyright IBM Corporation 2012

Figure 7-22. LEAVESR

AS067.0

Notes:

The LEAVESR operation exits a subroutine from any point within the subroutine. Control passes to the ENDSR operation for the subroutine. LEAVESR is allowed only from within a subroutine.

Reference:

ILE RPG Language Reference, Operation Codes

Calls in RPG IV

IBM i

- Why call other programs?
 - Execute logic not included in your program
 - Write once, use many times
- How?
 - Dynamic CALL:
 - CALL opcode: Legacy method
 - CALLP opcode: Uses prototyping to reduce errors
 - Static CALL:
 - 'Bound Call'
 - Less overhead at run-time
 - Use CALLP or CALLB opcode

© Copyright IBM Corporation 2012

Figure 7-23. Calls in RPG IV

AS067.0

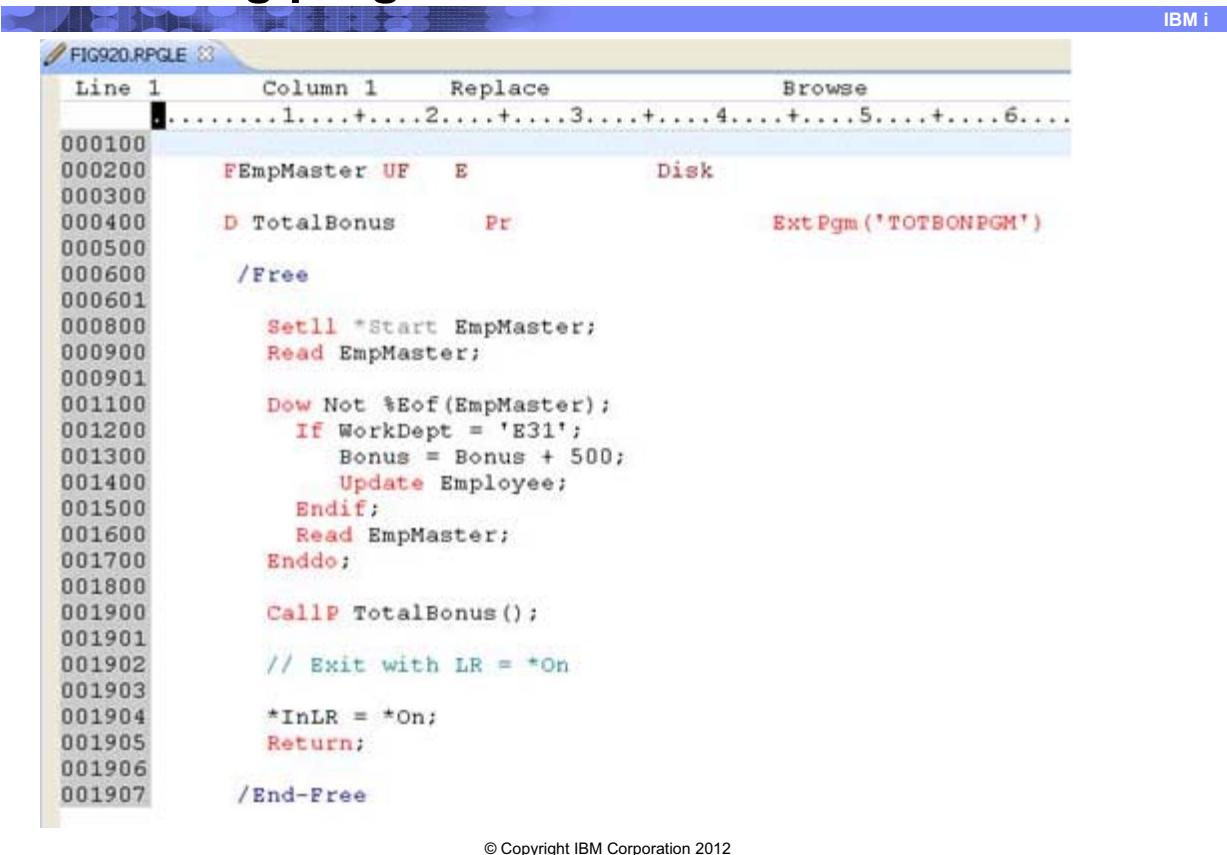
Notes:

RPG IV can call other RPG IV programs as well as programs written in other languages such as CL, COBOL, and C.

RPG IV gives you three different opcodes to call other programs (or ILE Modules).

We briefly discuss the prototyped CALLP opcode in this topic.

The calling program



The screenshot shows an IBM i terminal window with the title 'FIG920.RPGLE' and the sub-title 'IBM i'. The window displays RPGLE code for a program named 'FIG920'. The code includes declarations for a file 'FEmpMaster' with parameters 'UF' and 'E', and a prototype 'TotalBonus' with parameter 'Pr'. The main processing loop reads from 'FEmpMaster', updates a variable 'Bonus' by adding 500 if the 'WorkDept' field is 'E31', and then calls the 'TotalBonus' prototype. Finally, it exits with a return code of *On.

```

Line 1      Column 1      Replace      Browse
.....1....+....2....+....3....+....4....+....5....+....6....
000100
000200      FEmpMaster UF   E           Disk
000300
000400      D TotalBonus     Pr          Ext Pgm ('TOTBONPGM')
000500
000600      /Free
000601
000800      Setll *Start EmpMaster;
000900      Read EmpMaster;
000901
001100      Dow Not %Eof(EmpMaster);
001200          If WorkDept = 'E31';
001300              Bonus = Bonus + 500;
001400              Update Employee;
001500          Endif;
001600          Read EmpMaster;
001700      Enddo;
001800
001900      CallP TotalBonus();
001901
001902          // Exit with LR = *On
001903
001904          *InLR = *On;
001905          Return;
001906
001907      /End-Free

```

© Copyright IBM Corporation 2012

Figure 7-24. The calling program

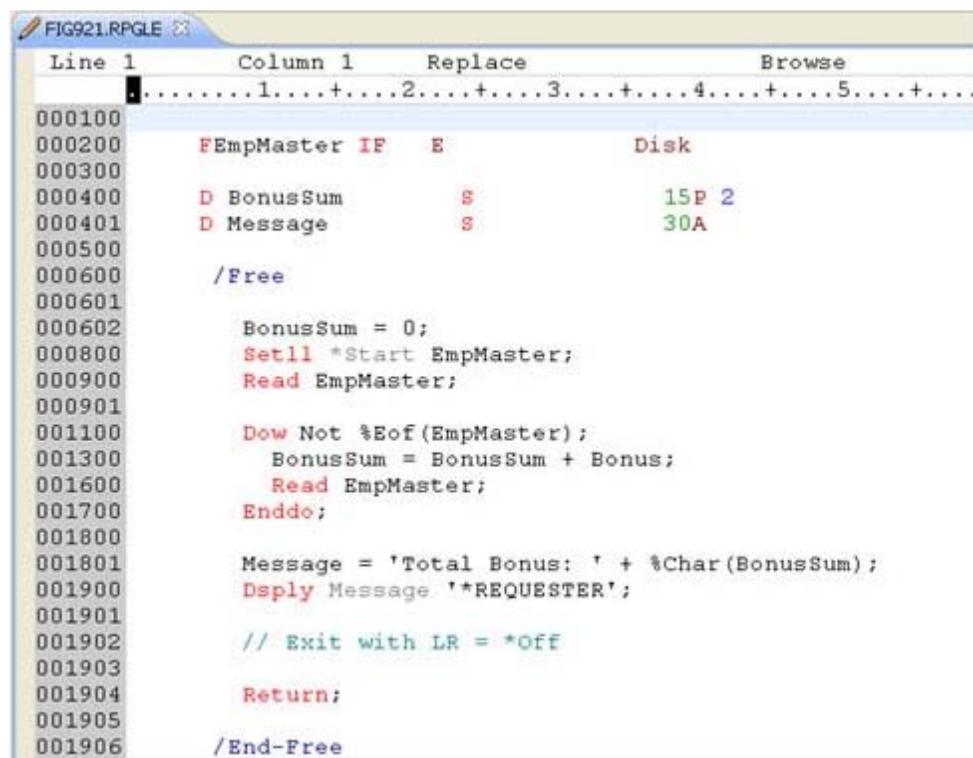
AS067.0

Notes:

1. The calling program contains a prototype (PR) for the called program. We see the called program next.
2. Notice in the call, the CALLP (call prototype) opcode is used. You must call a program (the name of the PR). Notice that the name of the PR might be different than the actual name of the program (EXTPGM keyword). This is handy when you call system routines or functions that sometimes are not named the way you might want them to be named.

The called program: Return to caller

IBM i



```

FIG921.RPGLE
Line 1      Column 1      Replace      Browse
. ....1....+....2....+....3....+....4....+....5....+....
000100
000200      FEmpMaster IF   E           Disk
000300
000400      D BonusSum      S           15P 2
000401      D Message       S           30A
000500
000600      /Free
000601
000602      BonusSum = 0;
000800      Setl1 *Start EmpMaster;
000900      Read EmpMaster;
000901
001100      Dow Not %Eof(EmpMaster);
001300      BonusSum = BonusSum + Bonus;
001600      Read EmpMaster;
001700      Enddo;
001800
001801      Message = 'Total Bonus: ' + %Char(BonusSum);
001900      Disp Message '*REQUESTER';
001901
001902      // Exit with LR = *OFF
001903
001904      Return;
001905
001906      /End-Free

```

© Copyright IBM Corporation 2012

Figure 7-25. The called program: Return to caller

AS067.0

Notes:

Because you can call a program repeatedly, you might not want to set on LR after the called program has completed. There is a way to leave the called program active.

When you return to the caller without having turned on LR, the called program remains active. This is good if you intend to call the program repeatedly. You do not pay the price of initializing the called program more than once. This improves call performance.

The RETURN opcode returns control to the caller. In the example above, we have returned to the caller without setting LR on.

Setting *INLR

IBM i

Tidy program exit

- End of calculations and LR set *ON
- RETURN with LR set *ON:
 - Variables cleared
 - Files closed
 - Slower program activation

Untidy program exit

- RETURN with LR set *OFF:
 - Variables left intact
 - Files left open
 - Re-entrant program coding required
 - Faster program activation

© Copyright IBM Corporation 2012

Figure 7-26. Setting *INLR

AS067.0

Notes:

We have seen how one RPG IV program can call another. We have also seen two methods for ending a program and returning to the caller.

This visual summarizes the two methods for ending a program.

On a subsequent course we explore Program and Procedure calls in much more detail. Particular consideration needs to be given to passing information between programs. There are many techniques for achieving this. The most common way of passing information is through the use of parameters.

Machine exercise: Coding subroutines

IBM i



© Copyright IBM Corporation 2012

Figure 7-27. Machine exercise: Coding subroutines

AS067.0

Notes:

Perform the coding subroutines machine exercise.

Checkpoint

IBM i

1. True or False: The parameter list in both the calling and the called programs must match field name by field name.

2. Which of the following opcodes does not have a corresponding ENDxx statement to terminate the group?
 - a. DO
 - b. IF
 - c. PLIST
 - d. BEGSR
 - e. SELECT

3. True or False: An RPG subroutine can be called from another program outside of where it is written by coding an EXSR.

© Copyright IBM Corporation 2012

Figure 7-28. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Write structured RPG IV code
- Write RPG IV code using conditional IF, DOW, DOU, and FOR opcodes
- Write RPG IV code using conditional SELECT groups
- For a nested group, determine the number of nesting levels
- Code a subroutine in an RPG IV program

© Copyright IBM Corporation 2012

Figure 7-29. Unit summary

AS067.0

Notes:

Unit 8. Accessing the DB2 database using RPG IV

What this unit is about

This unit describes the two types of database files on the i (Power i). It briefly reviews their creation and discuss how they are accessed in RPG IV programs. This unit also discusses the format and function of RPG IV file handling operation codes, including Read, Write, Chain, Update, and Delete.

What you should be able to do

After completing this unit, you should be able to:

- Define physical and logical files in an RPG IV program
- Access data in physical and logical files using Read and Chain
- Modify data in physical and logical files using Write, Update, and Delete
- Use the sequential and random access methods to process data
- Define record keys and use them to process data

How you will check your progress

- Checkpoint questions
- Machine exercises

Given a problem statement, the students code an RPG IV program that accesses records in the database using I/O operation codes.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Define physical and logical files in an RPG IV program
- Access data in physical and logical files using Read and Chain
- Modify data in physical and logical files using Write, Update, and Delete
- Use the sequential and random access methods to process data
- Define record keys and use them to process data

© Copyright IBM Corporation 2012

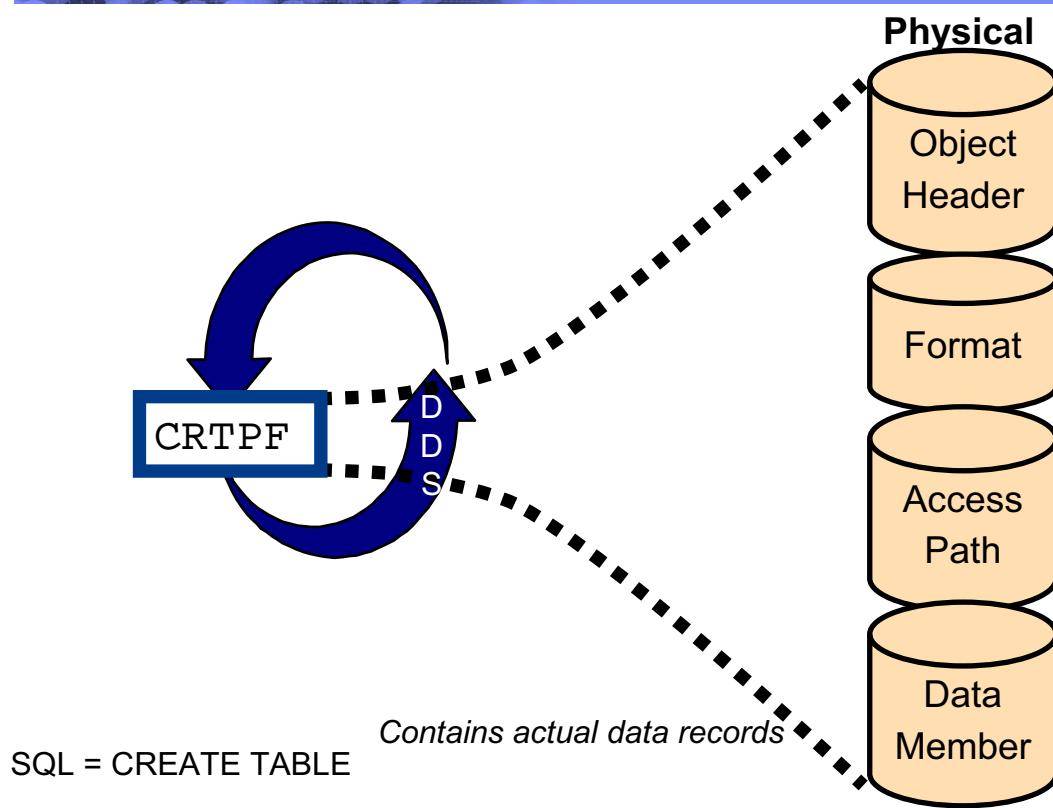
Figure 8-1. Unit objectives

AS067.0

Notes:

DB file: Physical

IBM i



© Copyright IBM Corporation 2012

Figure 8-2. DB file: Physical

AS067.0

Notes:

A **physical file** is the i object type used to hold data records. As a rule, physical files are composed of four parts:

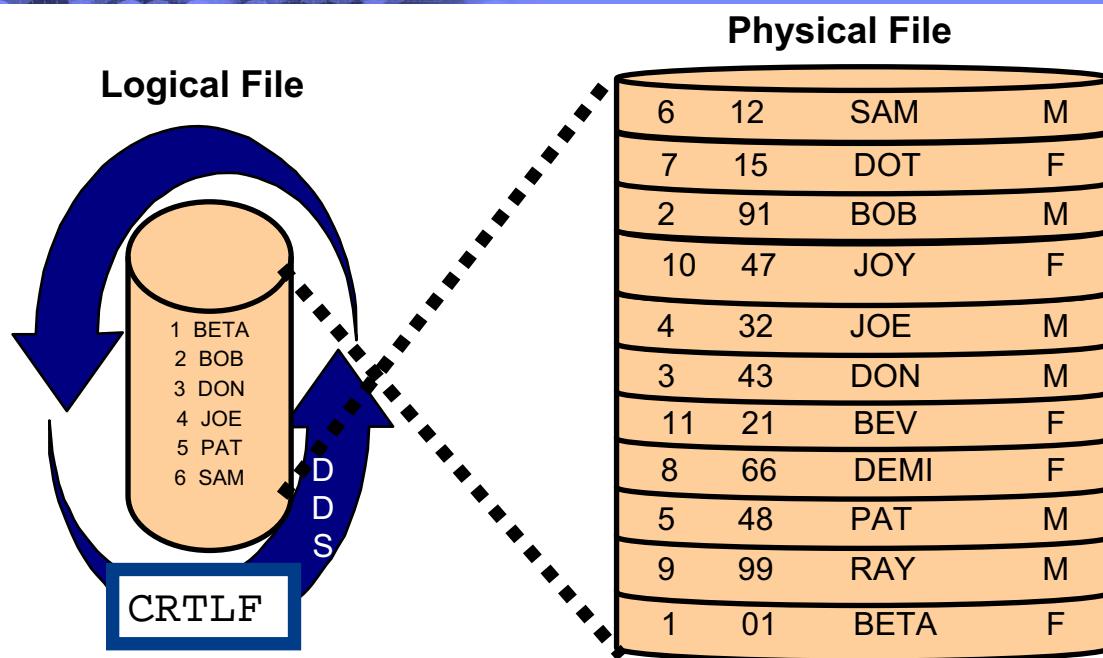
- Object header
- Format
- Access path
- Member

A physical file actually stores data records. Using DDS record and field level specifications, you can define a physical file. At the field level, if one or more fields is designated as the key, you can access records in the file in either key sequence or arrival sequence. If key fields are not designated, your access can only be in arrival sequence. The access path is what enables record retrieval using a key. The access path is the index for the file to all of its data records.

The CREATE TABLE SQL statement does the same job as CRTPF.

DB file: Logical

IBM i



Determines how records are selected and their order when accessed

SQL = CREATE VIEW or
CREATE INDEX

© Copyright IBM Corporation 2012

Figure 8-3. DB file: Logical

AS067.0

Notes:

Logical files appear to have data records, but they do not actually contain data records. Instead, an access path, or pointers to records in the physical files, are stored in logical files.

A logical file is always based upon one or more physical files. The logical file stores an access path (no data) in its member. It is with logical files on the i that a programmer might define views of data that are different from the definition in the physical file.

The traditional relational database operators are available. Record sequencing, record selection, field projection, record union and join are all available tools for processing data from a physical file.

Sequence: Ordering records based upon one or more fields.

Selection: Using criteria to include or omit selected records.

Projection: Choosing which fields are used in a view of the data.

Union: Taking records from two or more files and presenting them in a view as though they were coming from one file.

Join: Picking fields from two or more records from different files and presenting them in a single record format.

Although data resides in the physical file, programs can use logical or physical files in order to manipulate the data.

The CREATE VIEW and CREATE INDEX SQL statements also create logical files.

DB file descriptions

- Define record formats for physical and logical files:
 - Single record format for physical files
 - One or more record formats for logical files
- Code DDS at levels below (shown highest to lowest):
 - Physical file:
 - File
 - Record
 - Field
 - Key
 - Logical file:
 - File
 - Record
 - Join
 - Field
 - Key
 - Selection

© Copyright IBM Corporation 2012

Figure 8-4. DB file descriptions

AS067.0

Notes:

DB files are defined using DDS. Keywords enhance your file definitions and can be entered at four levels for physical files and six levels for logical files.

Physical files have *one record format*.

Logical files have *one or more record formats*.

The file, record, and field levels for keywords work for database files just as they do for printer files. But you can define key fields for both physical and logical files. Also, you can specify record selection criteria for logical files only. And, you can define a logical file to join fields from different physical files together for presentation to a program as though they were in a single record.

There are two types of logical files. A non-join logical file provides a one-to-one relationship with the physical file on which it is based. Some advantages to using a non-join logical file in an RPG program:

- Alternate sequence (from physical file) of data records
- Program not burdened with record selection

- Efficiency; only necessary fields are part of record buffers

A join logical file allows data from more than one physical file to be presented in a single record format. Along with the above advantages, join logical files provide your RPG IV programs with the ability to access data from up to 32 different files by doing only ONE read. The alternative would be to read records from multiple physical files for the data. There are a few things to remember about join logical files:

- *Join logical files cannot be updated.* They are read only.
- One or more common fields must exist in the physical files that are being joined.
- The maximum number of physical files that can be joined is thirty two.



Note

Files can also be defined using SQL CREATE statements. The tables, views, and indexes that are created can be accessed as files in RPG IV programs.

DDS limits

IBM i

Description	DDS
Record length	32,766
Number of fields	8,000
Character field	32,766
Numeric field	63 Digits
Decimal positions	63 Digits
Key fields (composite)	120
Key length	2,000
File members	32,766

© Copyright IBM Corporation 2012

Figure 8-5. DDS limits

AS067.0

Notes:

These limits coincide with most of the limits of RPG IV.

Physical file DDS keywords

IBM i

●● ALTSEQ	COLHDG
●● CCSID	DFT
●● FCFO	REFFLD
●● FIFO	VALUES
●● LIFO	VARLEN
●● REF	●●● ABSVAL
●● UNIQUE	●●● DESCEND
● TEXT	●●● ZONE
CHECK	

blank - Field level

●●● - Key level

● - Record level

●● - File level

© Copyright IBM Corporation 2012

Figure 8-6. Physical file DDS keywords

AS067.0

Notes:

Refer to the DDS section of the IBM i Information Center for complete listings and details.

Reference:

DDS Reference: Physical and Logical Files, Keywords for Physical and Logical Files

Logical file DDS keywords

IBM i

●● ALTSEQ	●●●● JFLD	●●● VARLEN
●● DYNSLT	●●●● JOIN	●●● ABSVAL
●● FCFO	CMP	●●● DESCEND
●● FIFO	COLHDG	●●● ZONE
●● JDFTVAL	CONCAT	●●●● ALL
●● LIFO	JREF	●●●● CMP
●● REFACCPTH	RENAME	●●●● RANGE
●● UNIQUE	SST	●●●● VALUES
● JFILE	TEXT	
● PFILE	VALUES	

blank - Field level

●●● - Key level

● - Record level

●●●● - Join level

●● - File level

●●●●● - Select/Omit level

© Copyright IBM Corporation 2012

Figure 8-7. Logical file DDS keywords

AS067.0

Notes:

Refer to the DDS section of the IBM i Information Center for complete listings and details.

Reference:

DDS Reference: Physical and Logical Files, Keywords for Physical and Logical Files

DDS field reference file

IBM i

Line 1	Column 1	Replace	
000100	A*****	+A*.1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000200	A** Field Reference PF: DICTONARY		
000300	A*****		
000400	A R REFFMT		TEXT('Field Reference File')
000500	A*		
000600	A** Fields Used in Vendor Mastor File, VENDOR_PF		
000700	A*		
000800	A VNDNBR	5 0	TEXT('Vendor Number') COLHDG('Vend' 'Num')
000900	A VNDNAME	25	TEXT('Vendor Name') COLHDG('Vendor' 'Name')
001000	A VNDSTREET	25	TEXT('Vendor Street') COLHDG('Vendor Street')
001100	A VNDCITY	23	TEXT('Vendor City') COLHDG('Vendor City')
001200	A VNDSTATE	2	TEXT('Vendor State') COLHDG('Vnd' 'St')
001300	A VNDADDR3	25	TEXT('Address Line 3') COLHDG('Address Line 3')
001400	A VNDZIPCODE	5 0	TEXT('Zip Code') COLHDG('Zip' 'Code')
001500	A VNDAREACD	3 0	TEXT('Vendor Area Code') COLHDG('Vend' 'Area' 'Code')
001600	A VNDTELENO	7 0	TEXT('Vendor Telephone Number') COLHDG('Vendor' 'Tel' 'No')
001700	A VNDDISCPCT	3 3	TEXT('Discount % For Prompt Pymt')

© Copyright IBM Corporation 2012

Figure 8-8. DDS field reference file

AS067.0

Notes:

This is a subset of the DDS from the course exercises. All fields for our application are defined in the data dictionary, named **DICTONARY**, the field reference file. The physical file, **VENDOR_PF**, defines a file referencing these definitions. We reference those fields in the sample Physical file and the Logical file on the next visual.

DDS: PF/LF

```

000100 A-----Name-----RLen+TpB-----Functions-----
000200 A* Vendor master PF: VENDOR_PF
000300 A-----+
000400 A-----+
000500 A-----+
000600 A-----R VENDOR_FHT REF(DICTIONARY)
000700 A-----UNDMBR R UNIQUE
000800 A-----UNDNAME R TEXT('Vendor Master File Record')
000900 A-----UNDSTREET R
001000 A-----UND CITY R
001100 A-----UNDSTATE R
001200 A-----UNDZIPCODE R
001300 A-----UNDAREACD R
001400 A-----UNDTELNO R
001500 A-----UNDDISCPCTR
001600 A-----UNDUEDAYS R
001700 A-----UNDCLASS R
001800 A-----UNDACTIVE R
001900 A-----UND SALES R
002000 A-----UNDDISCHTDR
002100 A-----UNDDISCVTDR
002200 A-----UNDPRCHHTR
002300 A-----UNDPRCHYTR
002400 A-----UNDBALANCE R
002500 A-----UNDSERURTR
002600 A-----UNDDLURTG R
002700 A-----UNDCOMMENTR
002800 A-----K UNDMBR

-----+8*..1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+
000100 A-----+
000200 A* Vendors by Name LF: VNDNAM_LF
000300 A-----+
000400 A-----R VENDOR_FHT ALTSEQ(QSYSTRNTBL)
000500 A-----K UNDNAME PFILE(VENDOR_PF)
000600 A-----+

```

© Copyright IBM Corporation 2012

Figure 8-9. DDS: PF/LF

AS067.0

Notes:

The physical file, VENDOR_PF, references the field definitions in the field reference file, DICTIONARY.

The logical file, VNDNAM_LF, in the visual is simply a different view of the physical file. We are using the vendor name as the access path.

REFFLD, a field-level keyword, is used in the PF DDS when:

- You are basing field definition on fields from more than one file.
- The field name you are defining is based on a field in another file but differs in aspects such as length.

A key is *composite* when more than one field is used to define the key.

Creating DB files and entering data

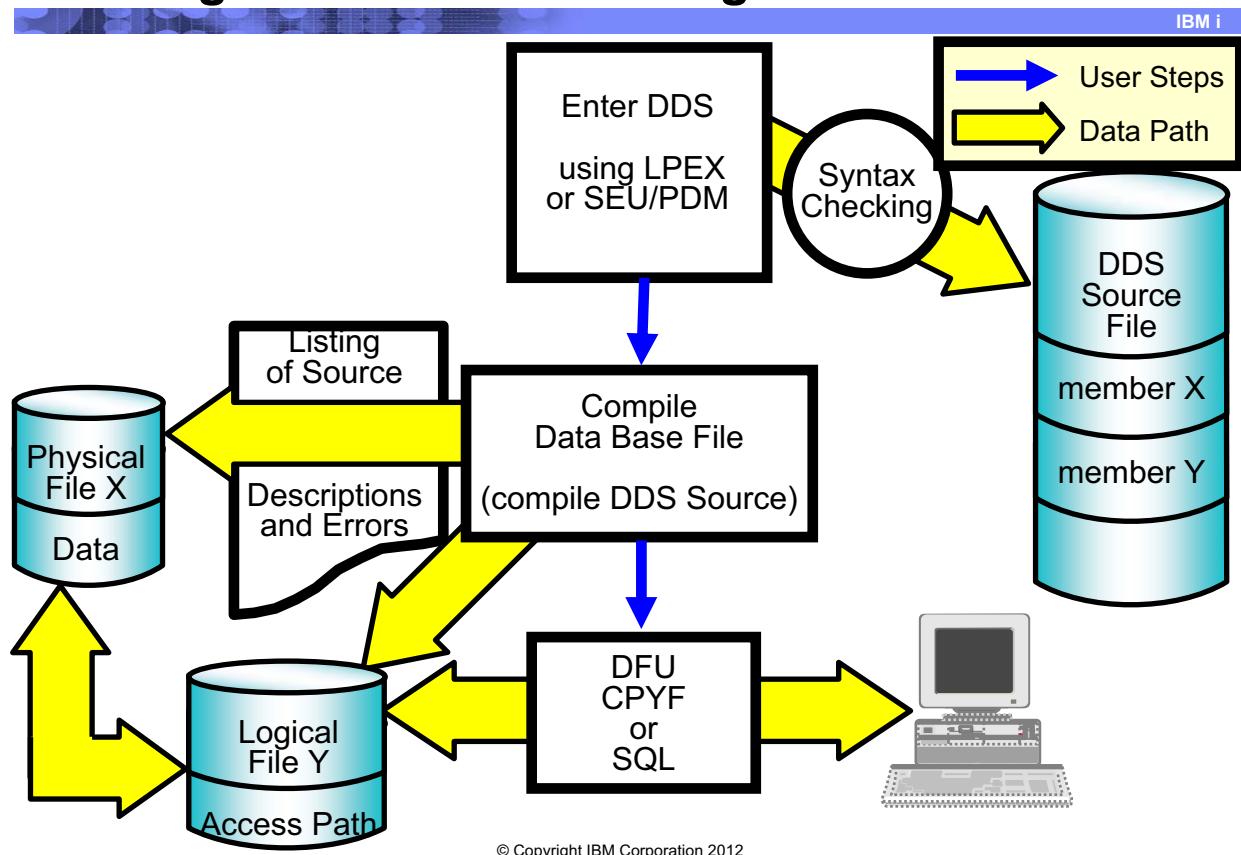


Figure 8-10. Creating DB files and entering data

AS067.0

Notes:

DDS source members for database files are specified as PF or LF for physical and logical files, respectively. When keying the source statements, the PF and LF prompts in SEU and LPEX provide the correct formats.

CRTPF and **CRTL**F are the CL commands that create the database file objects. There are many parameters that correspond to various physical and logical file attributes. Use the attributes appropriate to your needs.

Helpful commands

IBM i

CMD	Function
DSPFD	File definitions
DSPFFD	Field definitions
DSPPFM	Records in PF
RUNQRY	Records in PF
DSPDBR	File relationships
DSPPGMREF	Objects used by program
FNDSTRPDM	PDM string search
CPYF	Copies data records

© Copyright IBM Corporation 2012

Figure 8-11. Helpful commands

AS067.0

Notes:

The commands on this visual are helpful when exploring file definitions, relationships, content, and use.

DSPFD provides information retrieved from file descriptions for a given database file.

DSPFFD provides field-level information for one or more files.

DSPPFM displays the contents of a physical file's member.

RUNQRY runs a default query that displays in a formatted form the contents of a physical file's member.

DSPDBR provides relational information about database files. It identifies the logical files dependent on a specific file, files that use a specific record format, or the file members that are dependent on a specific file member.

DSPPGMREF provides a list of the system objects referred to by the specified programs. For RPG, *FILE, *DTAARA and *PGM are the system objects provided by this command.

FNDSTRPDM allows you to search for character or hexadecimal strings in source physical file members or data physical file members. You can use any PDM option or one of your own user-defined options on the member that contains a match for the string.

CPYF copies all or part of a file from the database or for an external device to the database or to an external device.

Another useful tool for examining data is the SQL SELECT statement. You can also examine the contents and characteristics of a file using Operations Navigator.

File operations

IBM i

- Input files:
 - Sequential:
 - READ
 - SETLL / SETGT
 - READE / READP / READPE
 - Random
 - CHAIN
 - Output files:
 - EXCEPT
 - WRITE
- Update files:
 - All Opcodes above
 - Plus:
 - DELETE
 - UPDATE
 - Build composite keys using::
 - KLIST/KFLD (V5R1 or earlier)
 - Pass key components as parameters (V5R2 or later)

© Copyright IBM Corporation 2012

Figure 8-12. File operations

AS067.0

Notes:

When processing database files with RPG IV programs, there are many choices that must be made concerning file access and record manipulation:

- What must your program do to the files and records in question?
- Are you adding new records to the file?
- Are you updating records currently in the file?
- Are you simply reading records for input or perhaps to print a report?

How you want to process a file determines how you declare it on the F-spec and what operation codes are used to read and change the data (if desired).

File type, position 17 of the file specification, designates how the file will be used by the program.

- **(I)nput** files contain data records that are to be read by the program.
- **(O)utput** files are designed for the program to write (new) records to the file.

- (**U**)pdate files can be used as input, can have records added and can additionally have records deleted and changed.

Sequential processing of records is generally used for batch processing. Reading typically starts with the first record in the file. With each subsequent read operation the next record in the file is read until the program reaches end of file.

Random processing allows direct retrieval of a specific record based upon the value of a key field or **RRN**. This allows your program to select directly a specific record for processing without having to process all the records before the one you want sequentially.

RPG IV offers several operation codes for accessing data from full procedural database files. Some of these operations are valid for sequential processing while others are used for random access processing.

Sequential access means that records are retrieved in either key order if the file has a key designated, or in arrival (first-in-first-out) sequence for non-keyed files.

Random access allows specific record retrieval based upon the value of a key field or relative record number. This kind of access enables you to retrieve the record you want, without any sequential processing.

Klist and Kfld should be used to describe a composite key in your RPG IV programs written at V5R1 or earlier. In V5R2, you can build a composite key in a parameter list that is the argument for the appropriate opcodes. We learn more later in this unit.

To use the **Write** operation with a file declared for update, you must also declare that you want to add records to the file.

References:

ILE RPG Language Reference, Operation Codes

File-related BiFs and extenders

- BiFs

— %Found(FileName)	Record Found?
— %Open(FileName)	File open?
— %Eof(FileName)	EOF (or BOF) Reached?
— %Equal(FileName)	Key match to search argument?
— %Error	Previous I/O failed?
— %KDS	Concatenated key data structure

- BiFs reference FileName NOT record format!

- Operation Extenders

- E is required with %Error

© Copyright IBM Corporation 2012

Figure 8-13. File-related BiFs and extenders

AS067.0

Notes:

We have been using some basic and let's hope obvious file operations in earlier units and exercises. Recent releases of RPG IV have been enhanced so that in almost all cases, it is unnecessary to use indicators to determine the answer to the typical questions that a program would ask following an I/O operation:

- **%Found(FileName)**

For a random read (CHAIN), a delete (DELETE) or cursor positioning operation, was a record found that satisfied the search condition defined by a key. %Found returns a '1' if search is successful and '0' if it is unsuccessful.

The FileName is expected as a parameter.

- **%Open(FileName)**

Is the FileName specified open? %Open returns a '1' if the file is open and a '0' if it is closed.

- **%Eof(FileName)**

This BiF checks whether the file specified has reached end of file for read (READ, READE, READP, READPE, READC) and write (WRITE) operations. For read operations that process records backwards %Eof checks for beginning of file (BOF). %Eof returns a '1' if either EOF or BOF is reached for the applicable operation. Otherwise, it is set to '0'.

- **%Equal(FileName)**

The %Equal BiF is used to check whether the previously executed cursor positioning operation for the specified filename resulted in an equal match to the search key.

- **%Error**

The %Error BiF can be used to determine whether the previously executed operation resulted in an error. This BiF replaces the error indicator that may be used with I/O opcodes. It is used in conjunction with the **E** opcode extender.

- **%Kds**

The %Kds BiF enables you to construct a composite key using a data structure.

File BiFs always reference the filename even if the I/O opcode references a record format.

File open and close: Explicit or implicit?

IBM i

IMPLICIT versus EXPLICIT
RPG versus PROGRAMMER



Explicit (perhaps) when:

- Using file once
- Many files; reduce program initialization
- Inquiry program to access one of many files

© Copyright IBM Corporation 2012

Figure 8-14. File open and close: Explicit or implicit?

AS067.0

Notes:

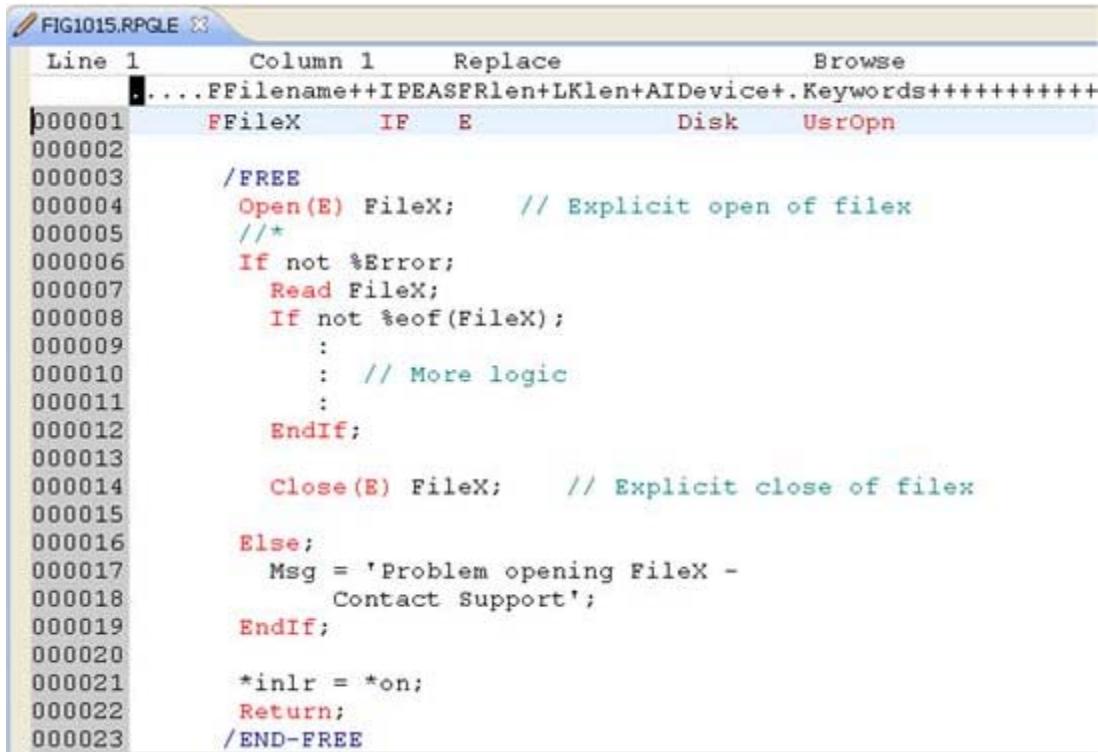
RPG IV, by default, automatically opens all files at program initiation and closes them at program termination. A programmer can choose to override this automatic process and elect to EXPLICITLY open or close a file within the program.

A generally accepted programming practice dictates that a program should not keep a file open any longer than it needs to have access to the data from the file. If you need access to a file for a brief period of the program's total run time, the program should control that file's opening and closing. The visual presents the following reasons for choosing explicit opens and closes:

- Using a file once
- Many files: Reduce number opened at program start
- Inquiry program accesses one of many files

Explicit open and close

IBM i



```

Line 1      Column 1      Replace      Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      FFileX      IF   E      Disk      UsrOpn
000002
000003      /FREE
000004          Open(E) FileX;    // Explicit open of fileX
000005          /*
000006          If not %Error;
000007              Read FileX;
000008              If not %eof(FileX);
000009                  :
000010                  : // More logic
000011                  :
000012          EndIf;
000013
000014          Close(E) FileX;    // Explicit close of fileX
000015
000016          Else;
000017              Msg = 'Problem opening FileX -
000018                  Contact Support';
000019          EndIf;
000020
000021          *inlr = *on;
000022          Return;
000023      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-15. Explicit open and close

AS067.0

Notes:

RPG IV has two operations, OPEN and CLOSE, that allow you to control explicitly file opens and closes. You can explicitly close a file that RPG IV has implicitly opened. If you want to control the first open of a file in a program, the keyword USROPN must be specified in the F-specification for the appropriate file. Once a file has been opened, either explicitly by the program or implicitly by program initialization, CLOSE, the OPEN operation may follow in execution sequence.

This example shows FILEX designated for explicit opening by the program. The OPEN operation opens the file and the close operation closes it. The only required entry for both operations is the file name following the opcode.

Remember that turning on *InLR ends the program and automatically (implicitly) closes all files.

Initial open at program start

IBM i

```

FIG1016.RPGLE X
Line 1      Column 1      Replace      Browse
. .... 1....+....2....+....3....+....4....+....5....+....
000001      // Implicit open of filex
000002      FFileX      IF      E          Disk
000003
000004      /FREE
000005      // Program logic
000006      Close(E) FileX; // Explicit close of filex
000007      If not %error;
000008      :
000009      EndIf;
000010      :
000011
000012      Open(E) FileX; // Explicit open of filex
000013      If not %error;
000014      :
000015      EndIf;
000016
000017      *inlr = *on;      // Implicit close of filex
000018      Return;
000019      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-16. Initial open at program start

AS067.0

Notes:

This example is similar to the preceding one, except that the F-spec omits the **USROPN** keyword on FILEX. This indicates that RPG IV should perform the initial file open. When the file has been opened, it can be explicitly closed when not needed by the program.

Open data path

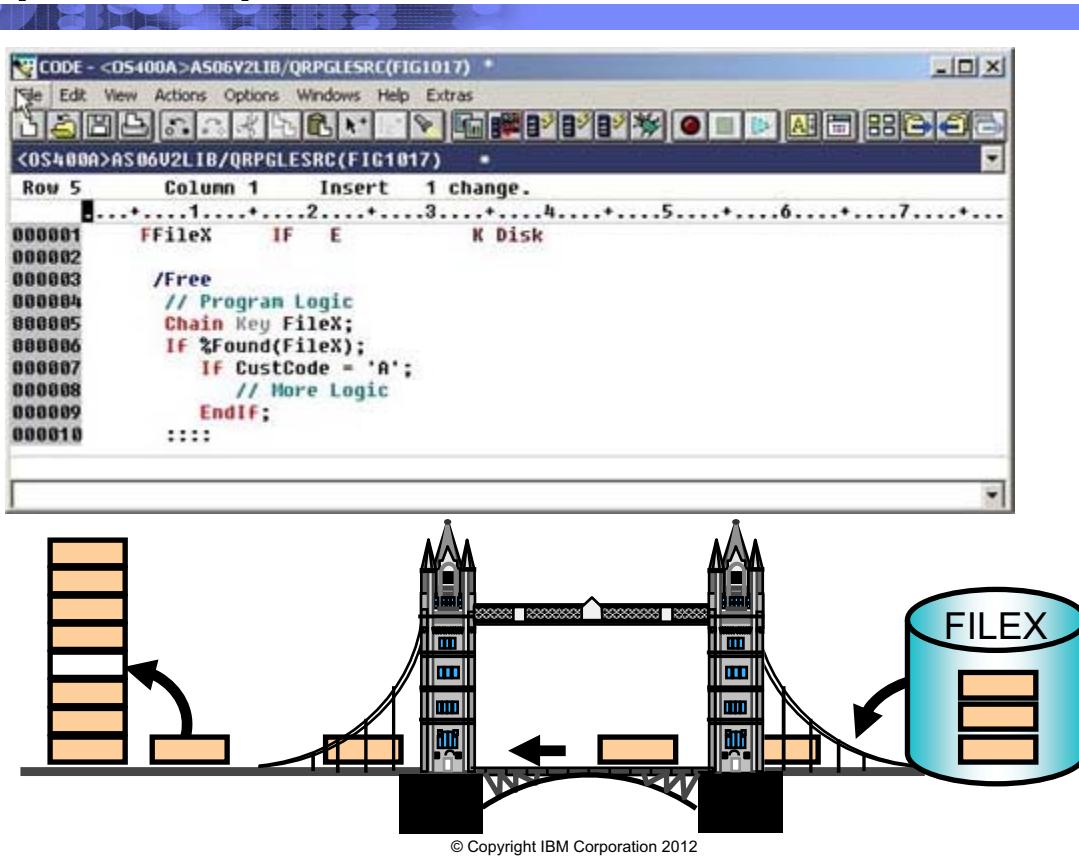


Figure 8-17. Open data path

AS067.0

Notes:

An open data path (ODP) is an I/O area that is opened for every file that you process in your program.

You can think of the ODP as the bridge built between your program and data files it is accessing. The ODP contains the record buffer and the file cursor. In this topic, we discuss RPG IV operation codes that allow you to position the file cursor from within your program.

Using the **SHARE (*YES)** parameter (CRTPF/CRTL or CHGPF/CHGLF) lets two or more programs running in the same job or activation group share an open data path (ODP). An open data path is the path through which all input/output operations for the file are performed. In a sense, it connects the program to a file. If you do not specify the **SHARE (*YES)** parameter for a file, a new open data path is created every time that file is opened.

If an active file is opened more than once *by the same program* in the same job or activation group, you can use the active ODP for the file with the current open of the file *if the program terminated with *InLR off on the prior call*.

Data can be accessed

IBM i

- Externally described file:
 - By file name
 - By record format name
- Program-described file
 - By file name only

© Copyright IBM Corporation 2012

Figure 8-18. Data can be accessed

AS067.0

Notes:

When using operation codes that manipulate data records, notice the name of the file on which the operation acts. In some instances (often in legacy programs), you reference the name of a record format of a file instead of the file name. For example, you would want to reference a specific record format name when a data file has more than one record format (a logical database file allows more than one record format), and you would like to access a particular type of record.

Sequential processing: READ

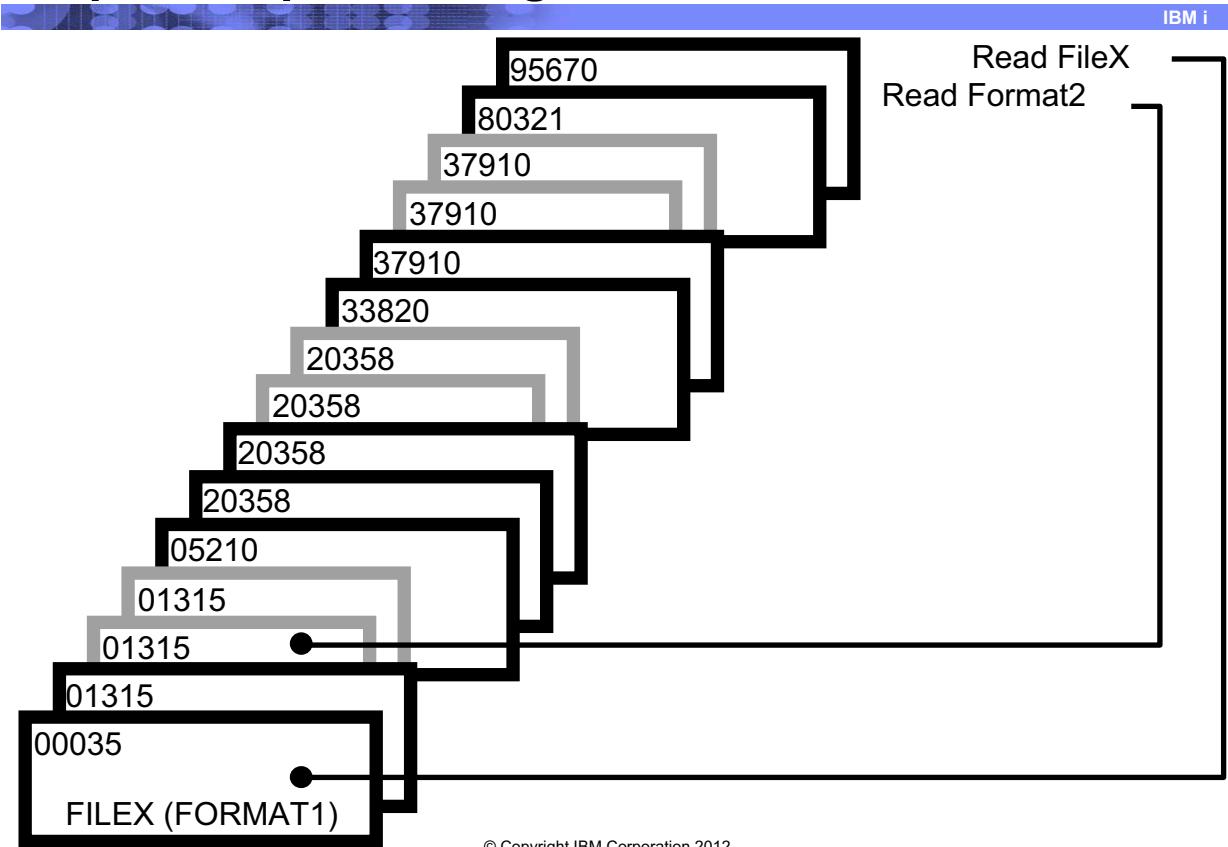


Figure 8-19. Sequential processing: READ

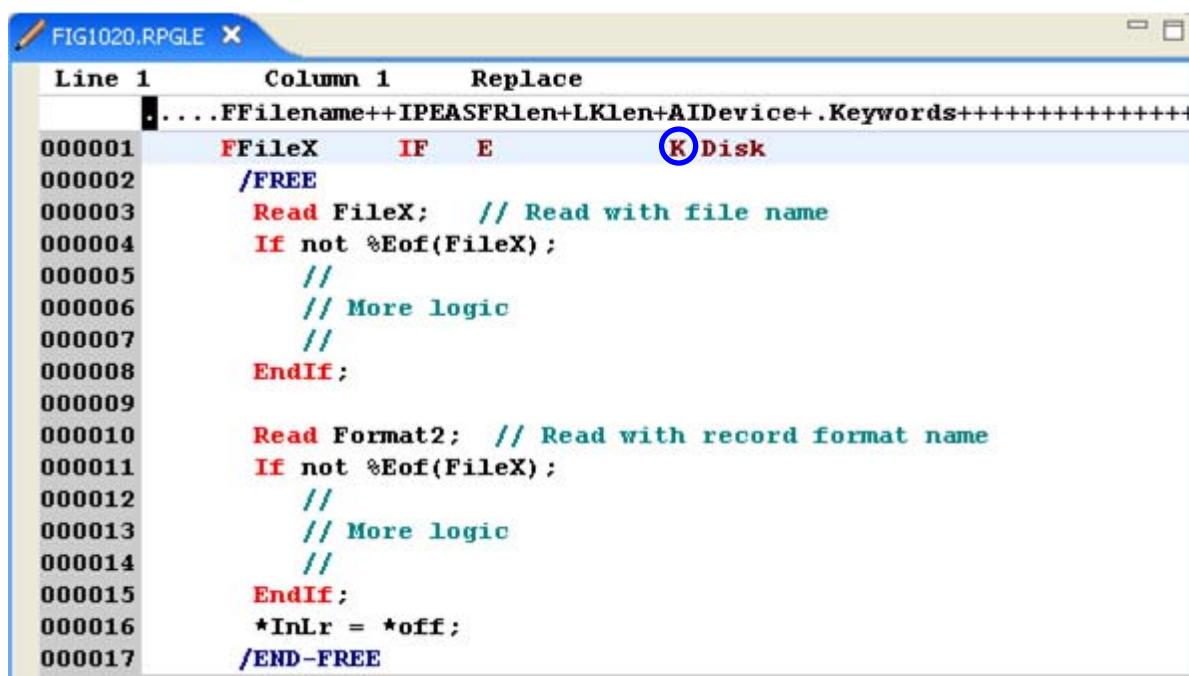
AS067.0

Notes:

What do you notice about the file in this visual?

Sequential processing: READ example

IBM i



```

FIG1020.RPGLE X
Line 1      Column 1      Replace
. ....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001     FFileX      IF      E          K Disk
000002     /FREE
000003     Read FileX; // Read with file name
000004     If not %Eof(FileX);
000005     //
000006     // More logic
000007     //
000008     EndIf;
000009
000010    Read Format2; // Read with record format name
000011    If not %Eof(FileX);
000012    //
000013    // More logic
000014    //
000015    EndIf;
000016    *InLr = *off;
000017    /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-20. Sequential processing: READ example

AS067.0

Notes:

The READ operation retrieves records sequentially. A file name or a record format name designates which file or record format in the file to access. The %Eof should be used to check when end of file is reached.

To use a record format name, the file must be externally described (for logical files only, if ever used).

Read under format results in only those records with this particular format being presented to the program. You would normally use a record format with a logical file.

Reading records using the file name results in all records, whatever their format, being presented to the program.

Sequential processing: Record addition with write

IBM i

```

 1.....1.....2.....3.....4.....5.....6.....7.....+
000001 FFileX  UF A E K Disk
000002
000003 /FREE
000004   Write(E) FileXFmt;
000005   IF %Error;
000006     // More logic
000007   EndIf;
000008   *InLr = on;
000009 /END-FREE

```

```

 1.....1.....2.....3.....4.....5.....6.....7.....+
00002 FFileY  0   E   DISK
00003
00004 /FREE
00005   Write(E) FileYFmt;
00006   IF %Error;
00007     // More Logic
00008   EndIf;
00009   *InLr = *on;
00010 /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-21. Sequential processing: Record addition with write

AS067.0

Notes:

Note the **A** (record addition) is required in position 20 of the file's F-spec, when the file is opened for update.

In the first example, we are adding records to an existing file. In the second example, we are writing to a new file.

Random processing: WRITE by RRN

The figure consists of two side-by-side screenshots of the IBM i RPGLE editor. Both windows have a title bar labeled 'FIG1021A.RPGLE' and 'FIG1021B.RPGLE' respectively.

FIG1021A.RPGLE:

```

Line 1      Column 1      Replace          Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
|000001     FFileX      UF A E           K Disk
000002
000003     /FREE
000004       Write(E) FileXFmt;
000005       If %Error;
000006         // More logic
000007       EndIf;
000008       *InLr = on;
000009     /END-FREE
  
```

FIG1021B.RPGLE:

```

Line 1      Column 1      Replace          Browse
.....1....+....2....+....3....+....4....+....5...
000001     // Output to a new file
000002     FFileY      O   E             DISK
000003
000004     /FREE
000005       Write(E) FileYFmt;
000006       If %Error;
000007         // More Logic
000008       EndIf;
000009       *InLr = *on;
000010     /END-FREE
  
```

© Copyright IBM Corporation 2012

Figure 8-22. Random processing: WRITE by RRN

AS067.0

Notes:

On the F-specification, note that position 34 is blank. The **RECNO** keyword defines the field **RRN** as the field that contains the relative record number of the record being accessed. The field **RRN** must be defined in the program, and its value must correspond to the number of the record being written.

Using WRITE with the **RECNO** keyword places the record at this specific **RRN** in the file. The position must not be occupied by an active record, that is, record previously deleted or member that has been initialized with INZPFM.

When you use WRITE without **RECNO**, the record is placed at the end of the member.

Position file cursor using SETLL

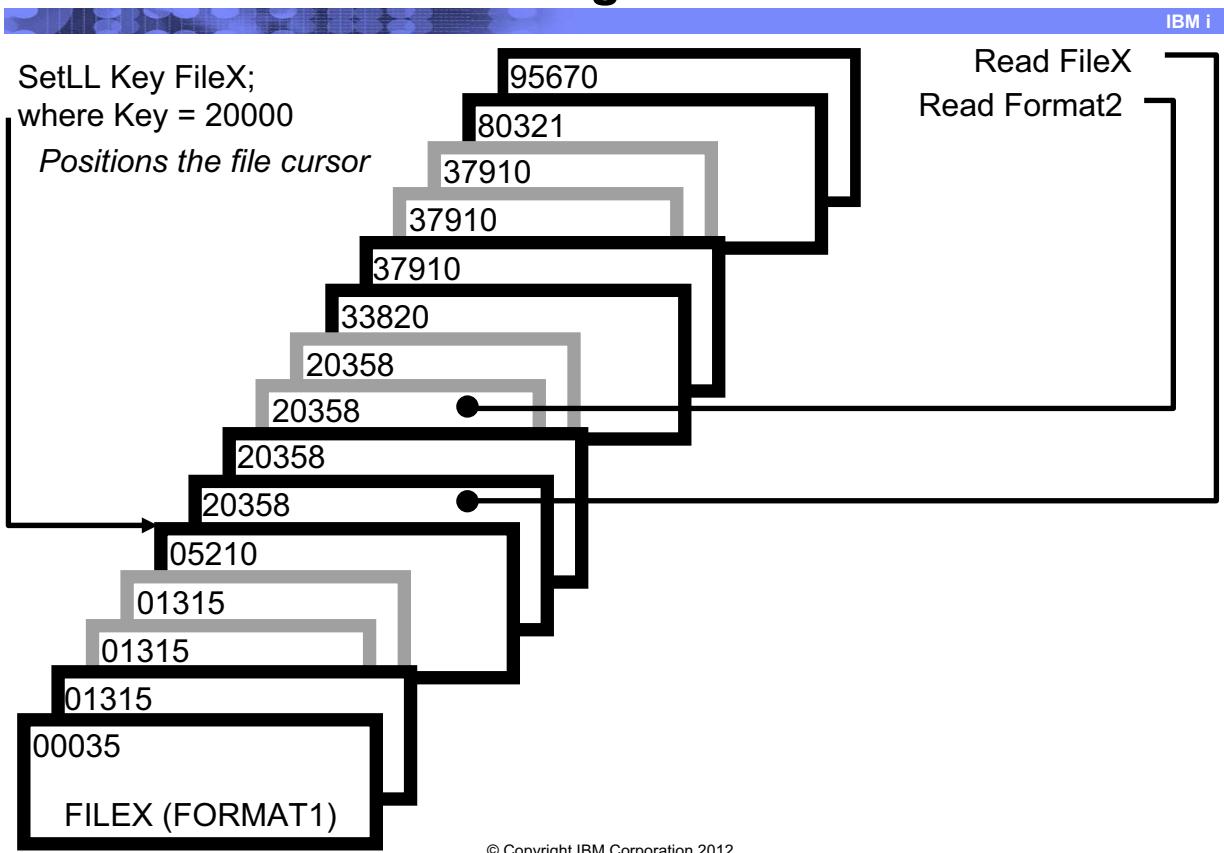


Figure 8-23. Position file cursor using SETLL

AS067.0

Notes:

The SETLL (set lower limit) operation is used to position the file cursor to point to a specific record in the file. The cursor is positioned at the next record that has a key or relative record number that is greater than or equal to the search argument (key or relative record number) specified in Factor1 to the immediate right of the SETLL.

This operation code provides flexibility in where to start sequentially reading a file, especially when you want to begin processing of a record other than the first one in the file. It can also be used to reposition the file at the beginning, once end-of-file has been reached.

SETLL automatically checks for the existence of a record with the key used as an argument. In the example, record key 20000 does not exist, and therefore, our file cursor is placed at the next higher record key, 20358.

Position file cursor using SETLL: Example

IBM i

```

FIG1024.RPGLE X
Line 1      Column 1      Replace      Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      EFileX      IF      E      K Disk
000002      /FREE
000003      SetLL KEY FileX;
000004      If %equal(FileX); //Does record key match key of SetLL?
000005      Read FileX;
000006      If not %eof(FileX); // Record key found
000007          // Process the record
000008      EndIf;
000009      EndIf;
000010      // The LOVAL/HIVAL figurative constants can be used to
000011      // position the file cursor for Keyed access
000012      SetLL *LOVAL FileX;
000013      If %equal(FileX);
000014      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-24. Position file cursor using SETLL: Example

AS067.0

Notes:

Built-in functions work together with **SetLL** to provide information that helps you to continue processing. Remember that **SetLL** positions the file cursor at the record that is equal to or greater than the search key or RRN:

%found = '1'

Cursor positioned at first record greater than the key or RRN.

%equal = '1'

Cursor positioned at record equal to the key or RRN.

%error = '1'

Error occurred; must use with **E** extender.

Also, figurative constants can be used to position the file.

The discussion and examples of using figurative constants which follow assume that ***LOVAL** and ***HIVAL** are not used as actual keys in the file.

When used with a file with a composite key, figurative constants are treated as though each field of the key contained the figurative constant value. Using SETLL with *LOVAL positions the file so that the first read retrieves the record with the lowest key. In most cases, when duplicate keys are not allowed, *HIVAL positions the file so that a READP retrieve the last record in the file or a READ receives an end-of-file indication.

Note that *LOVAL cannot be used to position the file cursor for a file to be processed by RRN. *LOVAL will generate an invalid RRN of -999999...9. To position the cursor at the beginning of a file to be processed by RRN, you would use:

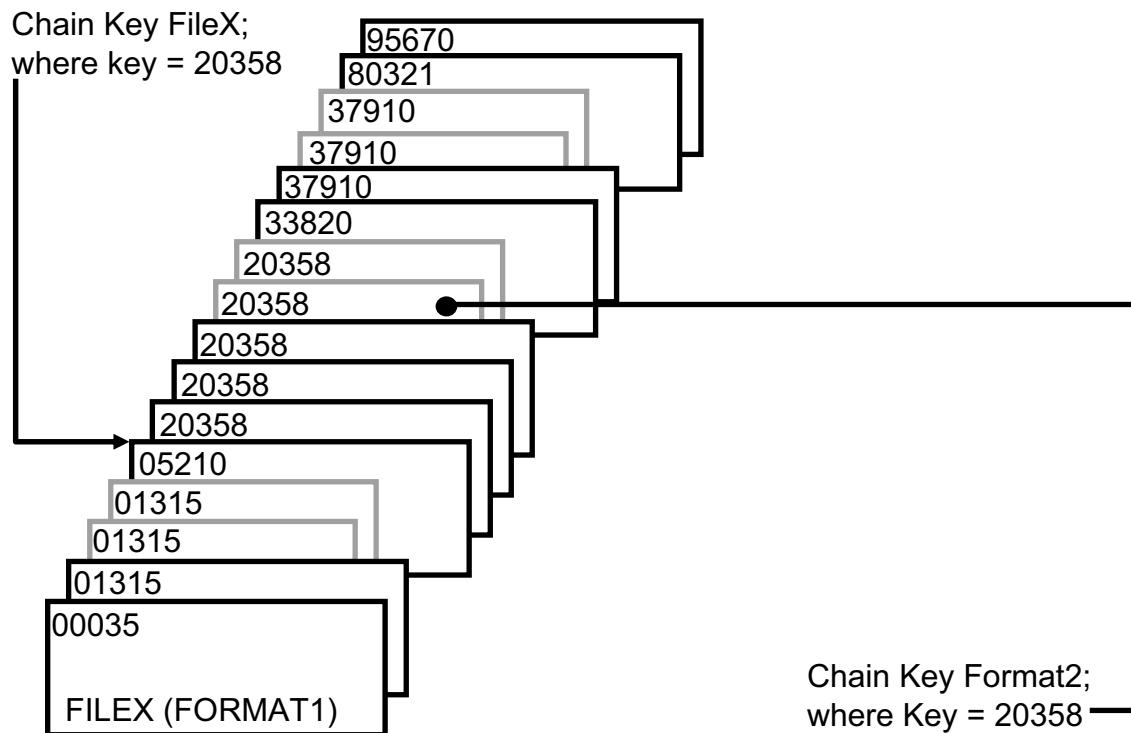
```
SetLL 1 FileX;
```

*START and *END are alternate ways to position the file cursor and are better ways to position to the beginning and end of file than *LOVAL/1 and *HIVAL.

SETLL does not perform an implicit READ or WRITE. It only positions the file cursor.

Random processing: CHAIN by key

IBM i



© Copyright IBM Corporation 2012

Figure 8-25. Random processing: CHAIN by key

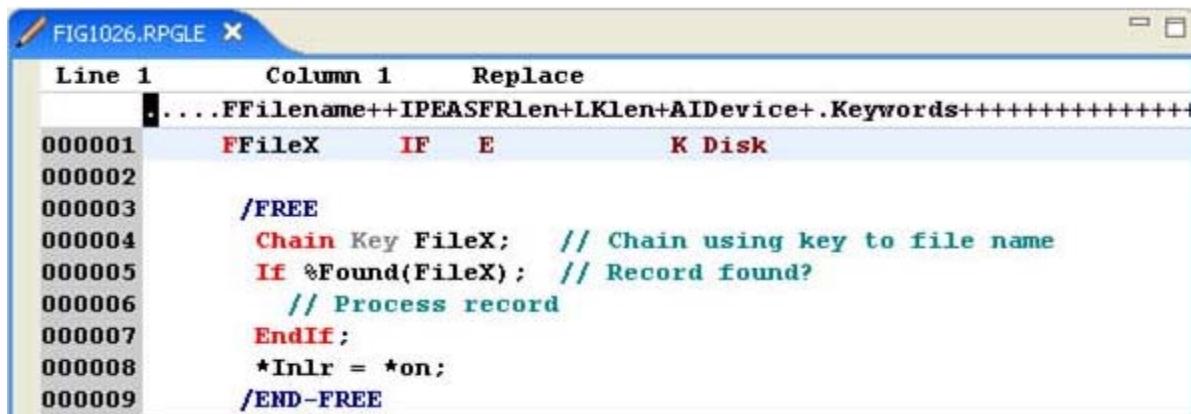
AS067.0

Notes:

RPG IV supports random access of full procedural database files using the CHAIN operation code. A CHAIN to FORMAT1 with a KEY value works the same as a CHAINing to the file name FileX.

Random processing using CHAIN by record key: Example

IBM i



```

Line 1      Column 1      Replace
. ....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      FFileX      IF      E          K Disk
000002
000003      /FREE
000004      Chain Key FileX; // Chain using key to file name
000005      If %Found(FileX); // Record found?
000006          // Process record
000007      EndIf;
000008      *Inlr = *on;
000009      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-26. Random processing using CHAIN by record key: Example

AS067.0

Notes:

The CHAIN operation requires a record key as the search argument. The literal or data field of the search argument contains the key value of the record to be randomly read. The argument that follows the key, also required, contains the name of the file or record format from which the record is to be randomly retrieved. You should use the %found BiF to check whether or not a record was found on the CHAIN.

In our example, the field, KEY, contains a value of the key argument to the record in FILEX that you want to read. If the CHAIN does not find a record with that key value, the value of **%found** is '0'. If the record is found, the value of **%found** is '1'. You would code appropriate logic to handle the found and not found situations. When the file contains records with duplicate keys, such that more than one record meets the key value criteria, the system retrieves the first record that matches the key value.

If the CHAIN is successful, the system positions the file cursor or pointer at the record immediately following the retrieved record. If a READ is performed following a successful CHAIN, the record immediately after the CHAINed record, is read. Conversely, if the

CHAIN is unsuccessful, a sequential read operation cannot follow it without a cursor positioning operation such as another CHAIN, SETLL or SETGT (which we cover soon).

Random processing composite key (1 of 2)

IBM i

The figure consists of two vertically stacked screenshots of the IBM i editor.

Top Screenshot (DDS View):

```

Line 1   Column 1   Replace
...+A+.1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
000100  ** FILEX
000200  A
000300  A
000400  A   R FORMATI
000500  A   FLDA   R
000600  A   FLDB   R
000700  A   DESCRIPT R
000800  A   ACTIVE  R
000900  A   FLDC   R
001000  A   X FLDA
001100  A   X FLDB
001200  A   X FLDC

```

Bottom Screenshot (RPG View):

```

Line 1   Column 1   Replace   Browse
...+FileX+IF E K Disk
000001  FFileX  IF E K Disk
000002  // File Key is composed of (FLDA/FLDB/FLDC)
000003  D FldX   I Like(FldA)
000004  D FldY   I Like(FldB)
000005  D FldZ   I Like(FldC)
000006
000007  C KeyArg   KList
000008  C          KFld   FldX
000009  C          KFld   FldY
000010  C          KFld   FldZ
000011
000012  /Free
000013  Chain KeyArg FileX;
000014  If %Found(FileX);
000015  Dow (Not %EOF(FileX)) And (FldX=FldA) And (FldY=FldB) And
000016  (FldZ=FldC);
000017  // Process record
000018  Read FileX;
000019  EndDo;
000020  EndIf;
000021  *InLR = *On;
000022  /End-Free

```

© Copyright IBM Corporation 2012

Figure 8-27. Random processing composite key (1 of 2)

AS067.0

Notes:

V5R1 and earlier releases only!

Use the KLIST and KFLD operations to define a key field search argument when an externally described file has a key consisting of two or more fields. When defined to your program, the composite key can be used as Factor 1 for operations such as CHAIN, SETLL, READE, and so forth.

KLIST names the field your program uses to represent your composite key. In the DDS, the composite key contains three fields, but it could also contain a partial key only composed of just **FLDA** and **FLDB** or simply **FLDA** only.

KFLD specifies a field that participates in the concatenation of the three fields that make up the composite key.

KLIST and KFLDs *can be located anywhere* in your calculations. Notice that the subfields of the composite key are defined in D-specs. The resulting composite key is the argument used as a key to the file, FILEX.

Random processing composite key (2 of 2)

The screenshot shows two code editors side-by-side, both titled 'FIG1028A.RPGLE' and 'FIG1028B.RPGLE'. Both editors have columns for Line, Column, Replace, and Browse.

FIG1028A.RPGLE:

```

Line 1      Column 1      Replace          Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000002    FFileX     IF   E           K Disk
000004        // File Key is composed of (FLDA/FLDB/FLDC)
000005    D FldX      B             Like (FldA)
000006    D FldY      B             Like (FldB)
000007    D Fldz      B             Like (FldC)
000009    /Free
000011        Chain (FldX:FldY:FldZ) FileX;
000012        If %Found(FileX);
000013            Dow Not %Eof(FileX);
000014                // Process record - READE with list syntax at V5R2
000015                    ReadE (FldX:FldY:FldZ) FileX;
000016                EndDo;
000017            EndIf;
000018            *InLR = *On;
000019        /End-Free

```

FIG1028B.RPGLE:

```

Line 1      Column 1      Replace          Browse
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000002    FFileX     IF   E           K Disk
000004        // File Key is composed of (FLDA/FLDB/FLDC)
000005    D KeyDS     E DS          ExtName(FileX:*Key)
000009    /Free
000011        Chain %Kds(KeyDS:2) FileX;           // Use partial key
000012        If %Found(FileX);
000013            Dow Not %Eof(FileX);
000014                // Process record - READE with list syntax at V5R2
000015                    ReadE %Kds(KeyDS:2) FileX;
000016                EndDo;
000017            EndIf;
000018            *InLR = *On;
000019        /End-Free

```

© Copyright IBM Corporation 2012

Figure 8-28. Random processing composite key (2 of 2)

AS067.0

Notes:

V5R2 and later releases!

Although the KLIST/KFLD method is supported, there are better methods that you can code to build a composite key. The key can be passed as a string of parameters. The order of the parameters determine the hierarchy of the constructed key.

Alternatively, you can use the %KDS BiF. This allows you to construct a composite key as a data structure. The data structure may be externally defined, as shown in the above example. We discuss this method in more detail on a subsequent course.

These two methods are available for any opcode that can use a key: Chain, Delete, ReadE, ReadPE, SetLL, and SetGT.

Random processing using CHAIN by RRN

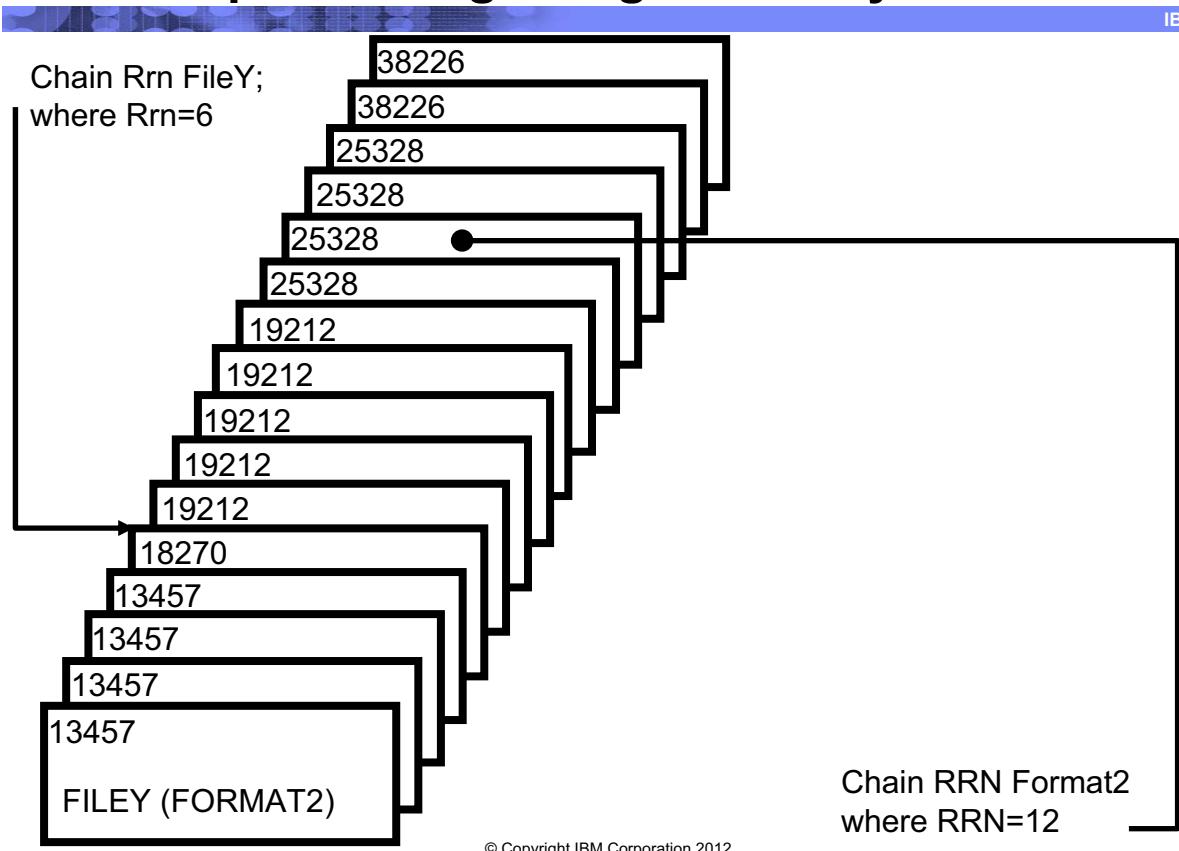


Figure 8-29. Random processing using CHAIN by RRN

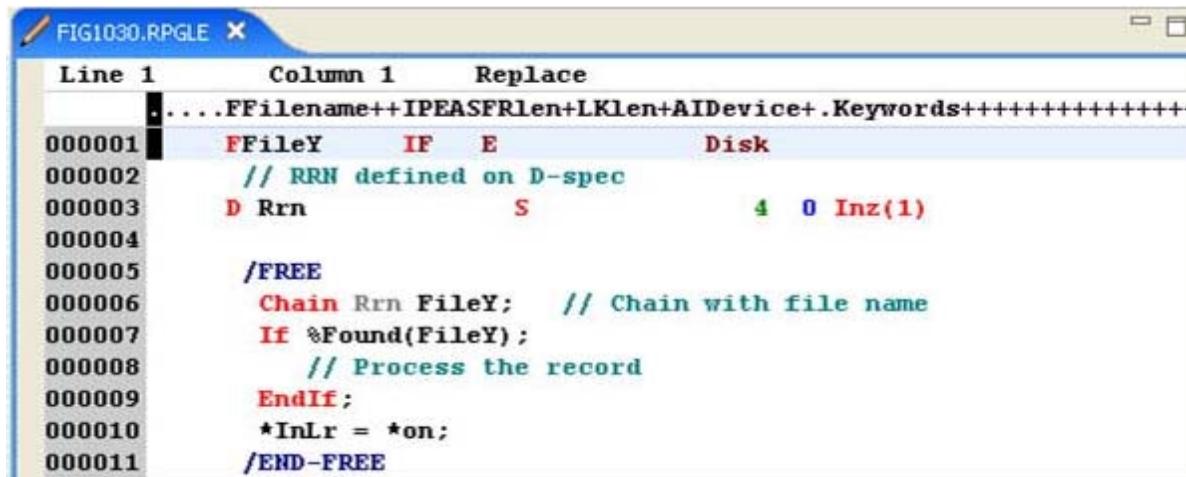
AS067.0

Notes:

In this case, we use a relative record number to process the file randomly. If the records we want is record 6, then we CHAIN to RRN number 6 in the file.

CHAIN not only positions the cursor to the sixth record but it also reads record number 6.

Random processing using CHAIN by RRN: Example



The screenshot shows an IBM i terminal window titled "FIG1030.RPGL". The code is as follows:

```

Line 1      Column 1      Replace
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      FFileY      IF   E           Disk
000002          // RRN defined on D-spec
000003      D Rrn          S           4   0 Inz(1)
000004
000005      /FREE
000006      Chain Rrn FileY; // Chain with file name
000007      If %Found(FileY);
000008          // Process the record
000009      EndIf;
000010      *InLr = *on;
000011      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-30. Random processing using CHAIN by RRN: Example

AS067.0

Notes:

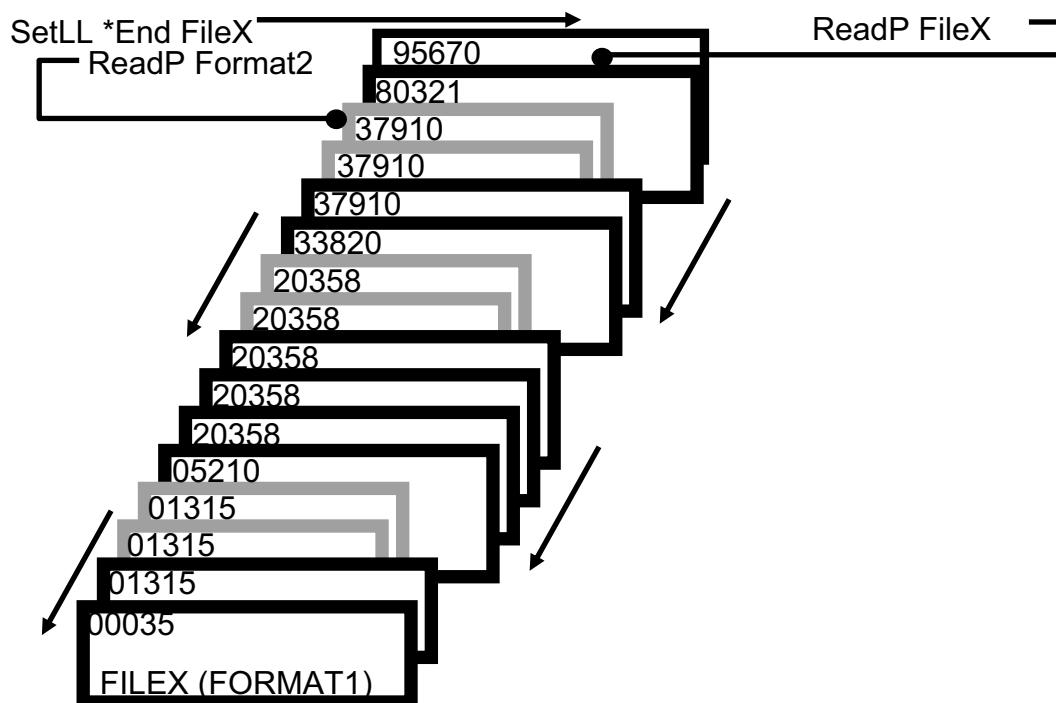
When you leave the record address type (RAT) (position 34 of F-spec) blank for a full procedural file, all CHAIN operations must be processed based upon a relative record number.

RRN is a field whose value is used to retrieve the record corresponding to that relative position in the file. If **RRN** equals 6, the sixth record in the file is retrieved.

RRN could be defined in D-specs as shown in the example, or it could be read from a data file.

Sequential processing: READP

IBM i



© Copyright IBM Corporation 2012

Figure 8-31. Sequential processing: READP

AS067.0

Notes:

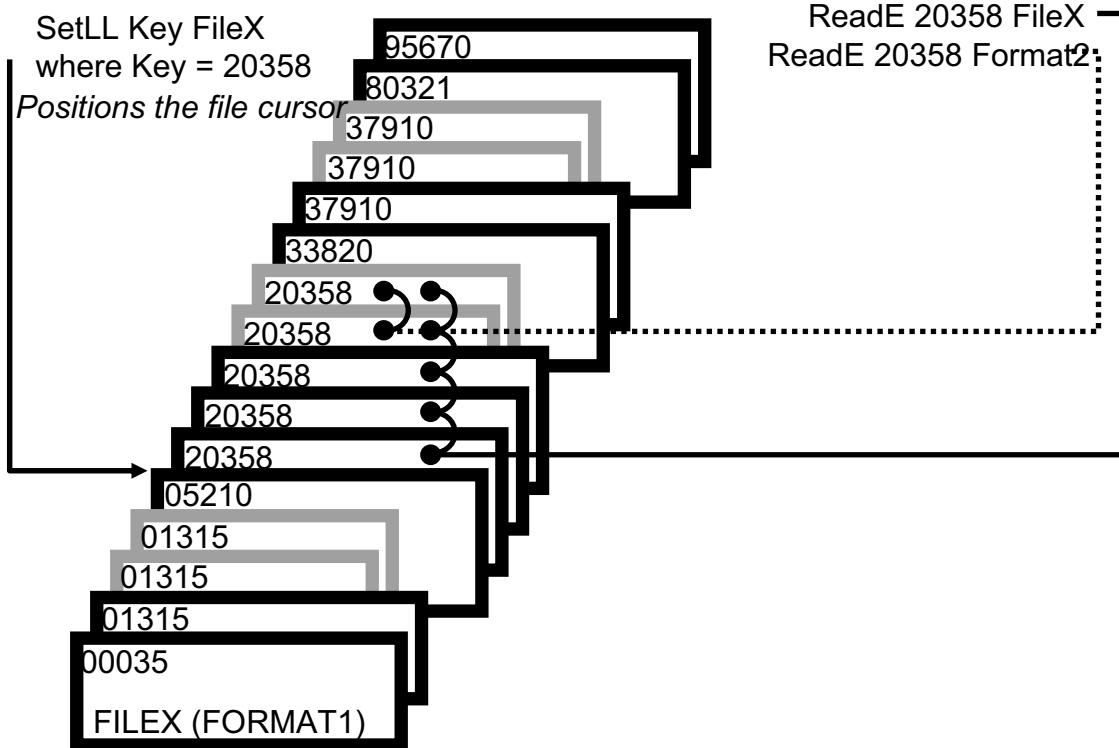
The READP (read prior record) operation, like the READ operation, sequentially reads the file. The difference between the two operations is the direction of the read. READP moves backwards (toward the front) through the file. You should use the %eof BiF to handle reaching the beginning of file (BOF).

Just as READ retrieves the next sequential record from the file, READP retrieves the prior record from point of cursor. This operation reads the file sequentially backwards.

SETLL with *HIVAL or *END is a good method for positioning to EOF in order for READP to read backwards or simply to read only the last record.

Sequential processing: READE

IBM i



© Copyright IBM Corporation 2012

Figure 8-32. Sequential processing: READE

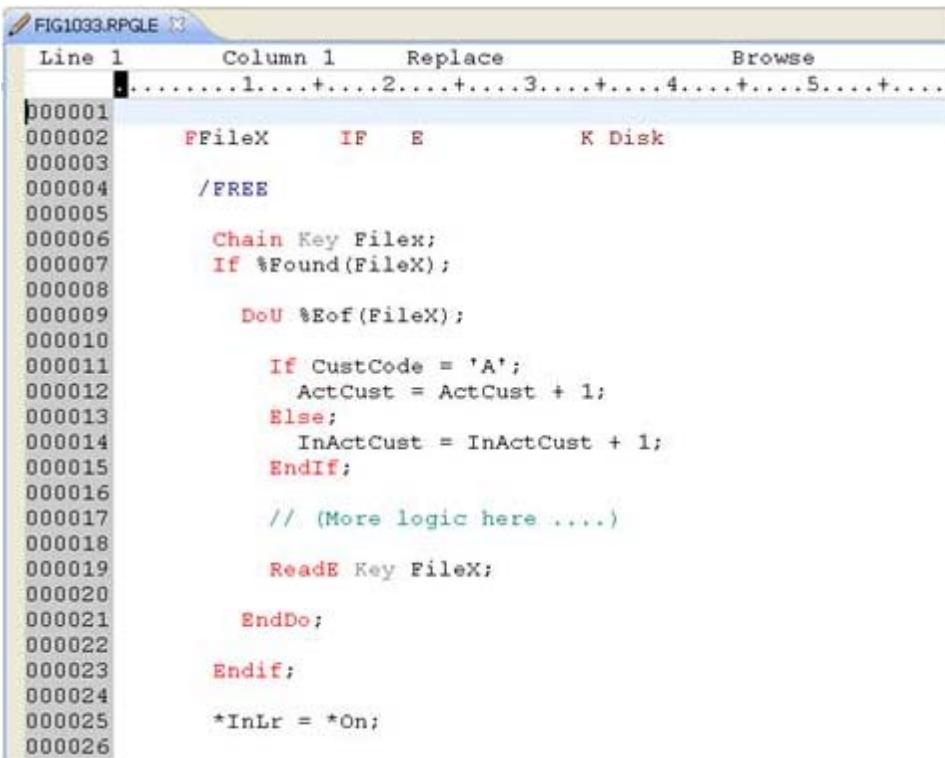
AS067.0

Notes:

Use the READE (read equal) operation code to sequentially read the next record of a full procedural file if the key of that record matches the value specified by the first argument.

This operation code can help process a group of records with identical keys.

Sequential processing: READE example



```

FIG1033.RPGLE
Line 1      Column 1      Replace      Browse
.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+
000001
000002      FFileX      IF      E      K Disk
000003
000004      /FREE
000005
000006      Chain Key FileX;
000007      If %Found(FileX);
000008
000009      DoU %Eof(FileX);
000010
000011      If CustCode = 'A';
000012          ActCust = ActCust + 1;
000013      Else;
000014          InActCust = InActCust + 1;
000015      EndIf;
000016
000017      // (More logic here ....)
000018
000019      ReadE Key FileX;
000020
000021      EndDo;
000022
000023      Endif;
000024
000025      *InLr = *On;
000026

```

© Copyright IBM Corporation 2012

Figure 8-33. Sequential processing: READE example

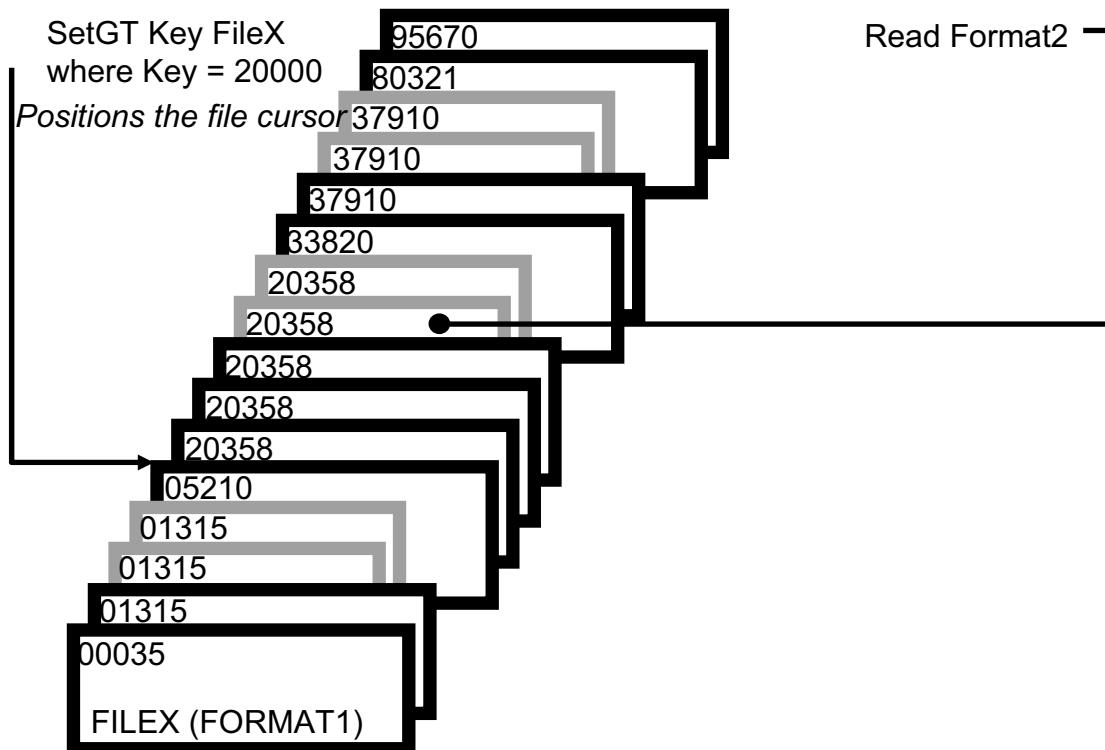
AS067.0

Notes:

This example starts with a CHAIN to FILEX. If a record is found (the CHAIN is successful), the %Found BiF is set to '1' and we process within the DoU LOOP. At the end of the loop, we request a read of the next record with the same KEY as our initial CHAINed record. This is a common method for processing sets of records with duplicate keys in a file.

Positioning file cursor: SETGT

IBM i



© Copyright IBM Corporation 2012

Figure 8-34. Positioning file cursor: SETGT

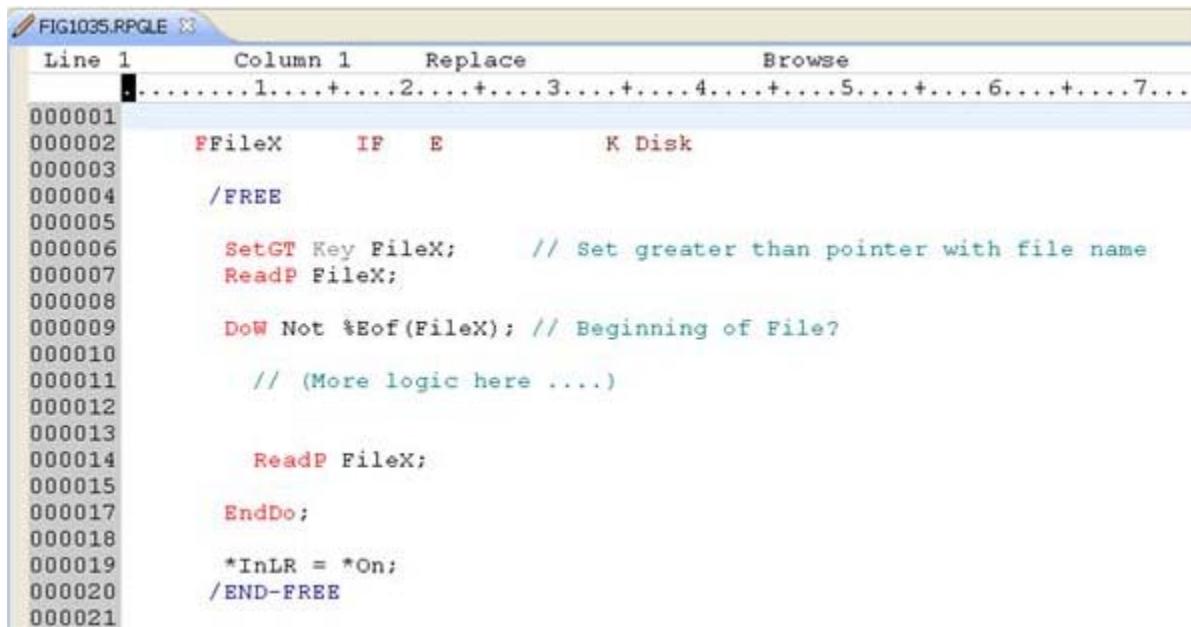
AS067.0

Notes:

Use the SETGT (Set greater than) operation to position the cursor (pointer) in the file. The cursor is positioned at the next record that is greater than the key or relative record number specified as the search argument.

READP and SETGT

IBM i



```

FIG1035.RPGLE X
Line 1      Column 1      Replace      Browse
.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7...
000001
000002     PFileX      IF      E          K Disk
000003
000004     /FREE
000005
000006     SetGT Key FileX;    // Set greater than pointer with file name
000007     ReadP FileX;
000008
000009     DoW Not %Eof(FileX); // Beginning of File?
000010
000011     // (More logic here ....)
000012
000013
000014     ReadP FileX;
000015
000016     EndDo;
000017
000018     *InLR = *On;
000019     /END-FREE
000020
000021

```

© Copyright IBM Corporation 2012

Figure 8-35. READP and SETGT

AS067.0

Notes:

This example uses SETGT and the search field KEY to position the file cursor to the first record whose value is greater than the value of the search field KEY in FILEX. Once the cursor is positioned, we use READP to read previous (or backwards) in the file.

READPE

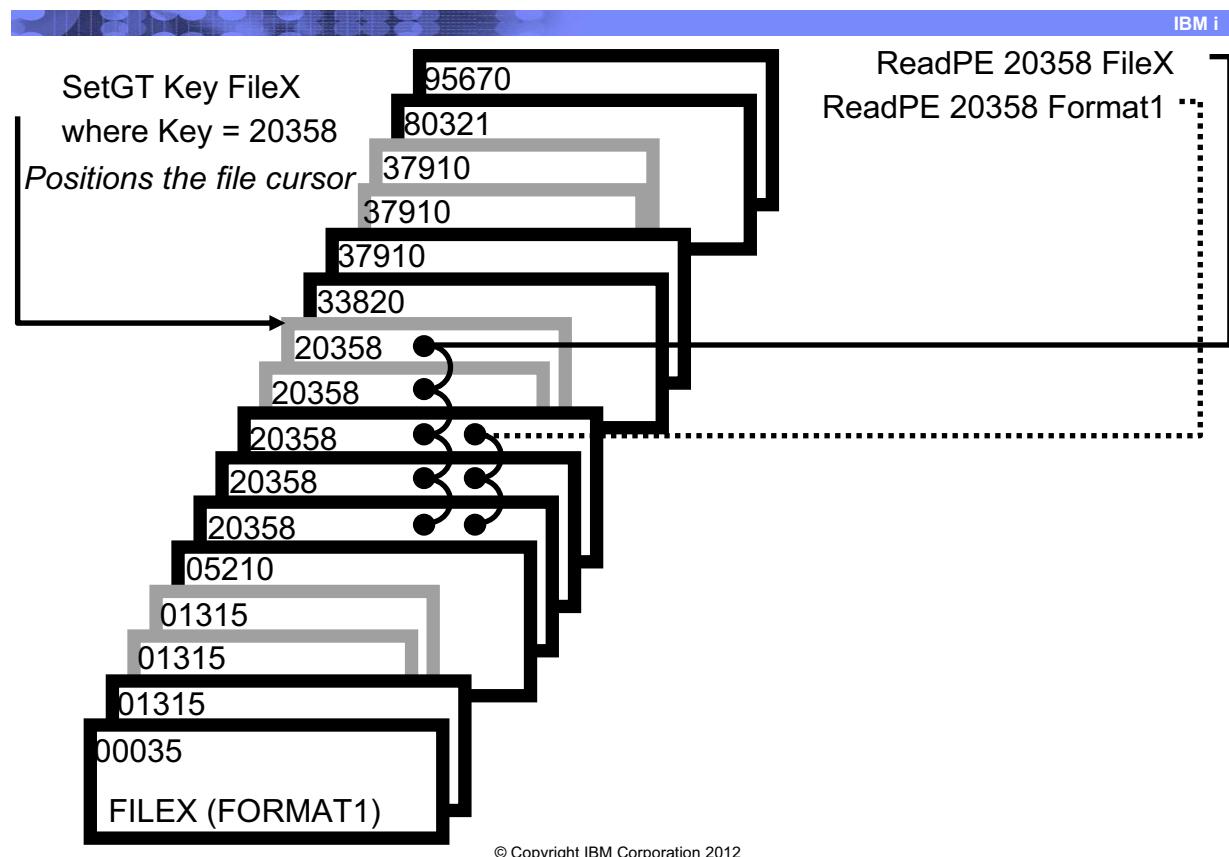


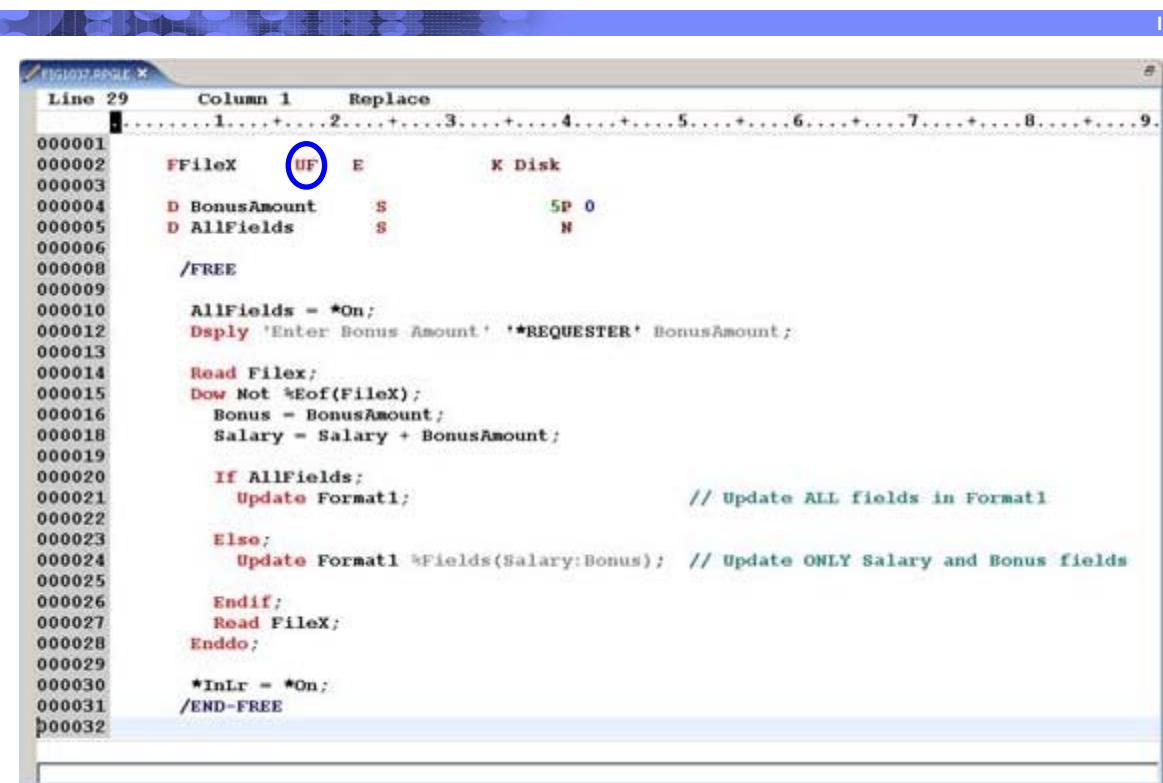
Figure 8-36. READPE

AS067.0

Notes:

Use the READPE (read prior equal) operation code to sequentially read the previous record of a full procedural file if the key of that record matches the value specified by the key search argument. This operation code can help process a group of records with identical keys in a backward (toward the front) fashion.

UPDATE



```

Line 29    Column 1      Replace
000001 1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9.
000002 FFileX   UF   E       K Disk
000003
000004 D BonusAmount     S           SP 0
000005 D AllFields       S           N
000006
000007 /FREE
000008
000009
000010 AllFields = *On;
000011 Dsply 'Enter Bonus Amount' /*REQUESTER*/ BonusAmount;
000012
000013 Read FileX;
000014 Dow Not %Eof(FileX);
000015   Bonus = BonusAmount;
000016   Salary = Salary + BonusAmount;
000017
000018 If AllFields;
000019   Update Format1;          // Update ALL fields in Format1
000020
000021 Else;
000022   Update Format1 %Fields(Salary:Bonus); // Update ONLY Salary and Bonus fields
000023
000024 Endif;
000025 Read FileX;
000026 Enddo;
000027
000028 *InLr = *On;
000029 /END-FREE
000030
000031
000032

```

© Copyright IBM Corporation 2012

Figure 8-37. UPDATE

AS067.0

Notes:

File maintenance involves adding or deleting records from database files or changing the information in database records. You designate a file as update-capable by coding a **U** in position 17 of the F-spec.

The UPDATE operation rewrites a single record, replacing all of the fields for the record to be updated to the file specified. You must be a valid record format name of the file to be updated if the file is externally described. Before you can update a record, the record must have been retrieved by successfully completing a READ or a CHAIN operation in order to establish which record is to be updated.

Since V5R2, you can use the %Fields BiF to identify the specific fields to be updated. Refer to the *Reference manual* for further details.

Database record locks

IBM i

- Program can lock only one record per file:
 - A record is locked when program reads it from an update file
 - A record is released when the program:
 - Updates the locked record of the update file
 - Writes another record to the update file
 - Closes the update file
 - Reads another record in same file for update

© Copyright IBM Corporation 2012

Figure 8-38. Database record locks

AS067.0

Notes:

On any multi-user system, problems can arise with the simultaneous use of the same database file. It is possible that if two users access the same record for update, make changes to the record, and then rewrite it to the file, one of the user's changes might get lost. File locking and record locking are two approaches that help programmers deal with this kind of problem.

File locking is the easiest kind of lock to implement although it is not recommended unless required by the circumstances of the application. Through the use of CL commands, you can limit access to a file to one user at a time.

From a user's viewpoint, it is more practical to enable multiple user access to the same files at the same time. RPG IV has a built-in, automatic locking feature that keeps users from UPDATING the same record. A record is locked when it is read for update, and released when the record is updated, a new record is written to the file, another record is read from the file, or the file is closed.

i file lock states

IBM i

	User A Lock State	User A Operations	User B Lock State	User B Operations
1	*EXCL	ALL	NONE	NONE
2	*EXCLRD	ALL	*SHRRD	READ
3	*SHRUPD	ALL	*SHRUPD	READ, ADD/DEL/UPD
			*SHRRD	READ
4	*SHRNUP	READ	*SHRNUP	READ
			*SHRRD	READ
5	*SHRRD	READ	*EXCLRD	READ, ADD/DEL/UPD
			*SHRUPD	READ, ADD/DEL/UPD
			*SHRNUP	READ
			*SHRRD	READ

© Copyright IBM Corporation 2012

Figure 8-39. i file lock states

AS067.0

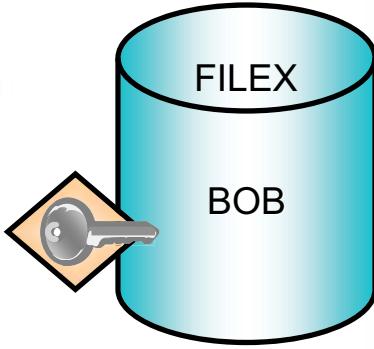
Notes:

- For *SHRUPD, IBM i enforces a lock at the record level if multiple users try to read the same record.
- RPG IV also requests *SHRUPD (3) when opening a file for update or add.
- The CL commands ALCOBJ can be used to obtain a lock state on a file. DLCOBJ releases allocations of lock states.

Record locking

IBM i

PROGA



```

PROGA.RPGLE
Line 1      Column 1   Replace
000001      .....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9.
000002      FfileX    UF   E           K Disk
000003
000004      D BonusAmount     S           5P 0
000005      D
000006      /FREE
000007
000008
000009
000010      Dsply 'Enter Bonus Amount' **REQUESTER* BonusAmount;
000011
000012      Read FileX;
000013      Dow Not %Eof(FileX);
000014          Bonus = BonusAmount;
000015          Salary = Salary + BonusAmount;
000016
000017
000018      Update Format1;
000019
000020      Read FileX;
000021      Enddo;
000022
000023
000024      *InLr = *On;
000025
000026
000027
000028
000029
  
```

Other programs can process other data but the record **BOB** is locked by PROGA.

© Copyright IBM Corporation 2012

Figure 8-40. Record locking

AS067.0

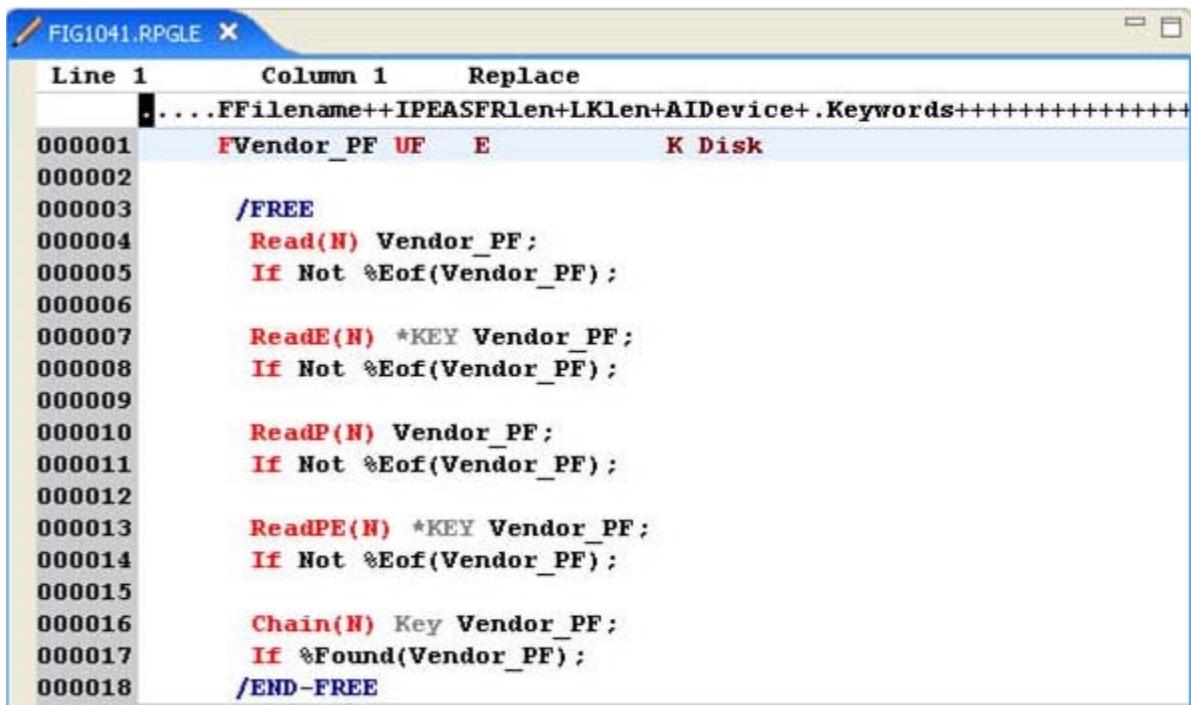
Notes:

A record lock is placed on a record as soon as a read is performed to an update-capable file. In this visual, we Read each record in a loop. Each record, (for example, for employee **Bob**) is locked until the next Read, or an Update is performed.

This can cause a problem for another program trying to access the same record. The other program might abend because it was waiting too long to read the record or because of some other exception. The programmer might need to manage record locking when the job might become disconnected or stuck in a wait.

Read update file without locking record

IBM i



The screenshot shows an IBM i RPGLE editor window titled "FIG1041.RPGL". The code is as follows:

```

Line 1      Column 1      Replace
. ....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      FVendor_PF UF   E           K Disk
000002
000003      /FREE
000004      Read(N) Vendor_PF;
000005      If Not %Eof(Vendor_PF);
000006
000007      ReadE(N) *KEY Vendor_PF;
000008      If Not %Eof(Vendor_PF);
000009
000010      ReadP(N) Vendor_PF;
000011      If Not %Eof(Vendor_PF);
000012
000013      ReadPE(N) *KEY Vendor_PF;
000014      If Not %Eof(Vendor_PF);
000015
000016      Chain(N) Key Vendor_PF;
000017      If %Found(Vendor_PF);
000018      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-41. Read update file without locking record

AS067.0

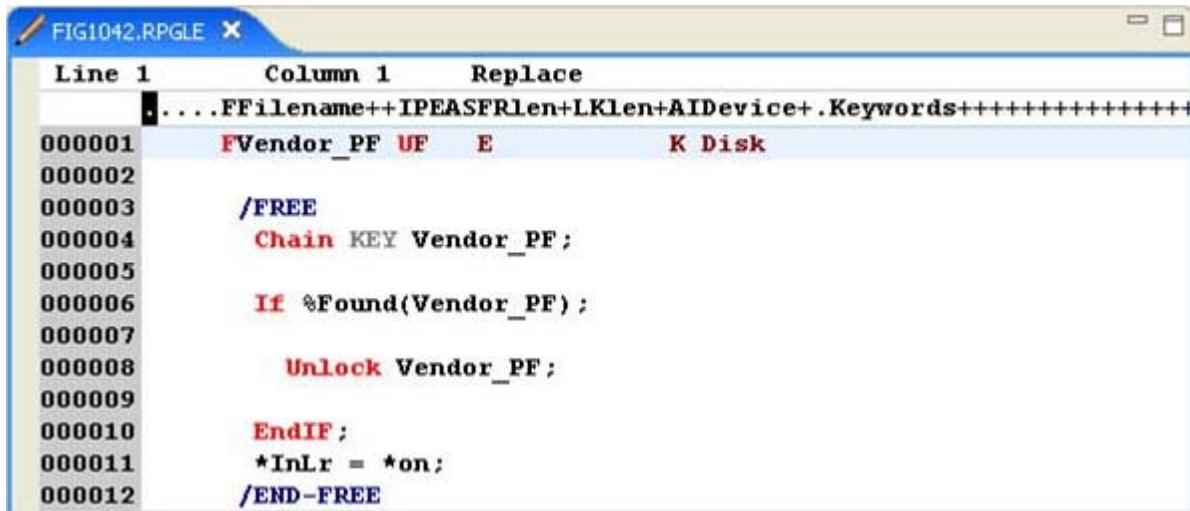
Notes:

The operation extender N allows a record to be read from an update file and *not* locked.

Note we have used the special value *KEY for READE and READPE. This allows the next record to be retrieved with the same key-value as the current record.

Releasing a locked record

IBM i



The screenshot shows a terminal window titled "FIG1042.RPGLE X". The code is written in RPGLE and performs the following steps:

- It includes a header line: ". Ffilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++".
- Line 1: **FVendor_PF UF E K Disk**
- Line 2: **/FREE**
- Line 3: **Chain KEY Vendor_PF ;**
- Line 4: **If %Found(Vendor_PF) ;**
- Line 5: **Unlock Vendor_PF ;**
- Line 6: **EndIF ;**
- Line 7: ***InLr = *on;**
- Line 8: **/END-FREE**

© Copyright IBM Corporation 2012

Figure 8-42. Releasing a locked record

AS067.0

Notes:

If no changes are required to a locked record, you can release it from its locked state using the UNLOCK operation.

Considerations when releasing locked record

IBM i

- Has record changed since read?
- If so, how do you detect record change?
- Use timestamps, flags, record image comparison.

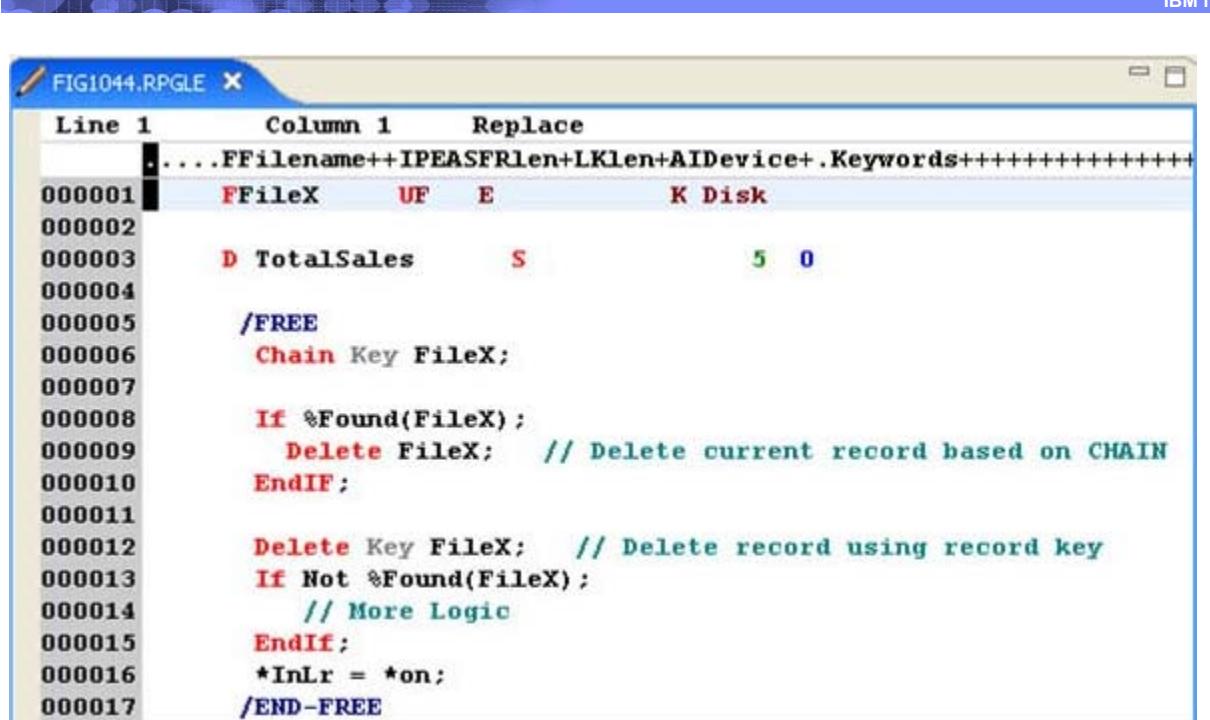
© Copyright IBM Corporation 2012

Figure 8-43. Considerations when releasing locked record

AS067.0

Notes:

DELETE



The screenshot shows an IBM i terminal window titled "FIG1044.RPGL". The window displays RPGLE code for a DELETE operation. The code includes declarations for file pointers, record keys, and various IF statements to handle different scenarios based on record existence. The code uses standard RPG syntax with labels like FFileX, UF, E, K Disk, and various IF blocks.

```

Line 1      Column 1      Replace
.....FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
000001      FFileX      UF      E          K Disk
000002
000003      D TotalSales      S          5  0
000004
000005      /FREE
000006      Chain Key FileX;
000007
000008      If %Found(FileX);
000009          Delete FileX; // Delete current record based on CHAIN
000010      EndIf;
000011
000012      Delete Key FileX; // Delete record using record key
000013      If Not %Found(FileX);
000014          // More Logic
000015      EndIf;
000016      *InLr = *on;
000017      /END-FREE

```

© Copyright IBM Corporation 2012

Figure 8-44. DELETE

AS067.0

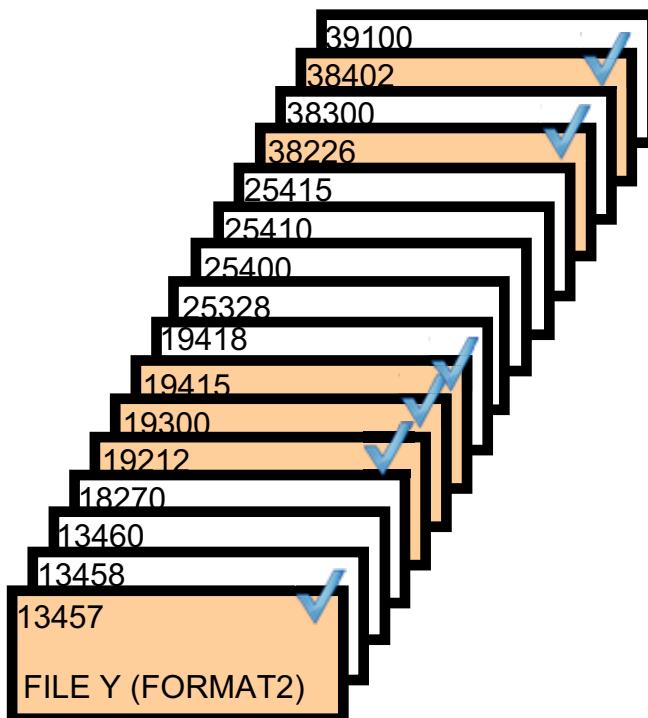
Notes:

If Factor 1 is specified on a DELETE operation, you must use a BiF to check the result. Use %Found to test that the record you want to delete was found. This is the more common form of delete.

You can also perform a delete using a record key.

Deleted records

IBM i



© Copyright IBM Corporation 2012

Figure 8-45. Deleted records

AS067.0

Notes:

You must take a specific action to remove deleted records from a file. Some of your choices are:

- The CL command **RGZPFM** can be used to compress (remove deleted records) and optionally reorganize (order) the data records of a file member. An alternate solution is to have the system automatically reuse deleted record space for new records.
- Recall our discussion of **RECNO(RRN)** where we noted that it is possible to write over deleted records using the WRITE opcode.
- Finally, you can use **CPYF** to remove deleted records from a file by copying the file to a new location.

Machine exercise: Maintaining database files

IBM i



© Copyright IBM Corporation 2012

Figure 8-46. Machine exercise: Maintaining database files

AS067.0

Notes:

Perform the maintaining database files exercise.

Checkpoint

IBM i

1. True or False: The keyword levels for logical files are file, record, field and key.

2. Database files can be specified (position 17 of the F-specification) as which of the following types?
 - a. (I)nput
 - b. (O)utput
 - c. (C)ombined
 - d. (U)pdate

3. True or False: SETLL and SETGT are used to position the file cursor prior to sequential read operations.

© Copyright IBM Corporation 2012

Figure 8-47. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Define physical and logical files in an RPG IV program
- Access data in physical and logical files using Read and Chain
- Modify data in physical and logical files using Write, Update, and Delete
- Use the sequential and random access methods to process data
- Define record keys and use them to process data

© Copyright IBM Corporation 2012

Figure 8-48. Unit summary

AS067.0

Notes:

Unit 9. Coding inquiry programs

What this unit is about

This unit describes how to interact with IBM i (Power i) display files. It reviews their coding and creation and discuss how they are used by interactive RPG IV programs. It also discusses the format and function of RPG IV operation codes ExFmt and Write when used with display files.

What you should be able to do

After completing this unit, you should be able to:

- Describe the properties of a display file
- Create display files using DDS
- Use Exfmt and Write RPG IV operation codes with display files
- Code inquiry RPG IV programs to support one display file

How you will check your progress

- Checkpoint questions
- Machine exercise

Given screen design definition and a program description, the student codes and creates a display file, and writes an RPG program that uses this display file for a simple inquiry application.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe the properties of a display file
- Create display files using DDS
- Use Exfmt and Write RPG IV operation codes with display files
- Code inquiry RPG IV programs to support one display file

© Copyright IBM Corporation 2012

Figure 9-1. Unit objectives

AS067.0

Notes:

Display file descriptions

IBM i

- They define one or more panels (record formats)
- One display file contains all panels for one program or an entire application
- They are coded using DDS (such as printer and database files)
- They can be created or tested using:
 - CODE Designer
 - Screen Design Aid (SDA)
- They are defined at three levels:
 - File
 - Record
 - Field

© Copyright IBM Corporation 2012

Figure 9-2. Display file descriptions

AS067.0

Notes:

Display files are defined using DDS. Keywords enhance your file definitions and can be applied at the file level, the record level and the field level.

Display files can have one record format or several record formats.

Keywords apply at the file, record and field levels. Some keywords can be used at just one level while others can apply at more than one level.

Display record format

IBM i

- It is the basic unit of information passed between user and program
- It is sent by one WRITE or EXFMT operation
- It occupies all lines of a display starting with the lowest defined line number and including all lines used and unused through the highest defined line number:
 - Smallest format occupies one line on the display.
 - Largest display format occupies all lines of the display.
- Multiple display formats can be displayed at the display device
- The limit per display file is 1024 record formats

© Copyright IBM Corporation 2012

Figure 9-3. Display record format

AS067.0

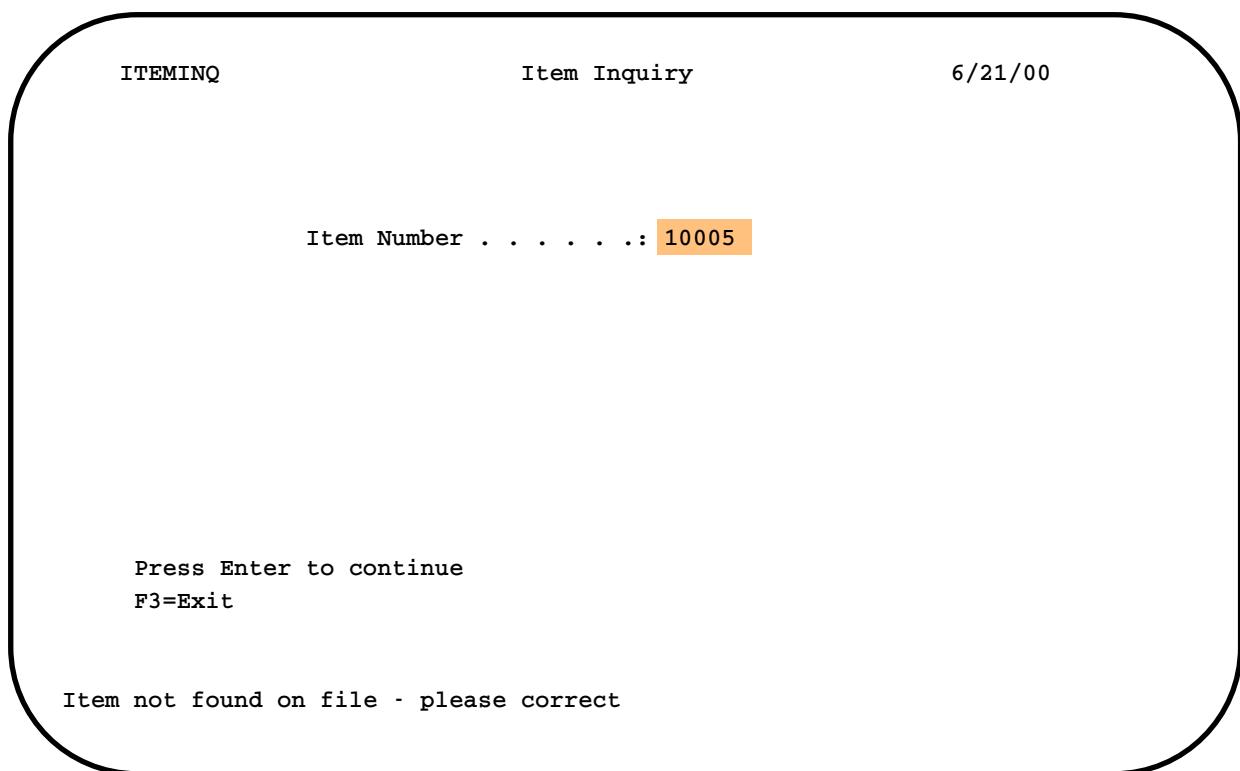
Notes:

Display formats behave by default in the following manner:

- They write a display format:
 - Clears the device of all previous formats
 - Unlocks the keyboard for input
- They read a display format:
 - Locks the keyboard from input
- All input and input/output (I/O - both) fields are underlined.
- All character data entry is uppercase.
- DDS validity checking (if specified by the programmer) causes:
 - Field in error to be displayed in reverse image
 - Cursor to move to beginning of field in error
 - A message to appear
 - Keyboard to lock

Display design (1 of 5)

IBM i



© Copyright IBM Corporation 2012

Figure 9-4. Display design (1 of 5)

AS067.0

Notes:

Let us assume that we have been asked to develop an application that prompts a user for an item number. Randomly read the item master file and display.

1. The detailed information about the item entered as shown in the following visual when the item number exists in the file.
2. The prompt display (shown above) with an error message conditioned by the result of the chain if it was unsuccessful.

Display design (2 of 5)

IBM i

ITEMINQ Item Inquiry 6/21/00

Item Number : 20005
Description : Dry Erase Marker Packs

Quantity on hand : 428
Quantity on order : 10

Supplier Number : 10010
Supplier Catalog No . . . : V145125

F3=Exit F12=Cancel

© Copyright IBM Corporation 2012

Figure 9-5. Display design (2 of 5)

AS067.0

Notes:

This is the detailed information that is displayed for a valid item, in this case, item number 20005, rather than 10005 which was invalid.

We corrected the item number on the previous screen from 10005 to 20005 (which we did not show you).

Display design (3 of 5)

IBM i

```

0.....+....1.....+....2.....+....3.....+....4.....+....5.....+....6.....+....7...
-
-
-
-          0000000000          Item Inquiry          DD/DD/DD
-          (PGMNAM)
5
-
-
-
-          Item Number . . . . . : 33333
-
10
-
-
-
-
15
-
-
-
-
-
20      Press Enter to continue
-          F3=Exit
-
-
-
-          Work screen for record PROMPT: Press Help for function keys.

```

© Copyright IBM Corporation 2012

Figure 9-6. Display design (3 of 5)

AS067.0

Notes:

DDS is used to describe a display file. The DDS can be written manually by entering directly into a source member, or can be generated using either the SDA or the CODE Designer tool as a design and DDS generation tool.

We must produce the DDS source, type DSPF, to create the display file. The display file, as explained previously, contains two record formats of which this is the first format. When using SDA, this is what the layout of the format looks like.

This prompting format is used for inquiry of the item master file by using item number.

An error message is displayed if the item number entered is an invalid one.

Display design (4 of 5)

IBM i

```

0.....+....1.....+....2.....+....3.....+....4.....+....5.....+....6.....+....7...
-
-
-
-      0000000000          Item Inquiry          DD/DD/DD
-      (PGMNAM)
5
-
-
-
-      Item Number . . . . . : 66666
-      Description . . . . . : 00000000000000000000000000000000
10
-      Quantity on hand . . . : 6666666
-      Quantity on order . . . : 6666666
-
-
-      Supplier Number . . . . : 66666
15
-      Supplier Catalog No . . . : 0000000
-
-
-      ** NOTE ** The total quantity available is low (<20) - reorder
-
-
20
-      F3=Exit    F12=Cancel
-
-
-
-      Work screen for record DETAIL: Press Help for function keys.

```

© Copyright IBM Corporation 2012

Figure 9-7. Display design (4 of 5)

AS067.0

Notes:

This is the second display format. It displays the information for the item when it is found successfully. Also, note the message warning of a low stock situation.

Display design (5 of 5)

IBM i

```

-----AAN01N02N03-----Functions*****
REF(=LIBL/ITEM_PF)
CA03(03 'Exit')
INDARA

R PROMPT          10A 0 3 7
PGMNAME          3 35'Item Inquiry'
                  COLOR(WHT)
                  3 64DATE
                  EDTCDE(Y)
ITMNBR           R  D  I  8 20'Item Number . . . . :'
                  8 45
ERRMSG('Item not Found on file - pl-
ease correct' 40)
20 7'Press Enter to continue'
21 7'F3=Exit'
                  COLOR(BLU)

R DETAIL          CA12(12 'Cancel')
PGMNAME          10A 0 3 7
                  3 35'Item Inquiry'
                  COLOR(WHT)
                  3 64DATE
                  EDTCDE(Y)
ITMNBR           R  0 8 45
                  9 20'Description . . . . :'
ITMDESCR         R  0 9 45
                  11 20'Quantity on hand . . . :'
ITMQTYOH         R  0 11 45EDTCDE(2)
                  DSPATR(HI)
                  12 20'Quantity on order . . . :'
ITMQTYOO         R  0 12 45EDTCDE(2)
                  DSPATR(HI)
                  14 20'Supplier Number . . . :'

-----AAN01N02N03-----Functions*****
REF(=LIBL/ITEM_PF)
CA03(03 'Exit')
INDARA

```

© Copyright IBM Corporation 2012

Figure 9-8. Display design (5 of 5)

AS067.0

Notes:

This DSPF illustrates the following elements:

1. F-keys at file/format level
2. Field-level keywords (EDTCDE)
3. DSPATR and COLOR keywords at field level
4. Elementary error message handling (ERRMSG)
5. Conditioning Indicator for field existence and presentation

For display files in general:

- **Conditioning** (Pos 7 - 16) option indicators 01 - 99 are used to condition record or field output; use AND, OR for more than 3 indicator conditioning.
- **Type of Name** (Pos 17) R indicates name of a record format, blank indicates a field name.
- **Name** (Pos 19 - 28) used to name a record format or a field.

- **Reference** (Pos 29) R indicates reference to a previously defined file for field attributes.
- **Length** (Pos 30 - 34) used to determine length of field, if not referenced. This length does not include beginning or ending attribute characters of the field, even though field positioning on the screen must consider the attribute characters. This length is also known as the program length of the field.
- **Data type/KB shift** (Pos 35) designates type of data or keyboard shift attributes for a field.
- **Usage** (Pos 38) specifies a field to be (O)utput, (I)nput, (B)oth I and O, (H)idden, a special output/input field, and (M)essage, a special output field.
- **Location** (Pos 39 - 44) determines the exact location on the display where each field begins.

Display attribute (DSPATR)

IBM i

<u>DDS Keyword</u>	<u>Meaning</u>
• DSPATR (UL)	Underline
• DSPATR (BL)	Blinking field
• DSPATR (CS)	Column separator
• DSPATR (HI)	High intensity
• DSPATR (ND)	Non-display
• DSPATR (RI)	Reverse image
• DSPATR (MDT)	Set on modified data tag
• DSPATR (PC)	Position cursor
• DSPATR (PR)	Protect field

© Copyright IBM Corporation 2012

Figure 9-9. Display attribute (DSPATR)

AS067.0

Notes:

The **DSPATR** keyword requests one or more attributes for the same field. You can specify one keyword with multiple attributes or you can specify multiple keywords, each with an attribute. For example, **DSPATR (CS RI)** is the same as specifying **DSPATR (CS)** followed by **DSPATR (RI)**. Another example is that **DSPATR (UL HI RI)** is the same as **DSPATR (ND)**.

DSPATR and color displays

IBM i

DSPATR(CS) Display Attribute Selected	DSPATR(HI) Display Attribute Selected	DSPATR(BL) Display Attribute Selected	Color Produced on the Color Display Station
			Green (normal)
X			Turquoise
	X		White
		X	Red, no blinking
	X	X	Red, with blinking
X	X		Yellow
X		X	Pink
X	X	X	Blue

© Copyright IBM Corporation 2012

Figure 9-10. DSPATR and color displays

AS067.0

Notes:

When you specify the **DSPATR** keyword (without the COLOR keyword), fields are displayed on color displays with the colors in visual table but without the display attributes that you specified.

For example, if you specify **DSPATR (BL HI)**, a color display shows the field in red with blinking.

Be aware of this when you are coding on a color display device.

Reference:

IBM i Information Center - DDS Reference: Display Files, Keywords for Display Files

Enabling function keys

IBM i

```

Row 4    Column 1      Insert 13 changes.
1...+A*..1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000001 A
000002>>1 A
000003 A
000004 **
000005 A          R PROMPT
000006 A          PGMMNR      10A  O  3  7
000007 A          3 35'Item Inquiry'
000008 A          COLOR(WHT)
000009 A          3 6DATE
000010 A          EDTCODE(Y)
000011 A          8 20'Item Number . . . . .'
000012 A          ITMNBR      R  D  I  8 45
000013 A 40
000014 A          ERRMSG('Item not found on file - pl-
000015 A          EASE CORRECT' 40)
000016 A          20 7'Press Enter to continue'
000017 A          21 7'F3=Exit'
000018 A          COLOR(BLU)
000019 A          **
000020>>2 A          R DETAIL
000021 A          CA12 12 'Cancel'

```

© Copyright IBM Corporation 2012

Figure 9-11. Enabling function keys

AS067.0

Notes:

Display file source code can have DDS keywords coded at the file, record and field levels. This example illustrates various display file keywords coded at each of the three levels.

Note that function keys are enabled at both the file level and the record-level.

CA12, coded at the record-level (**2**), specifies that when DETAIL is in use, F12 is enabled. CA03 is also enabled for this display record format because it was coded at the file level (**1**).

Using a response indicator with the function key is optional. However, the indicator is very useful in your program. It is the convention that the indicator number matches the function key number, but, this is not mandatory.

Function keys

IBM i

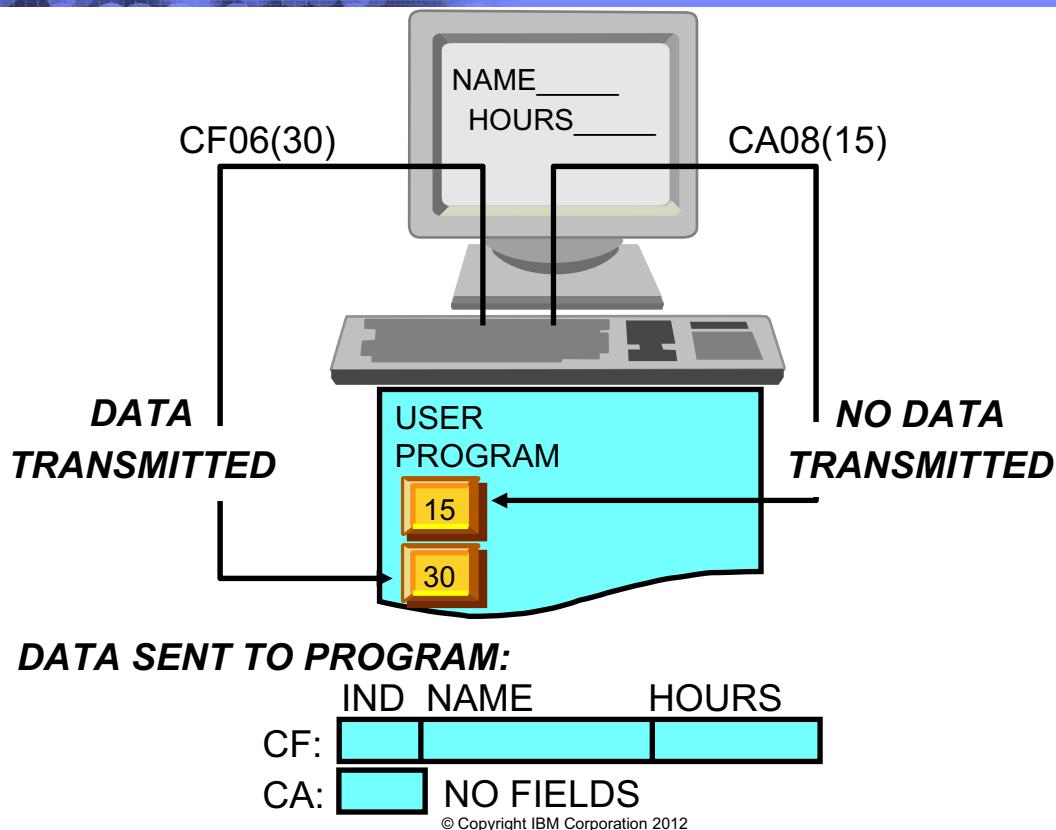


Figure 9-12. Function keys

AS067.0

Notes:

- Input record content - Workstation to i (program).
 1. Workstation ID (WSID)
 2. Function key which ended transaction
 3. Cursor position
 4. Address and data for each input-capable field with modified data tag (MDT) ON
- The above information is formatted by workstation data management.
- A READ makes the information available to the program.

Field-level keywords

IBM i

```
■.....AAN01N02N03..Name+++++RLen++TDpBLinPosFunctions+++++
000001 A REF(*LIBL/ITEM_PF)
000002 A CA03(03 'Exit')
000003 A INDARA
000004 **
000005 A R PROMPT
000006 A PGMNAME 10A 0 3 7
000007 A 3 35'Item Inquiry'
000008 A COLOR(WHT)
000009 A 3 64DATE
000010 A EDITION(Y)
000011 A 8 20'Item Number . . . . .'
000012 A ITMNBR R D I 8 45
000013 A 40 ERMSG('Item not found on file - pl-
000014 A EASE CORRECT' 40)
000015 A 20 7'Press Enter to continue'
000016 A 21 7'F3=Exit'
000017 A COLOR(BLU)
000018 **
000019 A R DETAIL
000020 A CA12(12 'Cancel')
```

© Copyright IBM Corporation 2012

Figure 9-13. Field-level keywords

AS067.0

Notes:

This example illustrates various display file keywords coded at the field level.

Note that keywords, such as **DATE**, **TIME**, and **SYSNAME**, are keywords that retrieve and display the obvious system values.

You can use the **USER** field-level keyword to display the user profile name for the current job as a constant (output-only) field that is 10 characters long. Use the **SYSNAME**, **USER**, **DATE**, and **TIME** field-level keywords to display the appropriate information as a constant on your display formats.

We also use the **ERRMSG** keyword to be displayed when data entered into this field on the screen is in error. If it is, the user sees an error message and has to correct the data.

Validity checking keywords

CHECK(XX)

ME: Mandatory Enter

MF: Mandatory Fill

FE: Field Exit Check

COMP (Relational-Operator value)

LC: Lowercase

RB: Right justify / Blank Fill

RZ: Right justify / Zero Fill

EQ, NE, LT, NL, GT, NG, LE, GE

CMP (Relational-Operator value)

Example: FIELD 1 ----- COMP(EQ 'ABC')

FIELD2 ----- COMP(NG 1000)

RANGE (Low-Value High-Value)

Example: FIELD1 ----- RANGE('A' 'F')

FIELD2 ----- RANGE(1 1000)

VALUES (Value1 Value2 ----- Value 100)

Example: FIELD1 ----- VALUES('A' 'B' 'C')

FIELD2 ----- VALUES(33 42 01)

CHKMSGID: allows a user defined message when validity check error is detected

Example: FIELD1 ----- RANGE (1 50)

----- CHKMSGID(MSG1044 MSGFILE)

© Copyright IBM Corporation 2012

Figure 9-14. Validity checking keywords

AS067.0

Notes:

It is very easy to edit data entry, control certain keyboard functions and move the cursor as your users key data. DDS keywords are available for many of these functions.

Validity checking keywords help offload data entry edit tasks from your program. i5/OS (OS/400) takes on the task of performing some of the more basic edits of your users' input.

DDS message keywords

IBM i

- **ERRSFL**
 - Provides system-supplied subfile
 - Displays multiple messages at once
 - Allows message scrolling
- **ERRMSG**
 - Specifies message text
 - Associates messages with a data field
 - Detects program errors
 - Appears at bottom of display
- **ERRMSGID**
 - Uses text of predefined message
 - Associates messages with a data field
 - Detects program errors
 - Appears at bottom of display
- **CHKMSGID**
 - Uses text of predefined message
 - Associates message with data field when used with the **CHECK** keyword
 - Detects DDS errors on the display
 - Appears at bottom of display

© Copyright IBM Corporation 2012

Figure 9-15. DDS message keywords

AS067.0

Notes:

There are many options for message handling on your displays. Some things to consider before you choose a method:

- Are you using predefined or impromptu messages?
- Where on the display would you like the message?
- Is the message the result of a program edit?
- Does DDS perform a validity check?
- Is the message really a constant?

Note that **ERRMSG** and **ERRMSGID** require the use of indicators and are associated directly with data fields. Any display that uses these messages must be coded in similar fashion. What happens when your message or indicators must change?

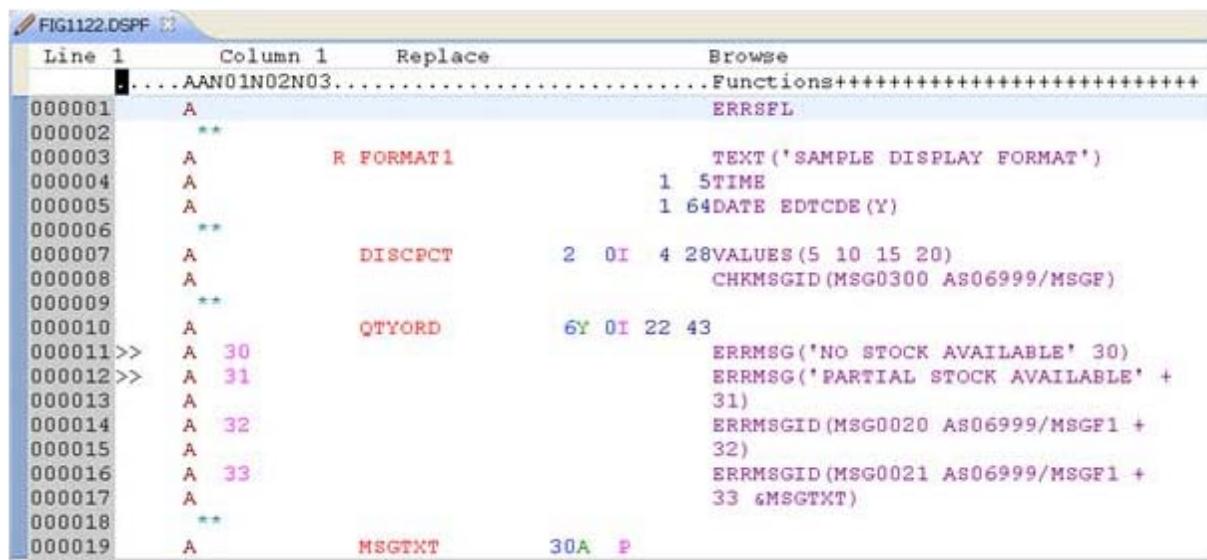
Additional options: The DDS keyword **MSGLOC** (message location) designates the line of the display on which error messages are displayed. For example, **MSGLOC(22)** sets line

22 as the line for error messages. If the keyword is not used, the message line position defaults to line 24 (on a standard 24 x 80 screen).

Use the **ERRSFL** keyword if multiple messages may potentially be displayed when a record format is written to the display. The system displays an error subfile on the message line. This allows the user to page through all of the error and validity messages issued.

Message coding examples

IBM i



```

FIG1122.DSPPF
Line 1      Column 1      Replace          Browse
.....AAN01N02N03.....Functions+++++
000001     A               ERRSFL
000002     **+
000003     A               R FORMAT1      TEXT('SAMPLE DISPLAY FORMAT')
000004     A               1 5TIME
000005     A               1 64DATE EDTCDE(Y)
000006     **+
000007     A               DISCPCT        2 0I 4 28VALUES(5 10 15 20)
000008     A               CHKMSGID(MSG0300 AS06999/MSGF)
000009     **+
000010     A               QTYORD         6Y 0I 22 43
000011>>   A   30           ERREMSG('NO STOCK AVAILABLE' 30)
000012>>   A   31           ERREMSG('PARTIAL STOCK AVAILABLE' +
31)
000013     A   32           ERREMSGID(MSG0020 AS06999/MSGF1 +
32)
000014     A   33           ERREMSGID(MSG0021 AS06999/MSGF1 +
33 &MSGTXT)
000015
000016
000017
000018     **+
000019     A               MSGTXT        30A  P

```

© Copyright IBM Corporation 2012

Figure 9-16. Message coding examples

AS067.0

Notes:

&MSGTXT contains replacement text for the specified message. It must have a corresponding character field designated for usage P (column 40 - program to system field) in the record format in which it is to be used. This allows you to load the data field at program execution with variable data that is substituted into the body of your predefined message. Of course, your predefined message has substitution parameters defined so as to accept the data from this field.

An example of the ADDMSGD command that can be customized at program execution time follows:

```

ADDMMSGD MSGID(MSG0021)
MSGF(OL86999/MSGF1)
MSG('Item &1 is no longer available.
It has been replaced with &2.' )

```

Creating and using screen formats

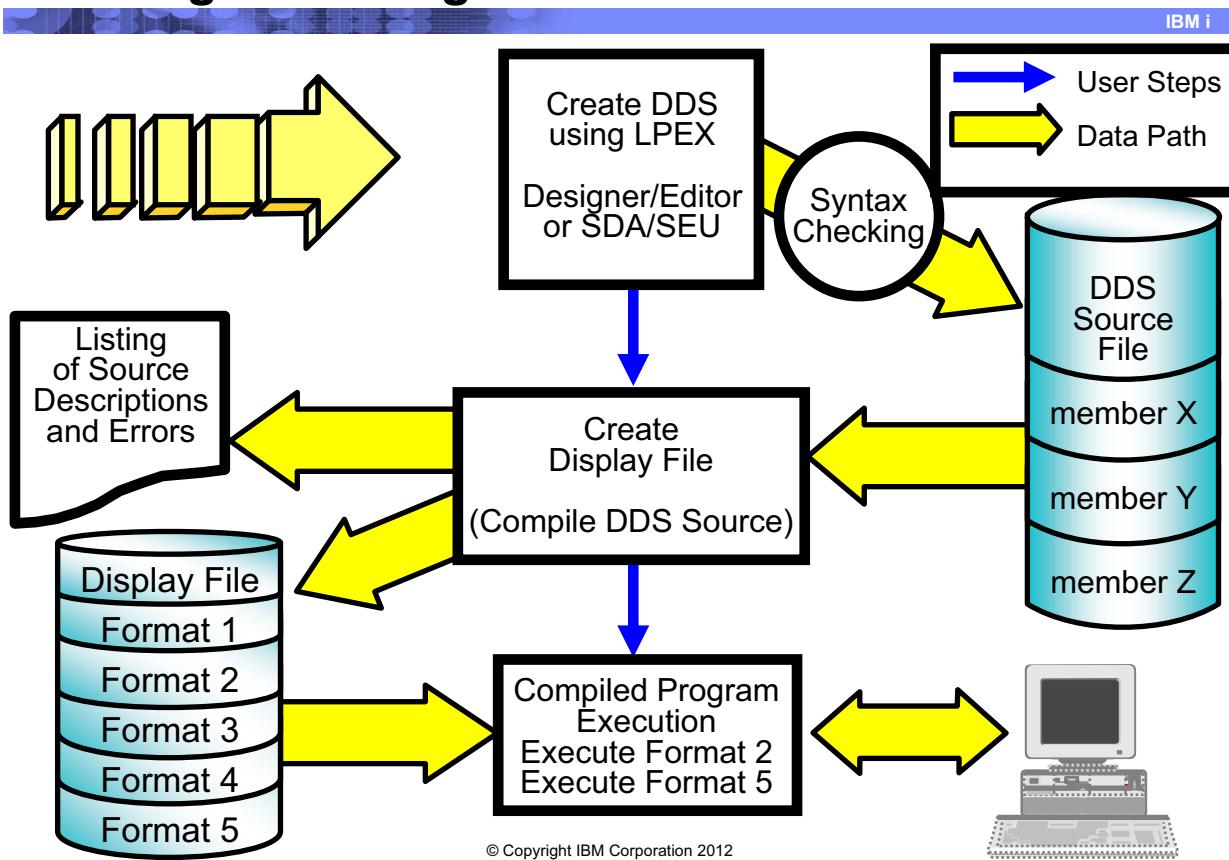


Figure 9-17. Creating and using screen formats

AS067.0

Notes:

These steps should look familiar. Display file members are DSPF type members. The SEU prompt for proper syntax checking while keying is DP. After creating the display file (object) with multiple formats, a program can be written and compiled to use them.

Tools

IBM i

- Screen Design AID
- IBM i utility to create screen images:
 - Layout screen on display (edit)
 - Define keywords at file, record and field levels
 - Change existing screen image
 - Save screen image as DDS source member
 - Create display file from generated DDS
 - Create display file menus
 - Can simplify screen design cycle
 - Good testing tool to test display file record formats
- Rational Developer for Power Screen Designer
 - PC-based GUI tool
 - Much more WYSIWYG than SDA
 - Generates DDS, can create DSPF
 - Good testing tool from WYSIWYG perspective

© Copyright IBM Corporation 2012

Figure 9-18. Tools

AS067.0

Notes:

SDA usage:

- Layout screen images using free form screen
- Define keywords at appropriate levels
- Modify and test existing display files
- Create menus

The command for starting an SDA session is:

```
STRSDA SRCFILE(AS06V7LIB/QDDSSRC) SRCMBR(DSPLYF_NAME)
```

Reference:

ADTS/400: Screen Design Aid

RPG screen operations

IBM i

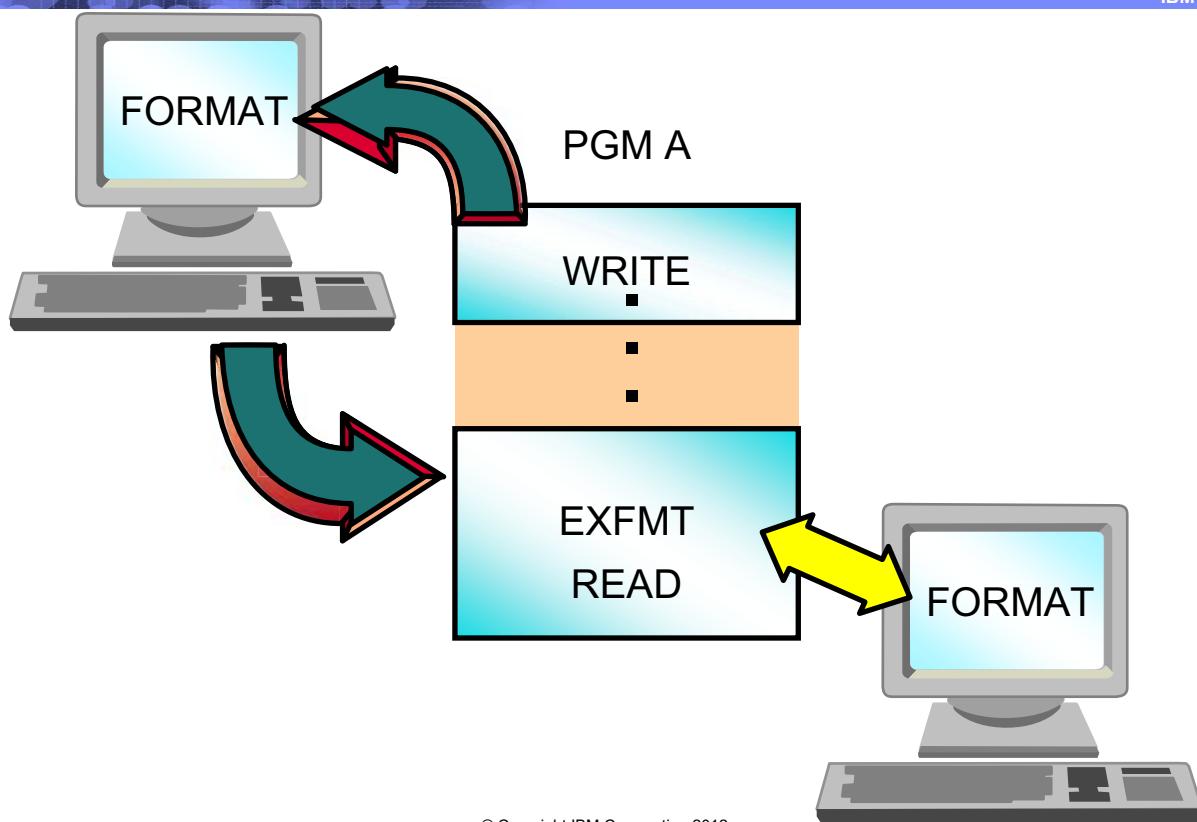


Figure 9-19. RPG screen operations

AS067.0

Notes:

The RPG IV WRITE operation sends formatted data from the program to the display device buffer.

The EXFMT operation performs a write, displays the data, and then performs a read when a function key or **Enter** is pressed.

The READ operation retrieves information from the display device buffer into the RPG IV program.

Inquiry example: Display design

IBM i

```

0....+....1....+....2....+....3....+....4....+....5....+....6....+....7...
-
-
-
-      0000000000          Item Inquiry          DD/DD/DD
-      (PGMNAM)
5
-
-
-
-      Item Number . . . . . : 66666
-      Description . . . . . : 00000000000000000000000000000000
10
-      Quantity on hand . . . : 6666666
-      Quantity on order . . . : 6666666
-
-
-      Supplier Number . . . . : 66666
15      Supplier Catalog No . . . : 0000000
-
-      ** NOTE ** The total quantity available is low (<20) - reorder
-
-
20      F3=Exit    F12=Cancel
-
-
-      Work screen for record DETAIL: Press Help for function keys.

```

© Copyright IBM Corporation 2012

Figure 9-20. Inquiry example: Display design

AS067.0

Notes:

This is the item inquiry display file (DETAIL) format that we covered earlier. Now we add the RPG IV program that interacts with the display file.

First, let us review the DDS for the display file.

Inquiry example: Display file DDS

```

.....RAN01N02N03.....Functions*****
000001 A REF(*LIBL/ITEM_PF)
000002>>1 A CA03(03 'Exit')
000003>>7 A INDARA
000004 ** R PROMPT
000005>>3 A PGHNAME 10A 0 3 7
000006 A 3 35'Item Inquiry'
000007 A COLOR(WHT)
000008 A 3 64DATE
000009>>4 A EDTCDE(Y)
000010 A 8 20'Item Number . . . . :'
000011 A 8 45
000012 A ITMNR R D I 10A 0 3 7
000013>>6 A 40
000014 A ERRMSG('Item not found on File - pl-
000015 A ease correct' 40)
000016 A 20'Press Enter to continue'
000017 A 21 7'F3=Exit'
000018 A COLOR(BLU)
000019>>3 A R DETAIL
000020>>2 A PGHNAME 10A 0 3 7
000021 A CA12(12 'Cancel')
000022 A 3 35'Item Inquiry'
000023 A COLOR(WHT)
000024 A 3 64DATE
000025 A EDTCDE(Y)
000026 A 8 20'Item Number . . . . :'
000027 A 8 45
000028 A 9 20'Description . . . . :'
000029 A 9 45
000030 A 11 20'Quantity on hand . . . . :'
000031 A 11 45EDTCDE(2)
000032 A DSPATR(HI)
000033 A 12 20'Quantity on order . . . . :'
000034 A 12 45EDTCDE(2)
000035 A DSPATR(HI)
000036 A 14 20'Supplier Number . . . . :'

**5 A 30

```

© Copyright IBM Corporation 2012

Figure 9-21. Inquiry example: Display file DDS

AS067.0

Notes:

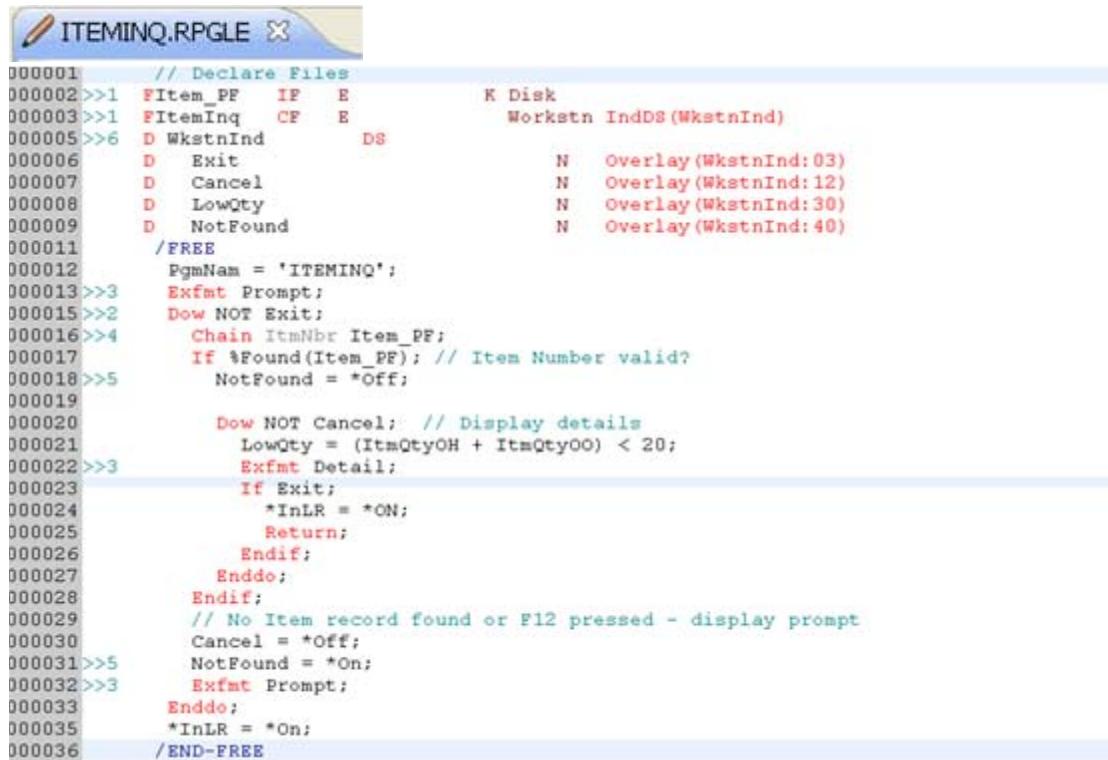
1. **F3**, CA03, allows the user to indicate that he/she is finished with the inquiry application. When **F3** is pressed, *in03 should be checked by the program.
2. **F12** allows a user to indicate that he/she wishes to inquire about a different item. When **F12** is pressed, *in12 should be checked by the program.
3. There are two record formats, PROMPT and DETAIL, in this member.
4. Note that the row and column numbers are coded in the line and position fields for each field. The special name DATE is used to retrieve information from the system that will display today's date.
5. Note the display attribute **HI** (highlight) is used to highlight quantity available and on order when the total quantity available is lower than desired. Indicator 30 is used to display the below minimum condition. Notice that we highlight the quantity fields in addition to displaying a message.
6. If Indicator 40 is set on in the RPG IV program because an item number on the PROMPT format is invalid, a message will be displayed on line 24 of the display.

Because we repeated the indicator next to the message, i5/OS (OS/400) will automatically set it off on the next input operation after the message has been displayed.

7. Because we are using named indicators in the RPG IV program, we must code the **INDARA** keyword at the file level.

Inquiry example: Item inquiry program

IBM i



```

000001 // Declare Files
000002 >>1 FItem_PF IF E K Disk
000003 >>1 FItemInq CF E Workstn IndDS(WkstnInd)
000005 >>6 D WkstnInd DS
000006 D Exit N Overlay(WkstnInd:03)
000007 D Cancel N Overlay(WkstnInd:12)
000008 D LowQty N Overlay(WkstnInd:30)
000009 D NotFound N Overlay(WkstnInd:40)
000011 /FREE
000012 PgmNam = 'ITEMINQ';
000013 >>3 Exfmt Prompt;
000015 >>2 Dow NOT Exit:
000016 >>4 Chain ItmNbr Item_PF;
000017 If %Found(Item_PF); // Item Number valid?
000018 >>5 NotFound = *Off;
000019
000020 Dow NOT Cancel; // Display details
000021 LowQty = (ItmQtyOH + ItmQtyOO) < 20;
000022 >>3 Exfmt Detail;
000023 If Exit;
000024 *InLR = *ON;
000025 Return;
000026 Endif;
000027 Enddo;
000028 Endif;
000029 // No Item record found or F12 pressed - display prompt
000030 Cancel = *Off;
000031 >>5 NotFound = *On;
000032 >>3 Exfmt Prompt;
000033 Enddo;
000035 *InLR = *On;
000036 /END-FREE

```

© Copyright IBM Corporation 2012

Figure 9-22. Inquiry example: Item inquiry program

AS067.0

Notes:

- Both Item_PF and ItemInq are externally described. The input and output specs for the display file will be copied into the program at RPG IV compile time. Note that the name used in the F-spec corresponds with the name of the display file. The file is designated a (C)ombined WORKSTN type of file for use with display station input and output.
- A common technique used to code interactive processing is to code the display of a screen in a loop using in this case, the DoW operation code conditioned by the F3 (exit) key.
- EXFMT is the RPG operation that writes then reads a display format. The EXFMT opcode writes the record format to the screen, waits for input from the user, and then reads the data from the display and passes it to RPG IV.
- If the CHAIN is successful, data from the DETAIL record format is displayed with the EXFMT operation.
- The only situation that should be controlled is when no record is found during the CHAIN (invalid key - record not found). We have coded an error message in the display

file that is conditioned on *In40. Note that in the program, *In40 (NotFound) is set on when there is no record found as a result of the CHAIN. If the CHAIN is unsuccessful, our error message is displayed when the display format is written. The user sees the error message, presses error reset to unlock the keyboard, and tries another customer number by pressing **F12** or presses **F3** to end the inquiry program.

- Also notice the structure we discussed in the printing unit again. Because DDS supports only numbered indicators, we must map the numbered indicators in DDS to our named indicators. To do this, we specify the **INDARA** keyword in the display file and the **IndDS** keyword in the RPG IV program.



Note

Because the field names from the database file and the field names from the display file match, there is no need to code EVALS to assign values from the data file to the display file.

The DDS for the *Item_PF* follows:

```
A*****
A* Item Master PF: ITEM_PF
A*****
A                               REF (DICTIONARY)
A                               UNIQUE
A     R ITEM_FMT               TEXT ('Item Master Record')
A         ITMNBR      R
A         ITMDESCR    R
A         ITMQTYOH    R
A         ITMQTYOO    R
A         ITMCOST      R
A         ITMPRICE    R
A         VNDNBR      R
A         ITMVNDCAT#R
A     K ITMNBR
A     K CUSTNO
```

Machine exercise: Coding an inquiry program

IBM i



© Copyright IBM Corporation 2012

Figure 9-23. Machine exercise: Coding an inquiry program

AS067.0

Notes:

Perform the coding an inquiry program machine exercise.

Checkpoint

IBM i

1. True or False: For interactive inquiry, display record formats must use the same field names as the database file being processed.

2. Which of the following are used to validate data entry fields?
 - a. CHECK
 - b. DSPATR
 - c. RANGE
 - d. All of the above

3. True or False: An inquiry program requires at least two display record formats.

© Copyright IBM Corporation 2012

Figure 9-24. Checkpoint

AS067.0

Notes:

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe the properties of a display file
- Create display files using DDS
- Use Exfmt and Write RPG IV operation codes with display files
- Code inquiry RPG IV programs to support one display file

© Copyright IBM Corporation 2012

Figure 9-25. Unit summary

AS067.0

Notes:

Unit 10. What's next?

What this unit is about

This unit describes the activities that students can perform to apply the knowledge gained in this class and to prepare to attend the follow-on courses:

- AS07/AS070: *RPG IV Programming Intermediate Workshop for IBM i*
- AS10/AS100: *RPG IV Programming Advanced Workshop for IBM i*

This is the first in a series of three courses designed to help you to become a skilled RPG IV programmer. This unit discusses what is taught in these follow-on courses and why it is important for you to attend.

What you should be able to do

After completing this unit, you should be able to:

- List the prerequisite skills needed prior to attending the Intermediate and Advanced RPG IV courses
- List the topics of the next course

Unit objectives

IBM i

After completing this unit, you should be able to:

- List the prerequisite skills needed prior to attending the intermediate and advanced RPG IV courses
- List the topics of the next course

© Copyright IBM Corporation 2012

Figure 10-1. Unit objectives

AS067.0

Notes:

What have you learned?

IBM i

- Write simple RPG IV Version 7 programs to produce reports
- Write simple RPG IV Version 7 inquiry programs that interact with displays
- Use the editor to enter and modify source programs compile RPG IV programs
- Review compilation listing, and find and correct compilation errors
- Maintain existing applications written in the RPG IV Version 7 language
- Use the debugger tool to determine the cause of incorrect results
- Use many popular RPG IV built-in functions
- And much more!

© Copyright IBM Corporation 2012

Figure 10-2. What have you learned?

AS067.0

Notes:

You have worked very hard over the past four days. We hope that you think that you have accomplished a lot by attending this class.

You should now be able to write working RPG IV programs to produce reports and to handle inquiry displays.

As we said at the beginning of this class, it is the first in a series of three courses that, when followed in sequence, will provide the skills necessary to be proficient RPG IV programmer.

You have learned a lot about using the RPG IV language to write applications. To maximize the new skills that you now have, you must apply this knowledge *on the job*.

To do list

IBM i

- Get hands on ASAP!
- Read the reference manuals.
- Read the ITSO Redbook *Who Knew You Could Do That..?*
- Read articles in trade magazines and on the web.
- Prepare to attend follow-on courses.
- Join your local i users group.
- Participate in www.midrange.com forum lists.

© Copyright IBM Corporation 2012

Figure 10-3. To do list

AS067.0

Notes:

Topics list for next course

IBM i

RPG IV Intermediate Programming Workshop for IBM i

- Managing display record formats
 - Using OVERLAY/OVRDTA/PUTOVR/OVRATR
- Using subfile displays:
 - Simple inquiries
 - Subfile search
 - Other subfile operations
 - Subfile record maintenance
- Using arrays
- Using data structures and data areas
- Managing exceptions and handling errors:
 - % Error and E-Extender
 - Monitor groups
- Processing date and time data
- Code prototypes to define parameters to call programs and procedures
- Code subprocedures
- Introduction to ILE

© Copyright IBM Corporation 2012

Figure 10-4. Topics list for next course

AS067.0

Notes:

You should plan to attend the second course in this series of three courses next. Some on-the-job experience is strongly recommended before attending this course.

Another course that you should consider attending is RN500, IBM i RPG Development with IBM Rational Developer for Power Systems Software V8. This course teaches how to use the complete LPEX, Eclipse-based editor.

Useful references and web sites

IBM i

- RPG IV reference manuals
- *Who Knew You Could Do That with RPG IV?* (ITSO Redbook)
- <http://www.iprodeveloper.com>
- <http://www.midrange.com>
 - <http://lists.midrange.com/mailman/listinfo/rpg400-l>
 - <http://lists.midrange.com/mailman/listinfo/code400-l>
- <http://www-03.ibm.com/certify>
- <http://www.midrangenews.com>

© Copyright IBM Corporation 2012

Figure 10-5. Useful references and web sites

AS067.0

Notes:

This is a list of some useful documentation as well as some web sites that you should explore. Of particular value are the forum lists, sponsored by www.midrange.com.

Unit summary

IBM i

Having completed this unit, you should be able to:

- List the prerequisite skills needed prior to attending the intermediate and advanced RPG IV courses
- List the topics of the next course

© Copyright IBM Corporation 2012

Figure 10-6. Unit summary

AS067.0

Notes:

Appendix A. RPG IV style guide

Reprinted from '*RPG IV Jump Start*' (4th ed) by Bryan Meyers (www.bmeyers.net), with permission of the author.

Chapter 15

“Rethinking RPG Standards”

Professional programmers appreciate the importance of standards in developing programs that are readable, understandable, and maintainable. The issue of programming style goes beyond any one language, but the introduction of the RPG IV syntax demands that you re-examine standards of RPG style. Now would be a great time to begin thinking about how your own shop’s RPG standards might change as you move into application development with RPG IV. Make no mistake: Your existing RPG standards are now obsolete. This chapter presents some issues to think about before you start feeling your way around RPG IV.

Use comments judiciously

Good programming style can serve a documentary purpose in helping others understand the source code. If you practice good code-construction techniques, you will find that “less is more” when it comes to commenting the source. Too many comments are as bad as too few.

Use comments to clarify, not echo, your code. Comments that merely repeat the code add to a program’s bulk, but not to its value. In general, you should use comments for just three purposes:

- To provide a brief program or procedure summary
- To give a title to a subroutine, procedure, or other section of code
- To explain a technique that is not readily apparent by reading the source

Always include a brief summary at the beginning of a program or procedure. This prologue should include the following information:

- A program or procedure title
- A brief description of the program’s or procedure’s purpose
- A chronology of changes that includes the date, programmer name, and purpose of each change
- A summary of indicator usage
- A description of the procedure interface (the return value and parameters)
- An example of how to call the procedure

Use consistent marker line comments to divide major sections of code. For example, you should definitely section off with lines of dashes (-) or asterisks (*) the declarations, the main procedure, each subroutine, and any subprocedures. Identify each section for easy reference.

Use blank lines to group related source lines and make them stand out. In general, you should use completely blank lines instead of blank comment lines to group lines of code, unless you are building a block of comments. Use only one blank line, though; multiple consecutive blank lines make your program hard to read.

Avoid right-hand end-line comments in columns 81–100. Right-hand comments tend simply to echo the code, can be lost during program maintenance, and can easily become out of sync with the line they comment. If a source line is important enough to warrant a comment, it's important enough to warrant a comment on a separate line. If the comment merely repeats the code, eliminate it entirely.

Centralize declarations

With RPG IV, we finally have an area of the program source in which to declare all variables and constants associated with the program. The D-specs organize all your declarations in one place.

RPG IV still supports the *LIKE DEFINE opcode, along with Z-ADD, Z-SUB, MOVEx, and CLEAR, to define program variables. But for ease of maintenance as well as program clarity, you will want to dictate a standard that consolidates all data definition, including work fields, in D-specs.

Declare all variables within D-specs. Except for key lists and parameter lists, do not declare variables in C-specs -- not even using *LIKE DEFINE. Define key lists and parameter lists in the first C-specs of the program, before any executable calculations. Use a prototype definition instead of an *ENTRY PLIST.

Whenever a literal has a specific meaning, declare it as a named constant in the D-specs. This practice helps document your code and makes it easier to maintain. One obvious exception to this rule is the allowable use of 0 and 1 when they make perfect sense in the context of a statement. For example, if you are going to initialize an accumulator field or increment a counter, it's fine to use a hard-coded 0 or 1 in the source.

Indent data item names to improve readability and document data structures. Unlike many other RPG entries, the name of a defined item need not be left-justified in the D-specs; take advantage of this feature to help document your code:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++ETDsFrom+++To/L+++IDc.Functions+++++
D ErrMsgDSDS      DS
D   ErrPrefix          3
D   ErrMsgID           4
D   ErrMajor            2    OVERLAY (ErrMsgID:1)
D   ErrMinor            2    OVERLAY (ErrMsgID:3)
```

Use length notation instead of positional notation in data structure declarations.

D-specs let you code fields either with specific from and to positions or simply with the length of the field. To avoid confusion and to better document the field, use length notation consistently. For example, code:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++ETDsFrom+++To/L+++IDc.Functions+++++
D RtnCode      DS
D PackedNbr     15P 5
```

instead of

```
D RtnCode      DS
D PackedNbr     1      8P 5
```

Use positional notation only when the actual position in a data structure is important. For example, when coding the program status data structure, the file information data structure, or the return data structure from an application programming interface (API), you would use positional notation if your program ignores certain positions leading up to a field or between fields. Using positional notation is preferable to using unnecessary “filler” variables with length notation:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++ETDsFrom+++To/L+++IDc.Functions+++++
D APIRtn      DS
D PackedNbr     145     152P 5
```

In this example, to better document the variable, consider overlaying the positionally declared variable with another variable declared with length notation:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++ETDsFrom+++To/L+++IDc.Functions+++++
D APIRtn      DS
D Pos145       145     152
D PackNbr      15P 5  OVERLAY (Pos145)
```

When defining overlapping fields, use the **OVERLAY** keyword instead of positional notation. Keyword **OVERLAY** explicitly ties the declaration of a child variable to that of its parent. Not only does **OVERLAY** document this relationship, but if the parent moves elsewhere within the program code, the child follows.

If your program uses compile-time arrays, use the **CTDATA form to identify the compile-time data. This form effectively documents the identity of the compile-time data, tying the data at the end of the program to the array declaration in the D-specs. The ****CTDATA** syntax also helps you avoid errors by eliminating the need to code compile-time data in the same order in which you declare multiple arrays.

Expand naming conventions

Perhaps the most important aspect of programming style deals with the names you give to data items (for example, variables, named constants) and routines. Establish naming conventions that go beyond the traditional six characters, to fully identify variables and other identifiers. Those extra characters can make the difference between program code and a program description.

When naming an item, be sure the name fully and accurately describes the item. The name should be unambiguous, easy to read, and obvious. Although you should exploit RPG IV's allowance for long names, do not make your names too long to be useful. Name lengths of 10 to 14 characters are usually sufficient, and longer names may not be practical in many specifications. When naming a data item, describe the item; when naming a subroutine or procedure, use a verb/object syntax (similar to a CL command) to describe the process. Maintain a dictionary of names, verbs, and objects, and use the dictionary to standardize your naming conventions.

When coding an RPG symbolic name, use mixed case to clarify the named item's meaning and use. RPG IV lets you type your source code in upper- and lowercase characters. Use this feature to clarify named data. For RPG-reserved words and operations, use all uppercase characters.

Avoid using special characters (for example, @, #, \$) when naming items.

Although RPG IV allows an underscore (_) within a name, you can easily avoid using this noise character if you use mixed case intelligently.

Write indicator-less code

Historically, indicators have been an identifying characteristic of the RPG syntax, but with RPG IV they are fast becoming relics of an earlier era. Reducing a program's use of indicators may well be the single most important thing you can do to improve the program's readability.

Use indicators as sparingly as possible; go out of your way to eliminate them. At Version 5, RPG completely eliminates the need for conditioning indicators and resulting indicators, and does not support them in any free-form specifications. At prior release, the only indicators present in a program should be resulting indicators for opcodes that

absolutely require them (for example, LOOKUP). Whenever possible, use built-in functions (BiFs) instead of indicators. Remember you can indicate file exception conditions with error-handling BiFs (for example, %EOF, %ERROR, %FOUND) and an E operation extender to avoid using indicators.

If you must use indicators, name them. V4R2 supports a Boolean data type (N) that serves the same purpose as an indicator. You can use the INDDS keyword with a display-file specification to associate a data structure with the indicators for a display or printer file; you can then assign meaningful names to the indicators.

Use the EVAL opcode with *Inxx and *ON or *OFF to set the state of indicators. Do not use the SETON or SETOFF operation, and never use MOVEA to manipulate multiple indicators at once.

Use indicators only in close proximity to the point where your program sets their condition. For example, it is bad practice to set an indicator and not test it until several pages later. If it is not possible to keep the related actions (setting and testing the indicator) together, move the indicator value to a meaningful variable instead.

Do not use conditioning indicators -- ever. If a program must conditionally execute or avoid a block of source, explicitly code the condition with a structured comparison opcode, such as IF. If you are working with old System/36 code, get rid of the blocks of conditioning indicators in the source. The Version 5 free-form specification does not support conditioning indicators.

Include a description of any indicators you use. It is especially important to document indicators whose purpose is not obvious by reading the program, such as indicators used to communicate with display or printer files or the U1–U8 external indicators, if you must use them.

Structured programming techniques

Give those who follow you a fighting chance to understand how your program works by implementing structured programming techniques at all times. The IF, DOU, DOW, FOR, and WHEN opcodes are positively elegant. Banish IFxx, DOUxx, DOWxx, and WHxx from your RPG IV code forever. By the way, you would never use indicators to condition structured opcodes, would you? Good!

Do not use GOTO, CABxx, or COMP. Instead, substitute a structured alternative, such as nested IF statements, or status variables to skip code or to direct a program to a specific location. To compare two values, use the structured opcodes IF and ELSE. To perform loops, use DOU, DOW, and FOR. Never code your loops by comparing and branching with COMP (or even IF) and GOTO. Employ ITER to repeat a loop iteration, and use LEAVE for premature exits from loops, or LEAVESR to prematurely exit subroutines.

Do not use the obsolete IFxx, DOUxx, DOWxx, or WHxx opcodes. The newer forms of these opcodes (IF, DOU, DOW, and WHEN) support free-format expressions, making those alternatives more readable. In general, if an opcode offers a free-format

alternative, use it. This rule applies to the DO opcode as well; the free-format FOR operation is usually a better choice, if you are at V4R4 or later.

Perform multipath comparisons with SELECT/WHEN/OTHER/ENDSL. Deeply nested IFxx/ELSE/ENDIF code blocks are hard to read and result in an unwieldy accumulation of ENDIFs at the end of the group. Do not use the obsolete CASxx opcode; instead, use the more versatile SELECT/WHEN/OTHER/ENDSL construction.

Always qualify END opcodes. Use ENDIF, ENDDO, ENDFOR, ENDSDL, or ENDCS as applicable. This practice can be a great help in deciphering complex blocks of source.

Avoid programming tricks and hidden code. Such maneuvers are not so clever to someone who does not know the trick. If you think you must add comments to explain how a block of code works, consider rewriting the code to clarify its purpose. Use of the obscure bit-twiddling opcodes (BITON, BITOFF, MxxZO, TESTB, and TESTZ) may be a sign that your source needs updating.

Modular programming techniques

The RPG IV syntax, along with the i5 (iSeries)'s Integrated Language Environment (ILE), encourages a modular approach to application programming. Modularity offers a way to organize an application, facilitate program maintenance, hide complex logic, and efficiently reuse code wherever it applies.

Use RPG IV's prototyping capabilities to define parameters and procedure interfaces.

Prototypes (PR definitions) offer many advantages when you are passing data between modules and programs. For example, they avoid runtime errors by giving the compiler the ability to check the data type and number of parameters. Prototypes also let you code literals and expressions as parameters, declare parameter lists (even the *ENTRY PLIST) in the D-specs, and pass parameters by value and by read-only reference, as well as by reference.

Store prototypes in /COPY members. For each module, code a /COPY member containing the procedure prototype for each exported procedure in that module. Then include a reference to that /COPY module in each module that refers to the procedures in the called module. This practice saves you from typing the prototypes each time you need them and reduces errors.

Include constant declarations for a module in the same /COPY member as the prototypes for that module. If you then reference the /COPY member in any module that refers to the called module, you have effectively globalized the declaration of those constants.

Use IMPORT and EXPORT only for global data items. The IMPORT and EXPORT keywords let you share data among the procedures in a program without explicitly passing the data as parameters -- in other words, they provide a hidden interface between procedures. Limit use of these keywords to data items that are truly global in the program -- usually values that are set once and then never changed.

Hone Your Modular Programming Skills

Creating effective modular programs requires two things: an architecture that offers high-performance calls with a uniform runtime model for high-level languages, and a language (or languages) capable of harnessing that architecture. With ILE, the i (System i) has the necessary program/procedure call environment, and with RPG IV, the RPG programmer has a language that can exploit that environment. RPG's support for subprocedures (covered in Chapter 14) is an especially welcome tool as you work on modularizing applications and creating repositories of reusable code.

As you write new ILE applications, you will probably find yourself using program and/or procedure calls far more often than in the past. Consider a date conversion routine required by six fields in a one-million-record file. If you were to add that routine to an RPG/400 program as an external call, the six million times the routine is called could significantly degrade performance.

In the past, your only real option for this scenario was to code the routine as an internal subroutine. With ILE's static call mechanisms, though, you can code it as a subprocedure or a called program, with the effective performance of a subroutine call. This means you can rethink the structure of all your monolithic, inline RPG programs and achieve a substantial level of modularity without a trade-off in speed.

As you increasingly exploit the benefits of modular procedure calls, you need to know how best to carve up your code. Robust modular programming is not achieved by butchering old code into random chunks; rather, it requires filleting code into planned, manageable, reusable pieces. You need to engineer each module so that it does one thing well and so that your call interfaces are simple, yet flexible enough to do the job.

Free the factor 2

You can mix and match RPG III style, RPG IV fixed-form style, and RPG IV free-form style in your C-specs, but the result is inconsistent and difficult to read. Take full advantage of the more natural order and expanded space afforded by the free-form specification (or, previous to Version 5, the extended Factor 2). When you are coding loops and groups, you find that the code looks and feels better in free-form.

Use free-form expressions (or EVAL) wherever possible. Instead of Z-ADD and Z-SUB, use assignment expressions. Use expressions for any arithmetic in your program. Instead of CAT and SUBST, use string expressions. Use expressions to set indicators (if you need them).

But do not completely abandon columnar alignment as a tool to aid readability in expressions. Especially when an expression must continue onto subsequent lines, align the expression to make it easier to understand (refer to Chapter 7 for examples).

Character string manipulation

IBM has greatly enhanced RPG IV's ability to easily manipulate character strings. Many of the tricks you had to use with earlier versions of RPG are now obsolete. Modernize your source by exploiting these new features.

Use a named constant to declare a string constant instead of storing it in an array or table. Declaring a string (such as a CL command string) as a named constant lets you refer to it directly instead of forcing you to refer to the string through its array name and index. Use a named constant to declare any value that you do not expect to change during program execution.

Avoid using arrays and data structures to manipulate character strings and text. Use the new string manipulation opcodes, built-in functions, or both instead.

Use free-form assignment expressions whenever possible for string manipulation. When used with character strings, EVAL is usually equivalent to a MOVEL(P) opcode. When you do not want the result to be padded with blanks, use %SUBST or %REPLACE functions.

Use variable-length fields to simplify string handling. Use variable-length fields as CONST or VALUE parameters to every string-handling subprocedure, as well as for work fields. Not only does the code look better (eliminating the %TRIM function, for example), but it is also faster than using fixed-length fields. For example, code:

```
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  
DName++++++ETDsFrom+++To/L+++IDc.Functions++++++  
D QualName      S          33    VARYING  
D Library       S          10    VARYING  
D File          S          10    VARYING  
D Member        S          10    VARYING  
  
/FREE  
  QualName = Library + '/' + File + '(' + Member + ')';  
/END-FREE
```

instead of

```
D QualName      S          33  
D Library       S          10  
D File          S          10  
D Member        S          10  
  
/FREE  
  QualName = %TRIM/Library) + '/' + %TRIM(File)  
  + '(' + %TRIM(Member) + ')';  
/END-FREE
```

Avoid obsolescence

RPG is an old language. After 30 years, many of its original, obsolete features are still available. Do not use them.

Do not sequence program line numbers in columns 1–5. Chances are you never again drop that deck of punched cards, so the program sequence area is unnecessary. In RPG IV, the columns are commentary only. You can use them to identify changed lines in a program or structured indentation levels, but be aware that these columns may be subject to the same hazards as right-hand comments.

Avoid program-described files. Instead, use externally defined files whenever possible.

If an opcode offers a free-format syntax, use it instead of the fixed-format version. Opcodes to avoid include CABxx, CASxx, CAT, DO (at V4R4), DOUxx, DOWxx, IFxx, and WHxx. At Version 5, avoid any operation code that the free-form specification does not support (Figure 6.4 in Chapter 6 lists them).

If a BiF offers the same function as an opcode, use the BiF instead of the opcode.

With some opcodes, you can substitute a built-in function for the opcode and use the function within an expression. At V4R1, the SCAN and SUBST opcodes have virtually equivalent built-in functions, %SCAN and %SUBST. In addition, you can usually substitute the concatenation operator (+) in combination with the %TRIMx BiFs in place of the CAT opcode. The free-format versions are preferable if they offer the same functionality as the opcodes.

Use the date operations to operate on dates. Get rid of the clever date and time routines that you have gathered and jealously guarded over the years. The RPG IV operation codes and built-in functions are more efficient, clearer, and more modern. Even if your database includes dates in legacy formats, you can use the date opcodes to manipulate them.

Shun obsolete opcodes. In addition to the opcodes mentioned earlier, some opcodes are no longer supported or have better alternatives:

- CALL, CALLB: The prototyped calls (CALLP or a function call) are just as efficient as CALL and CALLB and offer the advantages of prototyping and parameter passing by value. Neither CALL nor CALLB can accept a return value from a procedure.
- DEBUG: With i5/OS (OS/400)'s advanced debugging facilities, this opcode is no longer supported.
- DSPLY: You should use display file I/O to display information or to acquire input.
- FREE: This opcode is no longer supported.
- PARM, PLIST: If you use prototyped calls, these opcodes are no longer necessary.

Miscellaneous guidelines

Here is an assortment of other style guidelines that can help you improve your RPG IV code.

In all specifications that support keywords, observe a one-keyword-per-line limit.

Instead of spreading multiple keywords and values across the entire specification, your program will be easier to read and let you more easily add or delete specifications if you limit each line to one keyword, or at least to closely related keywords (for example, DATFMT and TIMFMT).

Begin all H-spec keywords in column 8, leaving column 7 blank. Separating the keyword from the required H in column 6 improves readability.

Relegate mysterious code to a well-documented, well-named procedure. Despite your best efforts, on extremely rare occasions you simply are not able to make the meaning of a chunk of code clear without extensive comments. By separating such heavily documented, well-tested code into a procedure, you save future maintenance programmers the trouble of deciphering and dealing with the code unnecessarily.

Final advice

Sometimes good style and efficient runtime performance do not mix. Wherever you face a conflict between the two, choose good style. Hard-to-read programs are hard to debug, hard to maintain, and hard to get right. Program correctness must always win out over speed. Keep in mind these admonitions from Brian Kernighan and P.J. Plauger's *The Elements of Programming Style*:

- Make it right before you make it faster.
- Keep it right when you make it faster.
- Make it clear before you make it faster.
- Do not sacrifice clarity for small gains in efficiency.

Appendix B. Checkpoint solutions

Unit 1, "RPG IV introduction"

Solutions for Figure 1-18, "Checkpoint (1 of 2)," on page 1-29

Checkpoint solutions (1 of 2)

IBM i

1. True or False: Today's RPG IV programs must use ILE features like binding and subprocedures.
The answer is False.
2. In order to create a program from an RPGLE type source member, which of the following answers is correct?
 - a. CRTRPGPGM
 - b. CRTBNDRPG
 - c. CRTRPGMOD and CRTPGM
 - d. CRTRPGMOD
 The answer is CRTBNDRPG and/or CRTRPGMOD and CRTPGM.
3. The (blank) compiler directives specify the beginning and end of a free-form calculation specification block.
 - a. /DEFINE, /UNDEFINE
 - b. /BEGIN, /END-FREE
 - c. /FREE, /END-FREE
 - d. /FREE, /END
 The answer is /FREE, /END-FREE.
4. True or False: In order to include code from another source member in your RPG IV program, you would use the /COPY compiler directive.
The answer is True.

© Copyright IBM Corporation 2012

Solutions for Figure 1-19, "Checkpoint (2 of 2)," on page 1-30**Checkpoint solutions (2 of 2)**

IBM i

5. Number the following RPG specification types in correct sequence.

- 1 H Specifications
- 2 F Specifications
- 3 D Specifications
- 4 I Specifications
- 5 C Specifications
- 6 O Specifications
- 7 P Specifications

The answer is 1 H Specifications, 2 F Specifications, 3 D Specifications, 4 I Specifications, 5 C Specifications, 6 O Specifications, 7 P Specifications.

6. A source physical file for RPG IV source members has a (blank) byte record length?

- a. 80
- b. 92
- c. 100
- d. 112

The answer is 112.

© Copyright IBM Corporation 2012

Unit 2, "Coding specifications for RPG IV"

Solutions for Figure 2-29, "Checkpoint," on page 2-38

Checkpoint solutions

IBM i

1. True or False: An H-spec MUST be coded into every RPG IV source member.

The answer is False.

2. Which of the following are functions of the F-spec in RPG IV?
 - a. To reflect the status of the last input operation
 - b. To designate the file as Input or Output
 - c. To specify a file as described externally or in the program

The answer is To designate the file as Input or Output and To specify a file as described externally or in the program.

3. True or False: The D-spec is NOT needed to define fields from an externally described file.

The answer is True.

© Copyright IBM Corporation 2012

Unit 3, "Data representation and definition"

Solutions for Figure 3-20, "Checkpoint," on page 3-27

Checkpoint solutions

IBM i

1. True or False: Named indicators are an effective alternative to numeric indicators as they allow you to use a meaningful name.

The answer is True.

2. Data is defined in an RPG IV program with:
 - a. Input specifications and program described files
 - b. Externally described files
 - c. D-specification definitions
 - d. All of the above

The answer is All of the above.

3. True or False: The D-spec is NOT needed to define fields from an externally described file.

The answer is True.

© Copyright IBM Corporation 2011

Unit 4, "Manipulating data in calculations"

Solutions for Figure 4-39, "Checkpoint," on page 4-52

Checkpoint solutions

IBM i

- Given the following values for the variables, what will be the value of X after the EVAL statement?

A = 1, B = 2, C = 3

EVAL X = A * B + C ** B - 1

The answer is 10.

- Given the same initial values for the variables A, B and C, what will be the value of X after this EVAL statement?

EVAL X = A * B + C ** (B - 1)

The answer is 5.

- Given the same initial values for the variables A, B and C, and given that *IN03 is ON, what will be the value of *INLR after this EVAL statement?

EVAL *INLR = *IN03 AND B > A OR C > A + B

The answer is *INLR = '1'.

© Copyright IBM Corporation 2012

Unit 5, "Printing from an RPG IV program"

Solutions for Figure 5-20, "Checkpoint," on page 5-31

Checkpoint solutions

IBM i

1. True or False: Exact line numbers can be used to position a particular field or constant on the report.

The answer is True.

2. Which of the following DDS facilities are available when defining printer files?

- a. Continuation lines
- b. Edit codes and words
- c. Field definition referencing
- d. All of the above

The answer is All of the above.

3. The three levels at which DDS printer file keywords can be specified are:

The answer is file, record, field.

© Copyright IBM Corporation 2012

Unit 6, "Using the debugger"

Solutions for Figure 6-24, "Checkpoint," on page 6-32

Checkpoint solutions

IBM i

1. True or [False](#): If all the shipped system defaults are used when creating RPG IV modules and programs, the programs will be able to use source view debugging.

The answer is [False](#).

2. To change the value of a variable during a source or listing view debug session, you would:

- a. Press **F11** and key the new value over the previous one
- b. Use CHGPGMVAR command from the debug command line
- c. Press **F21** to get a system command line and key CHGPGMVAR
- d. Use the EVAL command on the debug command line

The answer is [Use the EVAL command on the debug command line.](#)

3. Which of the following parameter values for debug view (DBGVIEW) would normally result in the largest total program or module size?

- a. *LIST
- b. *COPY
- c. *SOURCE
- d. *STMT
- e. [*ALL](#)

The answer is [*ALL](#).

© Copyright IBM Corporation 2012

Unit 7, "Structured programming and subroutines"

Solutions for Figure 7-28, "Checkpoint," on page 7-30

Checkpoint solutions

IBM i

1. True or False: The parameter list in both the calling and the called programs must match field name by field name.
The answer is False.
2. Which of the following opcodes does not have a corresponding ENDxx statement to terminate the group?
 - a. DO
 - b. IF
 - c. PLIST
 - d. BEGSR
 - e. SELECTThe answer is PLIST.
3. True or False: An RPG subroutine can be called from another program outside of where it is written by coding an EXSR.
The answer is False.

© Copyright IBM Corporation 2012

Unit 8, "Accessing the DB2 database using RPG IV"

Solutions for Figure 8-47, "Checkpoint," on page 8-55

Checkpoint solutions

IBM i

1. True or False: The keyword levels for logical files are file, record, field and key.

The answer is False.

2. Database files can be specified (position 17 of the F-specification) as which of the following types?

- a. (I)nput
- b. (O)utput
- c. (C)ombined
- d. (U)pdate

The answers are (I)nput, (O)utput, and (U)pdate.

3. True or False: SETLL and SETGT are used to position the file cursor prior to sequential read operations.

The answer is True.

© Copyright IBM Corporation 2012

Unit 9, "Coding inquiry programs"

Solutions for Figure 9-24, "Checkpoint," on page 9-29

Checkpoint solutions

IBM i

1. True or False: For interactive inquiry, display record formats must use the same field names as the database file being processed..

The answer is False.

2. Which of the following are used to validate data entry fields?
 - a. CHECK
 - b. DSPATR
 - c. RANGE
 - d. All of the above

The answers are CHECK and RANGE.

3. True or False: An inquiry program requires at least two display record formats.

The answer is False.

© Copyright IBM Corporation 2012

Unit 10, "What's next?"

No checkpoint solutions in this unit.

Bibliography

Manuals:

IBM Rational Development Studio for i ILE RPG Language Reference

IBM Rational Development Studio for i ILE RPG Programmer's Guide

Other reference manuals on CD or information center:

DDS Reference: *Physical and Logical Files*

Display Files

Printer Files

ILE Concepts

ILE Application Development Example

ADTS/400: Screen Design Aid

Web URLs:

<http://www.ibm.com> IBM's Internet Connection web site

<http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/index.jsp>

IBM i Information Center (V7R1)

<http://www.redbooks.ibm.com> Internet site for ITSO Redbooks

<http://www.ibm.com> IBM's Internet Connection web site

<http://www.rfc-editor.org/> Internet "Requests for Comments" Editor

<http://www.midrange.com> Midrange lists and forums

<http://www.iprodeveloper.com/> Home of System i Network and Magazine

CD-ROMs:

SK3T-4091 *IBM i Information Center (Version 7)*

ITSO Redbooks

SG24-5402 *Who Knew You Could Do That with RPG IV?*

Other useful web sites

Certification: <http://www.ibm.com/certify>

MidRange News i Community <http://www.midrangenews.com/>

Articles:

News/400 (June 98) "The Essential RPG IV Style Guide" (updated summer 2000)

News/400 (Sept 00) "RPG IV Style Guide: Error Recovery"

News/400 (Nov 00) "RPG IV: Free Format and More"

IBM
®