Dear Reader: Thank you for downloading this free book from Brian W. Kelly's finished book catalog. I finished the book titled System/I Pocket RPG & RPGIV Guide https://letsgopublish.com/technology/rpg.pdf in October 2016. Concepts, Coding Examples, & Exercises.

Most of my books had previously been published on Amazon.

Click below if you would like to donate to help the free book cause:
https://www.letsgopublish.com/books/donate.pdf

Enjoy!

LETS GO RPG & RPGIV!

The System i Pocket RPG & RPGIV Guide

AS/400 and **System i** RPG & RPGIV Concepts, Coding Examples & Exercises

– A Comprehensive Textbook of up-to-date Information plus RPG & RPGIV examples for the new and experienced AS/400 and ***System i*** *Application Developer –*

BRIAN W. KELLY

The System i
Pocket RPG & RPGIV Guide

AS/400 and System i RPG Concepts, Coding Examples & Exercises

– A Comprehensive Book of updated Information and RPG examples for the new and experienced AS/400 and System i Application Developer –

BRIAN W. KELLY

Copyright © 2006, Brian W. Kelly
The System i Pocket RPG & RPGIV Guide Author Brian W. Kelly

All rights reserved: No part of this book may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopying, recording, scanning, faxing, or by any information storage and retrieval system, without permission from the publisher, LETS GO PUBLISH, in writing.

Disclaimer: Though judicious care was taken throughout the writing and the publication of this work that the information contained herein is accurate, there is no expressed or implied warranty that all information in this book is 100% correct. Therefore, neither LETS GO PUBLISH, nor the author accepts liability for any use of this work.

Trademarks: A number of products and names referenced in this book are trade names and trademarks of their respective companies. For example, System i is a trademark of IBM Corporation.

Referenced Material *: The information in this book has been obtained through personal and third party observations, interviews, and copious research and analysis. Where unique information has been provided or extracted from other sources, those sources are acknowledged within the text of the book itself. Thus, there are no formal footnotes nor is there a bibliography section. Any picture that does not have a source was taken from various sites on the Internet with no credit attached. If any picture owner would like credit in the next printing, please email the publisher.*

Published by: LETS GO PUBLISH!
Joseph McDonald, Publisher
P.O Box 425
Scranton, PA 18503
jmac@letsgopublish.com
www.letsgopublish.com
Library of Congress Copyright Information Pending

Book Cover Design by Michele Thomas; Editing by Brian P. Kelly

ISBN Information: The International Standard Book Number (ISBN) is a unique machine-readable identification number, which marks any book unmistakably. The ISBN is the clear standard in the book industry. 159 countries and territories are officially ISBN members. The Official ISBN for this book is:

0-9745852-7-0

The price for this work is : **$69.95.00 USD**

10 9 8 7 6 5 4 3 2 1

Release Date: **October, 2006**

Dedication

To my wonderful wife Patricia, and our loving children, Brian, Michael, and Katie as well as the greatest pack of brothers and sisters, aunts and uncles, cousins, nieces and nephews that any person could ever hope to have in just one life.

Acknowledgments

I would like to thank many people for helping me in this effort.

I would first like to thank my immediate family, starting with my lovely and dear wife, Patricia. Again, as I offer in all my books, my wife Patricia is my source. She is the person who keeps me alive and sane and well in more ways than can be mentioned. She is the glue that holds our whole family together. Besides that, she keeps getting better looking as the years go by, and I love to see her wonderful face every day in my life. No matter where I look when she is in my view, the view is very good. Her daddy, a wonderful man in his own right, Smokey Piotroski, called his little girl *Packy* as a nickname. Though Stash is now with the Angels, I love that name and the person who wears it and I still use it to address my little Packy. God gave me a gift that keeps me going. Thank you Packy for all you do to keep me and our whole family well and mostly, thank you for the smile that you always put on my face.

I would also like to thank my twenty-one year-old daughter, Katie, who is still my little baby doll. Kate helps me in any way she can. Even more than that, her sweet voice and her accomplished guitar playing gets the muse racing as my fingers pound the keyboard. Katie is starting to feel better now and we thank God for that but it still is not easy for her. He is now working, helping other children cope with their difficulties. She is well on her way to being OK. I thank my Katie for she will always be Daddy's Little Girl. I love you very much.

A special thank you also goes to Dr. Patrick Kerrigan, who takes credit for little but helps all who come his way in all ways. Dr. Patrick comes to the job with the abilities of Hippocrates and the patience of Job. He left no stone unturned in helping Katie through her need and as I have witnessed, he does it for all and with humility.

Thanks also go out to my twenty-four -year-old son, Michael is getting ready for his second year of Law School after receiving his B.S. Degree in Accounting. After a trek with a construction company this summer, Michael is more than anxious to be a lawyer. Of course, when he goes away – even for a week or two I miss him very much.

I also thank my twenty-six-year-old son, Brian, who just knocked 'em dead in Law School and graduated Magna Cum Laude. Brian spent most of this year taking courses in preparation for the Bar Examination, which he passed with flying colors. He's got all kinds of interesting job applications out there so soon he'll be saving the world.

I thank Brian for the editing of this book – at least most of the chapters. He is phenomenal wit the written word. If you find a mistake anywhere in this book, it had to be in one of the chapters I wouldn't let him touch. Brian, you make us very proud. Mom and I are very proud of all of our children and we thank each of them for their work in academia and their efforts on our behalf.

Thanks also to the extended family who are always there to lend a helping hand. Barb, Kim & Dave, Dawn, Cindy & Dave Boyle, Megan and Sean are some of the most wonderful family in my life. And dad Joe, with the angels, always gets his plugs in. Thanks also to Melissa and Paul Sabol and their new baby boy Paul IV.

Accomplishments often materialize because of a strong friendship infrastructure. I am pleased to have a number of great friends. Among them is my longtime best friend Dennis Grimes, who is always there to help, though he may think everything I write sounds the same. Professor Grimes is on the faculty with me at Marywood University and he is a CIO for Klein Wholesale. He is very talented and very helpful. I selected his comments about RPG as a cover quote. Barbara Grimes, Patricia Grimes Yencha, Elizabeth (Wizzler), Mary the PhD., Denyse back from the U.K., Grandma Viola, and Grandma Gert also pitch in whenever the opportunity arises. Dennis helps me in whatever way I ask, especially when I am stuck. I really appreciate all you do for me "D." Thank you

The busiest guy on all of my book projects, besides myself, is always Joe McDonald. Joe is the businessman in our publishing venture, and in that, he's all business. Joe is the former Publisher of the Scranton Tribune/Scrantonian Newspaper. So he's got the right background to make sure everything is A-OK! I promised Joe that my next book was going to be non-technical as we moved the publishing business to Children's books and third party authors. Joe assures me that after this book, he will have the courage to lead me to the children's side of the business where our next book is scheduled to be The Adventures of Eddy (The Dog) written by Joe's Grandson. Soon, it will be on the bookshelves of America. My thanks also go to Peg McDonald for making sure that Joe is always ready for action, especially now that he is recovering from a tooth extraction induced broken jaw.

Of course, the long list of helping hands contains lots of names: Gerry Rodski and Joyce, Jeanne and Farmer Joe Elinsky, John and Carol Anstett, Grandma Leona and Grandma Annie (from the "Mayflower"), Joe and Betta Demmick, Christopher & Emily, Carolyn and Joe Langan, Bob & Cathy Wood, Cousin Eddy & Rose, Karen and Al Komorek, Bonnie and George Mohanco, Becker and Robin

Mohanco, Lilya, Josh, and Alaina Like Mohanco, Bob and Nettie Lussi, Kim and Ruth Borland--- they are all there when needed.. Other helping hands include Dr. Lou and Marie Guarnieri as well as Mary and Cindy Guarnieri, whose hands have been indispensable. I can't forget Mike and Frannie Kurilla & Frankie and Tony, Jerry and Hedy Cybulski, Linda DeBoo and Bob Buynak, Joe, the Chief, LaSarge, John and Susan Rose, and Dave and Nancy Books. Thanks also to Dr. Rex Dumdum from Marywood-- my academic mentor. Special thanks also to the E.L. Meyers Class of 1965 (41st reunion this year) for some early training in the art of writing.

And don't let me forget Patricia's parents, Arline and Stanley Piotroski, who continue to guide us in our lives. Cathy and Marty Piotroski, Dr. Susan Piotroski and Dr. Mitch Bornstein, Matt and Allie, Dr. Stan Piotroski, Carol Piotroski, Sister Marlene, Justin and Katie, Merek, MacKenzie, Myranda, McKylee Mae, Lynn, and Scott Piotroski, Erin & Marty Piotroski Jr, Ralph Harvey, Brian, Margaret Brian Jr., Hannah Frostbutter, Beth & Dante Frostbutter,. Trooper Danny Oleniak and Maryann, Barb & Ed & Barb Hahn, Russ & Bernadette Banta, Louise Usloski, Theresa Vital & Roseann Dunay

The list includes the Kelly parents -- Ed and Irene also provide guidance from upstairs as well as direct intervention as needed; Anna Maye, Nancy and Angel Jim Flannery (Leland [No K] Zard), Renee (Bean), Jimmy (Jim Bob), Bridget, Mary (MeeWee), Danny, Michael (McPike) , Ken (La Rue), Jen, Angel David Davidow (a.k.a. Brunoch Zard), Stephen (P.Q. Whoozer), Matthew(M.Q. Peph), Bailee Roo, Viva La Vieve, and Billiard Peph, Joe and Rosalee, Raymond and the real Sparkey. Mary and Bill Daniels, Liz (Weezler), Brian, Sophia (Chubby Cheeks DiNardo,) Elise (Fleure d'Lise), Ginochetti. Megan (Meggledeebaigledee), Bill Jr (Billdog), Vicky. Diane and Joe , Tara and Colleen Kelly. Ed and Eudart, Eddy, John, and Robert Kelly. Bill Rolland- Notre Dame's # 1 Fan and master of accommodations, Bill Kustas, Bill & Helen Kush, Steve and Shelly Bartolomei, Keith and Dorie Zinn, Cheryl Danowski, Ricky, Joanne, Briana and Eric Bayer, and of course the great musical cutter Harrison Arthur and his friend Harry Heck Jr. The infamous Pierre Le Kep. More thanks to Judy Jones and Jerry Reisch and Judy Judy Judy Seroska

Going back to the top of the list of helpers is my wonderful and huge pack of cousins. The list begins with the Uncles and Aunts, many of whom are now Angels. Uncle Nick and Aunt Emma McKeown, Dave and Kathleen Conklin, Rita and Frank DeRiancho, Joan and Tom Nelson, Aunt Ruth and Uncle Joe McKeown, Kathy and Joe McKeown Jr., Aunt Louise and Uncle Jimmy McKeown, Patsy, Danny and Jerry McKeown, Nina and Jim Brady, Jimmy Brady, Tommy and Mary Rowan, Arlene and Richard May,

Little Tommy Rowan, Helen and Joe Drexinger, and all the other cousins, uncles & aunts who can't make it to the special muse event every summer in Montrose.

Of course, there's Uncle Martin Kelly, Aunt Marie and Uncle Bud Hopko, Aunt Helen Berger Aunt Catherine and Uncle Leonard Lamascola, Uncle Pat & Aunt Mary Kelly, Sharon, Maureen, Jud, Pat Jr., and Tommy Kelly. Uncle Mike & Aunt Fran, Uncle Joe Kelly, Marguerite and Helen. Red Cloud (Uncle Phil Kelly) is also on the list for his due diligence in writing postcards as is Uncle Johnny Kelly for being the youngest.

In the special care category, Dr. Lou Guarnieri has been making sure that my bones are aligned properly for years. So that I can give those speeches with a bright smile, I got some big help from Dr. Lou Kicha the Great and his highly competent team of professionals at Aspen Dental-- John Cicon, Carol Kephart, Nicole Arnone, Anita Florek, and the tooth architect, Mary Lou Lennox. Thank you all very much.

A big thank you to Ray Killian from Penn Millers Insurance in Wilkes-Barre PA for reporting some typos from the first printing that are corrected in this version. Other helpers in many ways include the former Klein development staff. In alphabetical order, by first name, the Klein team includes: Barb Chaderton, Bill 'Curly' Kepics, Cindy Dorzinsky, Cindy Goodwin, Dennis Grimes, Eric Priest, Jeff Massaker, Jerry Reisch, Joe Byorick, Joe Rydzewski, John Robbins, Paula Terpak, Rob Gilboy, Rod Smith, and Rosalind Robertson and.

I would also like to thank Nancy Lavan, our sponsor at Offset Paperback, our printer. She continually encourages us in our writing and publishing efforts. Chris Grieves, our new customer service person has made working with the printing process an easy task. Special thanks go to Michele Thomas, who takes ideas and makes wonderful images from them, such as this wonderful cover.

To sum up my acknowledgments, as I do in every book that I have written, I am compelled to offer that I am truly convinced that "the only thing you can do alone in life is fail." Thanks to my family, good friends, and a helping team, I was not alone.

Table of Contents at a Glance

Chapter 1 Introduction to the RPG Language 1
Chapter 2 The History of the RPG Language 9
Chapter 3 Understanding the RPG Fixed Logic Cycle...................................25
Chapter 4 Developing RPG Applications...47
Chapter 5 Your First RPG Program 97
Chapter 6 The Specifics of RPG Coding –H Spec – by Example 107
Chapter 7 The Specifics of RPG Coding –F & L spec by Example................. 127
Chapter 8 The Specifics of RPG Coding – I spec – by Example 161
Chapter 9 Specifics of RPG– Structures/Constants by Example 203
Chapter 10 The Specifics of RPG Coding– C spec – by Example.................. 219
Chapter 11 The Specifics of RPG Coding– O spec – by Example 247
Chapter 12 Decoding and Debugging RPG Programs 277
Chapter 13 Introduction to RPGIV.. 295
Chapter 14 RPG (/400) Operations 369
Chapter 15 RPGIV Operations and Built-In Functions 419
Chapter 16 RPG Arrays and Programming Structures....................................... 467
Chapter 17 RPG Data Structures 515
Chapter 18 String Coding In RPG 549
Chapter 19 RPG/400 & RPGIV Structured Programming 619
Chapter 20 Interactive RPG Programming 645
Chapter 21 RPG Subfile Programming.. 693
Chapter 22 RPG Database & Inter-Program Ops & Examples........................ 743
Chapter 23 Case Study Part I RPG Operations in Action................................. 775
Chapter 24 Case Study Part II RPG Operations in Action 823
Chapter 25 ILE & Static Binding. 855
Chapter 26 RPGIV Procedures and Functions 875
Chapter 27 Free Format RPG /FREE... 907
Chapter 28 Using Embedded SQL in RPG Programs 919
Index …. 933

Table of Contents

Chapter 1 Introduction to the RPG Language 1
What is RPG? 1
Creating New RPG Programmers 2
Non-RPG Programmers 3
IBM Leads the RPG Way 4
The Minicomputer Revolution 4
IBM Not Popular in Higher Education 5
RPG Kills the Minicomputer Revolution 6
Chapter Summary 7
Key Terms 7
Review Questions: 8

Chapter 2 The History of the RPG Language 9
Filling in the Blanks 9
In the Beginning 9
Fixed Cycle Accounting Machines 10
RPG on the IBM 1401 11
Here Comes the IBM System/3 11
System/3 as a 407 12
System/3 RPG Was RPG II 12
The Four RPGs 13
RPG I 13
RPG II Adds More Capabilities 14
RPG III 16
RPG IV – Best Language on any Platform 18
The Once and Future RPG 20
Chapter Summary 20
Key Terms 21
Review Questions 22

Chapter 3 Understanding the RPG Fixed Logic Cycle 25
Cycle Operations Are OK! 25
A Quick Look at the RPG Cycle 26
Report Headings 28
Step 1 – Write Heading and Detail Lines 30
Step 2-- Get Input Data 30
Step 3 – Perform Total Calculations 31
Step 4 -- Write Total Output 32
Step 5 – LR On? 32
Step 6 Move Data from Input Area to Fields 33
Step 7 Perform Detail Calculations 33
Additional INPUT Processing Information 34
RPG Matching Records Processing 34
Designating Record Types 35
Record Identifying Indicator Processing 36
Primary and Secondary Files 37
The MR Indicator 38
Multiple Output Records per Cycle. 40
RPG Fixed Logic Cycle Summary 40

Chapter Summary 42
Key Terms 43
Review Questions 43

Chapter 4 Developing RPG Applications 47

Writing Your Programs 47
Programming Development Manager (PDM) 48
PDM Features 48
PDM: the List Manager 50
What Does PDM Do? 50
Starting PDM 51
Work with Libraries Using PDM 52
Work With Objects 53
Work with & Other Options 54
Work With Members 55
Editing Source Members 57
Compiling (Creating Objects from Members) 57
Member Source Types 58
COPY Members with PDM 58
Source Entry Utility (SEU) 59
SEU Features Overview 60
SEU Features: 60
Getting SEU Started 62
SEU the Editor 66
Line Editing Commands 68
Why Line Commands? 69
List of Line Commands 69
Types of Line Commands 70
Window Line Command 70
SEU Top-Line Commands 70
The SEU Main Edit Panel 71
SEU Command Keys 72
Defaults, Find/Change, Browse/Copy 73
Line Command Exercises 73
Copy One Line 74
Delete Operations 76
Copy Blocks 77
More Line Tricks -Move, Insert, and Repeat 79
Compiling and Executing your Program 85
Running the NEW Program 91
Summary and Conclusions: 91
Key Terms 92
Review Questions 93

Chapter 5 Your First RPG Program 97

The Specs for Your First Program 97
A Description of the Data 100
The Data Itself 102
Chapter Summary 104
Key Terms 105
Review Questions 105

Chapter 6 The Specifics of RPG Coding –

Control Specification – by Example 107
From Coding to Decoding..107
Well Not Exactly!..109
Decoding the PAYREG RPG Program..112
Internally & Externally Described Data..112
H-- Header (Control) Specification Form..114
H Columns 7-14 (Reserved)..115
H Column 15 (Debug)..115
H Columns 16-17 (Reserved)..115
H Column 18 (Currency Symbol)..115
H Column 19 (Date Format)..116
H Column 20 (Date Edit)..116
H Column 21 (Decimal Notation)..116
H Columns 22-25 (Reserved)..116
H Column 26 (Alternate Collating Sequence)..116
H Columns 27-39 (Reserved)..116
H Column 40 (Sign Handling)..117
H Column 41 (Forms Alignment)..117
H Column 42 (Reserved)..117
H Column 43 (File Translation)..117
H Columns 44-56 (Reserved)..117
H Column 57 (Transparency Check)..117
H Columns 58-74 (Reserved)..118
H Columns 76-80 (Program Identification)..118
RPGIV Header (H) Specification..119
Chapter Summary..122
Key Terms..123
Review Questions..123

Chapter 7 The Specifics of RPG Coding – File Description & Line Counter Specifications – by Example 127
Talking to the Outside World..127
F-- File Description Specification Form..127
Why File Descriptions?..129
F Column 7-14 File Name..131
F Column 15 File Type – I, O, Etc...131
F Column 16 File Designation – Primary, Secondary, etc...132
Blank, P, S-- Output, Primary, and Secondary..132
R-- Record Address File..133
T-- Array or Table File..133
F-- Full Procedural File..134
F Column 17 (End of File)..134
F Column 18 (Sequence)..135
F Column 19 (File Format)..136
F Columns 20-23 (Reserved)..138
F Columns 24-27 (Record Length)..138
F Columns 28-39 (Other Entries)..139
F Column 28 Limits Processing..139
F Column 29-30 Length of Key or Record Address..139
F Column 31 Record Address File Type..140
F Column 32 File Organization..141
F Columns 33-34 Overflow Indicator..141
F Columns 35-38 (Key Field Starting Location)..142
F Column 39 (Extension Code)..143

F Columns 40-46 (Device) ... 144
F Column(s) 47-52 (Reserved) ... 145
F Column 53 (Continuation Lines) ... 145
F Column(s) 54-59 (Routine) ... 145
F Column(s) 60-65 (Reserved) ... 146
F Column 66 (File Addition) ... 146
F Column(s) 67-70 (Reserved) ... 146
F Column(s) 71-72 (File Condition) ... 147
F Column(s) 73-74 (Reserved) ... 147
F Column(s) 75-80 (Comments) ... 147
FC File Description Continuation Lines ... 148
FC Columns 7 – 18 Unused ... 148
FC Columns 19 – 28 External Name of Record Format ... 148
FC Columns 29 – 46 Unused ... 148
FC Columns 47 – 52 Record # Field for Subfile ... 149
FC Column 53 Continuation Character (K) ... 149
FC Columns 44 – 59 & 60 – 67; 'K' Options ... 149
FC Columns 68 – 74 Unused ... 149
FC Columns 75 – 80 Optional Comments ... 149
FC Options and Entries for Continuation ... 150
RPGIV File Description Keywords ... 153
L -- Line Counter Specification Form ... 155
L Column 7-14 File Name ... 156
L Column 15-17 Lines per Page ... 156
L Column 18-19 Form Length ... 156
L Column 20-22 Overflow Line Number ... 156
L Column 23-24 Overflow Line Indicator ... 157
L Column 25 - 74 Blank ... 157
L Column 75 - 80 Optional Comments ... 157
RPG IV Line Counter Information ... 157
Chapter Summary ... 157
Key Terms ... 159
Review Questions ... 159

Chapter 8 The Specifics of RPG Coding – Input – by Example ... 161

The Many Faces of RPG Input ... 161
Internally & Externally Described Input ... 163
I – Input Specification Form ... 163
Input Specification Quick Summary ... 165
RPG Input Form Types ... 165
PAREG Record and Field Statements ... 166
I – Externally Described Record ID Entries ... 167
IEDRI Columns 8-14 Record Name ... 167
IEDRI Columns 16 – 18 Reserved ... 168
IEDRI Columns 19 – 20 Record Identifying Indicator ... 168
IEDRI Columns 21 – 41 Unused ... 168
IEDRI Columns 42 – 74 Reserved ... 168
IEDRI Columns 75 – 80 Reserved ... 168
Applying Input Record IDs to PAREG ... 168
I – Externally Described Field Description Entries ... 172
Field Specifications in RPG/400 ... 172
IEDFD Columns 7 – 20 Reserved ... 173
IEDFD Columns 21 – 30 External Field Name ... 173

IEDFD Columns 31 – 52 Reserved ... 174
IEDFD Columns 53 – 58 Field Name ... 174
IEDFD Columns 59 – 60 Control Level ... 174
IEDFD Columns 60 – 61 Match Fields ... 175
What about M2 through M9? ... 176
IEDFD Columns 63 – 64 Reserved ... 179
IEDFD Columns 65 – 70 Field Indicators ... 179
IEDFD Columns 71- 74 Reserved ... 179
IEDFD Columns 75 – 80 Comments ... 180
I Program Described Record Identification Entries ... 180
IPDRI Columns 8-14 File Name ... 181
IPDRI Columns 14-16 (Logical Relationship) ... 182
IPDRI Columns 15-16 (Sequence) ... 183
IPDRI Column 17 (Number) ... 185
IPDRI Column(s) 18 (Option) ... 186
IPDRI Column(s) 19-20 (Record Identifying Indicators) ... 187
IPDRI Column(s) 21-41 (Record Identification Codes) ... 188
I Program Described Field Description Entries ... 190
IPDFD Position 43 (Data Format) ... 191
IPDFD Position 44-51 From / To Record Positons ... 192
Additional Information re: from / to length ... 193
IPDFD Column(s) 52 Decimal Positions ... 193
IPDFD Column 53-58 Field Name ... 194
IPDFD Column(s) 59 – 60 Control Level ... 194
IPDFD Column(s) 61 - 62 Matching Fields ... 195
IPDFD Column(s) 63 - 64 Field Record Relation ... 197
IPDFD Column(s) 65-70 Field Indicators ... 197
IPDFD Column(s) 71 – 74 Unused ... 198
IPDFD Column(s) 76 – 80 Comments ... 198
RPGIV Program Described Files ... 198
IPIV Columns 31-34 Data Attributes ... 198
IPIV Column 35 Date/Time Separator ... 199
Chapter Summary ... 199
Key Terms ... 200
Review Questions ... 200

Chapter 9 The Specifics of RPG Coding – Input Structures & Constants – by Example ... 203

What is a Data Structure? ... 203
Data Structure Record ID Entries ... 204
IDSRI Column(s) 7-12 Data Structure Name ... 205
IDSRI Column(s) 13-16 Reserved ... 206
IDSRI Column(s) 17 External Description ... 206
IDSRI Column(s) 18 Option ... 206
IDSRI Columns 19–20 Record ID Indicator ... 207
IDSRI Columns 21 – 30 External File Name ... 207
IDSRI Column(s) 31 – 43 Reserved ... 208
IDSRI Column(s) 44-47 Data Structure Occurrences ... 208
IDSRI Column(s) 48 – 51 DS Length ... 208
IDSRI Column(s) 52 – 74 Reserved ... 209
IDSRI Columns 75 – 80 Comments ... 209
I Data Structure Subfield Entries ... 209
IDSSF Column 7 Reserved ... 210
IDSSF Column 8 Initialization Option ... 210

IDSSF Columns 9 – 20 Reserved 210
IDSSF Columns 21 – 30 External Field Name 210
IDSSF Columns 21 – 42 (Initialize Value) 211
IDSSF Columns 31 – 42 Reserved 211
IDSSF Column 43 Internal Data Format 211
IDSSF Column 44 – 51 Field Location 212
IDSSF Column 52 Decimal Positions 212
IDSSF Column 53-58 Field Name 213
IDSSF Columns 59 – 74 Reserved 213
IDSSF Columns 75 – 80 Comments 213
I Named Constant Entries 213
INC Columns 7 – 20 Reserved 214
INC Columns 21 – 42 Constant 214
INC Columns 43 Type / Continuation 215
INC Columns 44 – 52 Reserved 215
INC Columns 53 – 58 Constant Name 215
INC Columns 59 – 74 Reserved 215
INC Columns 75 – 80 Comments 215
Chapter Summary 216
Key Terms 216
Review Questions 217

Chapter 10 The Specifics of RPG Coding – Calculations – by Example 219

Mathematics and Logic 219
The RPG Calculation Specification 221
Calculation Specification Statement Format 227
C Columns 7-8 (Control Level) 227
PAREG and Control Levels 228
C -- Columns 10-17 (Indicators) 230
Pseudo Code 231
Taking Totals 234
C – Columns 18-52 Factors and Operators 236
C -- Columns 18 – 27 Factor 1 237
C -- Columns 28 – 32 Operation 237
C -- Columns 33 – 42 Factor 2 237
C -- Columns 43 – 48 Result Field 238
C -- Columns 49 – 51 Length 238
C -- Columns 49 – 51 Decimal Positions 239
C- Column 53 (Operation Extender) 239
C – Columns 54-59 (Resulting Indicators) 240
C- Columns 60-80 (Comments) 241
Another Look at PAREG Example CALCS 241
Chapter Summary 243
Key Terms 244
Review Questions 244

Chapter 11 The Specifics of RPG Coding– Output – by Example 247

Showing the Results 247
O-- Output Specification Form 247
Output Record ID and Control Entries 248
OPDRI -- Specification Columns 7-14 250

Printing to Multiple Printers 250
OPDRI Columns 14-16 (Logical Relationship) 251
OPDRI — Column 15 (Type) 252
OPDRI – Column 16 (Fetch Overflow / Release) 254
OPDRI – Columns 17-22 (Space and Skip) 254
Powerful Report Writing 255
Spacing & Skipping 256
OPDRI – Columns 23-31 (Output Indicators) 257
Field Description and Control Entries 259
OPDFD- Columns 7 through 22 Reserved 260
OPDFD- Columns 23–31 Output Indicators 260
OPDFD-- Columns 32-37 Field Name 261
Field Names, Blanks, Tables and Arrays 262
OPDFD Column 38 Edit Codes 262
OPDFD —Column 39 Blank After 264
OPDFD - Columns 40-43 End Position 266
OPDFD - Column 44 (Data Format) 268
OPDFD - Columns 45-70 Constant or Edit Word 268
RPG/400 Edit Words 270
Body - Status - Expansion 271
OPDFD -- Columns 71-74 Reserved 274
OPDFD -- Columns 75-80 Comments 274
Chapter Summary 274
Key Terms 275
Review Questions 275

Chapter 12 Decoding and Debugging RPG Programs 277
The PAREG Program Decoded 277
The Decoding Process 281
RPG Cycle and PAREG Decoding 283
Debugging for Learning and Decoding 286
Chapter Summary 293
Key Terms 294
Review Questions 294

Chapter 13 Introduction to RPGIV 295
You've Already Seen RPGIV? 295
A Better RPG 295
What is RPGIV? 298
Who Needs It? 298
Many Beneficial Changes to RPGIV 300
7 Specification Sheets vs. 8 300
Functions Moved to More Logical Specification Sheet 300
Specifications Changed to Accommodate Extended Functions 301
Additional Functions Added To Language 301
Many RPG Limits Removed / Increased 301
RPGIV ILE Environment 302
Pointer Example 303
Static Calls 303
Dynamic Calls 303
Keyword Orientation 304
Decoding the PAREG RPGIV Program 310
The RPGIV Header Specification 310
RPGIV File Description Specification 311

RPGIV Input Spec Changes 312
Record ID in RPGIV 313
Input Field Spec in RPGIV 313
RPGIV Calculation Spec Changes 314
CALC Example 1 315
CALC Example 2 315
CALC Example 3 316
RPG IV Free-form Op-codes 317
CALC Example 4 - Free Form in Action 319
RPGIV Field Length and Decimals 321
Built-In-Functions (BIFs) 321
Other BIFs 323
Pointer Example 1 -- Pointer Variables 324
RPGIV Output Spec Changes 325
Output Example 1 -- RPGIV 325
Output Example 2 -- RPGIV Printer Output 326
Output Example 3 Literal End Positions 326
RPGIV Output Record Format and Control 327
RPG IV Field and Control Specification 331
Wrap-Up RPGIV with Similar Forms 332
The RPGIV Definition 'D' Specification 332
D Spec Example 1 Indentation 333
D Spec Example 2 – DS From / To 333
D Spec Example 3 DS Using Length 334
D Spec Example 4 Multiple Occurrence DS 335
D Spec Example 5 Named Constant 336
D Spec Example 6 Stand-Alone Fields 336
D Spec Example 7 Tables & Arrays 337
D Spec Example 8 Compile Time Tables 338
D Spec Example 9 Ext. Described DS 338
D Spec Example 9 Data Area DS 339
D Spec Keyword Continuation Line 339
'D' Spec Continued Name Line 340
D – Columns 7-21 Name 341
D – Column 22 External Definition 341
D – Column 23 Type of Data Structure 342
D - Columns 24-25 Definition Type 343
D Columns 26-32 (From Position) 343
D - Column 40 (Internal Data Type) 345
D -- Columns 41-42 (Decimal Positions) 346
D -- Column 43 Reserved 347
D -- Columns 44-80 (Keywords) 347
D -- Columns 81-100 Comments 347
D Specification Keywords. 348
Built-In Functions / Pointers 353
Date & Time Operations 354
Date and Time Op-Codes 358
Date Example 1 – Four Operations 359
Date/Time Duration Codes 359
Date Example 2 OP-CODES In Use 360
Where to specify D, T, Z Formats? 360
Date 362
Chapter Summary 363
Key Terms 364
Review Questions 365

Chapter 14 RPG (/400) Operations 369

How RPG Gets Things Done! 369
RPG Operations 371
Basic Operations - Including Arithmetic 371
Compare & Branch / Subroutine Operations 379
Call Operations (Inter-program) 386
Data Manipulation Operations 389
Database & Device File Operations 391
Data Structure, Data Area, Table, Array, String Operations 397
Data Structure (as a data structure) 398
Table 399
Strings 399
Program Control, Declarative, Informational & Other Operations 404
Structured Operations 406
All RPG Operations / Parameters 408
Chapter Summary 412
Key Terms 413
Review Questions 414

Chapter 15 RPGIV Operations and Built-In Functions 419

What Is the Same in RPGIV? 419
RPGIV-Only Operations 421
RPGIV Built-In Functions (BIFs) 431
Chapter Summary 463
Key Terms 463
Review Questions 464

Chapter 16 RPG Arrays and Programming Structures 467

Advanced RPG/400 Elements 467
IBM Definitions 468
Three Types of Arrays 469
Array Syntax Rules 471
Compile Time Arrays 475
Compile Time Array Record Rules: 476
Pre-run Time Array 477
Alternating Arrays or Tables 479
Related Arrays w/o Alternating Format 480
Tables 481
Related Tables w/o Alternating Format 483
Searching Arrays & Tables 484
LOKUP Example 486
The Meaning of the LOKUP Resulting Indicators 487
LOKUP (LOOKUP in RPGIV) Review 489
Changing Entries in Arrays 491
Adding Entries to Arrays 492
Another LOKUP Example – 494
Table LOKUP with one Table 494
Speeding up Array Searches 496
Faster Processing for Unordered Arrays 497
Consider Keyed Files Vs. Arrays 499
Named Constants 500

Named Constant Examples ... 500
Figurative Constants ... 502
Rules for Figurative Constants ... 504
Figurative Constant Example 1 ... 505
Figurative Constant Example 2 ... 506
Other RPG/400 Reserved Words ... 509
Chapter Summary ... 511
Key Terms ... 513
Review Questions ... 513

Chapter 17 RPG Data Structures ... 515
What is a Data Structure? ... 515
Special Types of Data Structures ... 516
How Are Data Structures (DS) Defined? ... 518
Moving Data into Data Structures ... 518
Data Structure Example 1 ... 519
Data Structure Example 2 ... 520
Accessing Multiple Occurrence Data Structure ... 521
Data Structure Example 3 ... 521
Data Structure Example 4 ... 522
Working with System i Data Areas ... 523
*LDA Local Data Area ... 524
Additional Data Area Data Structure Info ... 527
More on *NAMVAR DEFN ... 528
Summary of Data Area Operations ... 530
More Data Area Data Structure Examples: ... 531
Data Structure Initialization ... 535
Method 1 -- Global Initialization ... 535
Method 2 - Subfield Initialization ... 536
Initialization Subroutine ... 537
The RESET and CLEAR Operations ... 539
The RESET Op-Code ... 539
The CLEAR Op-Code ... 540
Data Structure Performance Trick ... 544
Speeding up Date Reformatting ... 544
Chapter Summary ... 545
Key Terms ... 546
Review Questions ... 546

Chapter 18 String Coding In RPG ... 549
Strings Are an RPG Later Arrival ... 549
String Characteristics: ... 549
Basic String Handling Functions ... 550
Complex String Handling Functions ... 551
RPG String Implementation ... 551
Variable Subscripting for String Manipulation: ... 554
Convert Upper/Lower (U/L) Case of State Field ... 561
RPG Fields ... 564
String Coding In RPG/400 ... 565
The CAT Operation (CAT RPGIV) ... 565
The CHECK Operation (CHECK RPGIV) ... 569
CHEKR CHECKR(RPGIV) ... 572
SCAN Operation -- RPG/400 & RPGI) ... 573
SUBST Substring Operation --- RPG/400 & RPGIV ... 575

XLATE Translate Operation --- RPG/400 & RPGIV 576
Complex String Examples 578
Complex String Example 1: 579
Hexadecimal Literals & Named Constants 579
Complex String Example 2: 580
Extract & Identify a String 580
Complex String Example 3: 581
Find the Length of a String 581
Complex String Example 4: 581
Convert Last, First to First Last 581
Complex String Example 5: 583
Convert Last First to First Last, Mixed Case 583
Complex String Example 6: 585
Center a Line of Text 585
Complex String Example 7: 587
Justify a Line of Text 587
Complex String Example 8: 588
Substring Insertion 588
Complex String Example 9: 589
Substring Text Replace 589
Complex String Example 10: 590
Substring Text Deletion 590
Complex String Example 11: 592
Scan and Replace Text 592
Complex String Example 12: 597
Translate Numbers 597
Complex String Example 13: 598
Remove Leading Zeros 598
Complex String Example 14: 599
Find a # in an Address 599
Complex String Example 15: 601
Replace All Blanks in a Text Line 601
Complex String Example 16: 604
Check a Name for Valid Characters 604
Arrays vs. String Operations Example 17: 608
Combine First & Last Name 608
Arrays vs. String Operations Example 18: 610
Separate / Reverse First & Last Name 610
RPGIV for String Manipulation 614
Chapter Summary 614
Key Terms 616
Review Questions 616

Chapter 19 RPG & RPGIV Structured Programming 619
What is Structured Programming? 619
Why Structured Programming 621
Effective Program Development 621
Easier Program Maintenance 621
Readability 621
Improved Program Efficiency 622
Control Logic Structures 623
Implementing Structured Programming 623
Structured Programming 624
RPG Structured Operations 624

Select Groups - SELEC - WHXX- OTHER ... 628
Conditionally Invoke ... 630
Do While DOWxx ... 634
Do Until DOUxx ... 635
Do Groups DO ... 636
Do Loop Leave Option (LEAVE) ... 639
Do Loop Iterate Option (ITER) ... 640
"Structured" Misc. Operations – CABXX ... 640
Chapter Summary ... 641
Key Terms ... 642
Review Questions ... 643

Chapter 20 Interactive RPG Programming ... 645

The WORKSTN Device ... 645
Web Programming ... 647
Data Description Language ... 648
WebSphere Development Studio Client (WDSC) ... 649
WDSC ... 650
The Program Development Manager ... 653
The WDSC Designer ... 653
Menus ... 658
Display Panels ... 659
SDA Features ... 659
Getting Started ... 660
Screen Design Aid Menus ... 661
Creating a Display File ... 661
Screen Panel Exercise Objectives ... 662
Working with an Existing Source Member ... 664
Building SDA Image from Scratch ... 665
Specify Record Format Type ... 666
The No-Nonsense Design Image Panel ... 667
Typing Your Screen Constants ... 667
Instantaneous Feedback upon ENTER ... 668
Intermediate Exit and Creation ... 669
Checking Intermediate DDS with SEU ... 670
Adding Variable Fields from the Database ... 671
Selecting Database Fields for Use ... 672
Exiting the Data Base Option ... 673
SDA Image Commands ... 674
Column Headings from the Database ... 676
Adding Fields & Changing Field Attributes ... 677
Field Manipulation Commands ... 678
Adding a New Field to Your Display ... 679
Changing Display Attributes ... 679
Making the Attribute Change ... 681
Adding Editing Keywords ... 682
Assigning End-of-Job Indicator ... 684
Indicator at File Level ... 685
Display File in an RPG Program ... 685
Other Display Operations ... 687
Chapter Summary ... 688
Key Terms ... 690
Review Questions ... 690

Chapter 21 RPG Subfile Programming................................ 693

Subfile Lists ... 693
Old –Time Display List Management ... 693
The Subfile Method ... 694
Learn Subfiles By Example ... 695
VENDSRCH RPG Subfile Program ... 699
SDA Design Panel ... 702
Subfile Records ... 705
Three Major Purposes for Subfiles ... 707
Inquiry for Multiple Records ... 708
Inquiry w/ Update for Multiple Records ... 714
Command Key Specification ... 715
Data Entry Subfiles ... 720
SFLNXTCHG Keyword ... 722
DSPATR Keyword ... 722
SFLMSG / SFLMSGID Keyword ... 723
Process of Activation ... 723
Performing Data Entry with Subfiles ... 725
How to Correct Errors in Subfile ... 731
Two Subfiles for Error Correction ... 731
Error Correction Using One Subfile ... 732
Data Entry Approach Correct Errors: ... 735
Heads Down w/ Batch Edit ... 735
Another Subfile technique for Data Entry ... 736
Poor Person's Subfile ... 737
Variable Line Numbering ... 737
Chapter Summary ... 739
Key Terms ... 740
Review Questions ... 741

Chapter 22 RPG DB & Inter-Program Operations & Examples ... 743

Input-Output Operations ... 743
General Operations ... 744
Workstation I/O Operations ... 745
Disk I/O Operations ... 745
Data Base File Processing ... 746
CHAIN Operation ... 746
Chain by file: ... 746
CHAIN by Key ... 747
CHAIN by Relative Record Number ... 747
READ (consecutive) ... 749
Set Lower Limit SETLL (Indexed) ... 750
READ EQUAL (READE) ... 750
Set Greater Than SETGT (Indexed) ... 751
Read Prior Record (READP) ... 752
Read Prior Equal (REDPE) ... 752
Composite Key ... 753
Update & Delete (UPDAT & DELET) ... 753
Output Delete / Add / Update ... 754
Add Record ... 755
Writing Records by RRN ... 755
Printer File ... 756
Group Name for Exception Output ... 757

EXCPT Externally Described File Output 758
File Control Options 759
FEOD vs CLOSE 760
External Subroutines 761
More -- Inter-Program Communication 763
To Subprogram or Not to Subprogram 763
Good and Bad Inter-program Techniques 765
Steps for Inter-program Communications 765
Running a CL command in RPG 767
Data Retention with Program Calls 768
RETRN Operation and RT Indicator 768
FREE Operation in RPGIV 769
To Exit From a Sub Program 770
Chapter Summary 771
Key Terms 772
Review Questions 772

Chapter 23 Case Study Part I: RPG Ops in Action 775

The Once and Future PAREG2 775
The Cycle Is Efficient 779
The RPG Cycle Does Lots of Work 780
Primary File vs Fully Procedural 781
Is the Data All Wrong Here? 782
Data Design Matters 783
Matching Records Processing 784
Using Exception Output 786
Salary Option 787
Missing Time Card Logic 789
What is a subroutine? 789
Check for Missing Master 790
Control Level Breaks – No RPG Cycle 796
The MOVMS1 Subroutine – First Record 798
The PAYCLC Subroutine - First Record 799
Control Break Level 1 Processing 801
Level 1 Subroutine 802
Control Break Level 2 Processing 803
Process the Record Just Read 804
Final Total Processing 805
Externally Described RPG/400 PAREG2E 806
What are the Differences in PAREG2 – Internal v External? 810
Externally Defined Data Structure 811
RPG IV Program Versions 812
RPGIV Program Described PAREG2P4 812
File Description 814
The “D” Spec 816
Conditioning Indicators and Operation Names 816
RPGIV Externally Described PAREG2E4 817
Chapter Summary 818
Key Terms 819
Review Questions 819

Chapter 24 Case Study Part II: RPG Ops in Action 823

The Once and Future PAREG3 823
Main Program Formats 840
Three More Versions of PAREG3 844
Externally Described PAREG3E 845
RPG IV Program Versions 847
RPGIV Program Described PAREG3P4 847
RPGIV Externally Described PAREG3E4 849
Chapter Summary 851
Key Terms 852
Review Questions 852

Chapter 25 ILE & Static Binding. 855

Integrated Language Environment 855
Migrating to RPGIV & ILE 856
What is ILE? 857
Dynamic Binding 858
Static Binding 859
Static Binding by Copy 859
Static Binding by Reference 860
Tools of the Trade 860
Program 861
Module 862
Service Program 862
Procedures 863
The New CALL Statement 865
Activation Groups 865
Default Activation Groups 867
Persistent Activation Groups 869
Chapter Summary 870
Key Terms 872
Review Questions 872

Chapter 26 RPGIV Procedures and Functions........ 875

Big Changes to RPGIV 875
What is a Procedure? 876
Subprocedures vs. Subroutines 879
Local and Global variables 880
Information Hiding 881
Prototyoing Subprocedures 881
CALLP Operation 882
Recursive Calls 883
Prototype Example – One Source Module 885
Decoding the Subprocedure 888
These Are Not Standalone Fields 889
Procedure Interface 889
Proving It Works 891
External Programs / RPG Modules 892
Making GLOBAL Work 895
Creating the Program from Modules 897
Creating RPGIV Functions 897
Return Value for Functions 897
Creating Service Programs 899
Dynamic vs. Static Calls 900
Chapter Summary 902

Key Terms 903
Review Questions 904

Chapter 27 Free Format RPG /FREE........ 907
Free At Last! 907
The Free Sandwich 910
More Rules 911
Control Level Calculations (L1 etc.) 911
Linoma Modernization Example 914
Chapter Summary 917
Key Terms 917
Review Questions 917

Chapter 28 Using Embedded SQL in RPG Programs 918
SQL Works with System i5 Languages 918
Create SQL Program Commands 919
CRTSQLRPG 919
CRTSQLRPGI 919
Writing SQL Code in RPG Programs 920
Set Processing Only 922
Select and Process Multiple Rows 923
Chapter Summary 929
Key Terms 930
Exercises 931

Index 933

Preface:

Though this book was built to be a textbook for university, college, and community college level courses on the RPG/400 and RPGIV programming languages, the finished product is much more than that. It is also a tutorial, a by-example guide, as well as a complete reference for all System i RPG based application development

Finally, there is a Pocket Developer's Guide for System i RPG & RPGIV programming. Yes, it is in big pocket guide form and it is tutorial in nature. Along with the tutorials to help you learn the language, this guide is also packed with reference material so you do not have to switch to a new book once you learn the language. For example, there is all the reference help you need to be able to use every op-code in RPG/400 and RPGIV as well as every BIF that you may ever need to use. If you are looking for how to use the new RPGIV keywords and the exclusive 'D' Spec, it's got that too! Moreover, instead of weighing you down with pounds of paper, its convenient size will encourage you to "take it along for the ride" rather than leaving it behind and having to guess.

There are lots of RPG books but there has never been an RPG book like this. Instead of arguing about the merits of RPG/400, the cycle, and the modern feel of ILE RPG, this book teaches it all. You'll be pleased with all the valuable explanations and examples. You won't want to put down this comprehensive guide to learning **System i** RPG now that you've got your hands on it. This book is almost 50 years overdue.

In today's IT landscape, most **System i** shops support both RPG and ILE RPG. Besides its down-home writing style, the major benefit of this book is that it is built as an essential text for anyone charged with the responsibility of maintaining and extending RPG code at all levels. And that means a new approach to the historical cycle, RPG/400, basic and advanced RPGIV, Eval and extended Factor 2 operations, prototypes and procedures, free form RPG and, of course embedded SQL. It's all in there – from the simple to the sublime. This Guide has an

example for nearly every type of RPG operation you can imagine from interactive workstation code to subfiles, to database and device operations.

Author Brian Kelly designed this book to show you how to use RPG by working with rich examples that you'll use over and over again. Additionally, for each example, there is the exact explanation you need to get a head start on being an RPG guru. This is the first RPG book to hand to your new developers and veterans alike. More importantly, it is the right size text for any relevant modern business programming course at your nearby university or community college.

Both entry level and existing programmers will enjoy the easy to read, down home style of this pocket guide. The book gives a general notion of how programming systems work and it shows how to begin developing and maintaining code to help get you started in learning RPG. Even if you are new to AS/400 and System i, and you want to understand how to use RPG for programs that you now code in other languages, you can learn all you need to get the job done right from this pocket book. It is written in a way that assumes very little prior RPG or even generic programming knowledge.

There is no CD with this book but you can get any of the save files by coming to the Lets Go Publish Web site at **www.letsgopublish.com** If the files are not posted yet, or if you have a problem downloading, please send me an email to my private email address at **jmac160@verizon.net**. You may also send to info@letsgopublish.com. I monitor that box regularly. I would love to hear from you.

Go ahead and leaf through this book now. You'll see it is chocked full of examples. Many screen shots are included so you can code the RPG examples in the book right along with your AS/400 or System i server.

Who Should Read this book?

New programmers, existing programmers, supervisors, operation personnel, or any other person in your organization who need to know how to program in basic or advanced RPG or RPGIV. Many IT managers today are looking for ways to educate other staff in System i RPG. Look no further. If you plan to train operations people or PC people as AS/400 developers, or you want to help your staff better understand the marvels of System i RPG business-oriented programming, this is the right book.

With all of the smart PC technicians in every business and institution today, there are many who would appreciate the opportunity to learn the major System i business programming language – RPG/400 and/or RPGIV. Many of these would do very well as programmers if redeployed. This book can be all you need to move them off the mark.

If you've always wanted to be able to tell your team what you know about RPG, ILE, RPGIV, and embedded SQL programming on the AS/400 and System i, but you did not have the time, rest assured that Brian Kelly has done it for you with this book. He's said what you would have said if you had the time to say it. Moreover, the folks at LETS GO PUBLISH think you'll like what you would have said.

Consider creating a home-made RPG programmer with a minimal start-up investment. It may be a good deal for you and for your company.

Though rich in content, none of IBM's RPG reference and user manuals are built to teach you the language. They are for reference. There is way too much in IBM's manuals to learn it all but they are great detail references for specific topics. This Pocket Guide for RPG uses a different approach. It is your teaching / learning vehicle to RPG. It is your new tool to help you solve programming problems efficiently with RPG coding. Once you have learned how to program in RPG, the completeness of this book permits it to also serve you as a handy "pocket" reference and as a guide for using new techniques.

There is no doubt that RPG and RPGIV together represent the finest business programming language ever developed. I wish you well in your RPG business programming endeavors, and I hope to see you again reading another Lets Go Publish Pocket Guide in the future.

Feel free to shop for this book and other LET'S GO PUBLISH! Books at BookHawkers (www.bookhawkers.com), IT Jungle (www.ITJungle.com), MC Press (www.mc-store.com), iSeries Network (www.iSeriesNetwork.com) and other fine booksellers.

Joseph J. McDonald, Publisher
Scranton, Pennsylvania

Chapter 1

Introduction to the RPG Language

What is RPG

RPG is one of the few languages that was originally created for punch card machines that is still in common use today. Today's most modern RPG version is known both as RPGIV and as ILE RPG. Though RPGIV can run many of the programs designed in the 1960's, this new RPG language is with little doubt the most functionally complete programming language of all time. Originally developed by IBM in 1960 to run on the very popular IBM 1401, IBM has continued to extend the RPG standard so that it stands well ahead of all other modern languages.

RPG began as an acronym for **Report Program Generator**, which was descriptive of the original purpose of the language: *generation of reports* from data files, including matching record and sub-total reports. IBM engineers took pains to make the 1950's 1401 computer system easy to program by those trained to work with the punched card equipment that was prevalent in those days. This simple language called RPG made it easy to automate routine processes, and to print results on standard tabular forms.

RPG was not created to be a general-purpose programming language. Many consider this the basis for its power. Underlying every early RPG program was the 407 Accounting Machine sequencing algorithm, hardwired into the software, relieving the programmer of the burden of controlling the input process procedurally. Early textbooks called RPG a problem oriented language because it was designed to solve simple business problems – especially reporting. These texts compared RPG to COBOL, which was characterized as a procedural language in which the

programmer used input output commands inside of the program to per form the functions that were inherent automatically in the RPG language. Thus, early COBOL programs were always substantially longer than early RPG programs.

The RPG programming language initially used fixed-format cards. When the cards had been used within the context of the 407 Accounting Machine, the control panel would pick up the data from various card columns, perhaps add the data to accumulators and then print a line on a report. By mimicking this electromechanical machine, original RPG programming was little more than a fill-in-the-blanks operation. For example, if columns 47-53 of a transaction card held a part number, which was used as a control field, a two character designation in RPG was all you had to provide to make it happen.

Early RPG purists believe that versions of RPG after the first version (RPG II to RPG IV) were corrupted by the pressure to turn RPG into a general-purpose programming language. Many who worked with the early versions were able to use and to study its sheer simplicity and elegance before this aspect was diminished with the many enhancements over the years. For the purists, the enhancements came at the expense of the simplicity that had been originally built into the language from its origins as a software replacement for IBM's big tabulating accounting machines.

In many ways, the design of the original RPG language had many similarities to what are called fourth generation languages or 4GL languages. In fact, just as most if not all 4GL's, the original RPG met with its greatest resistance from those who wanted to control all aspects of the programs they wrote. Understanding what a language was going to do for them and merely filling in the blanks was not something to which the experienced programmers of the day were ready to adapt.

Creating New RPG Programmers

In the 1960's the supply of programmers was not very deep so IBM and other companies trying to gain a computer sale would often have to sell the business prospect on the idea of creating its own programmer(s). Sometimes it was the shipping clerk; sometimes it was the head order taker; sometimes it was a bookkeeper, and sometimes it was a woman or

man picking or packing items in the warehouse. Once the "programmer" was selected, his or her life changed substantially.

Instead of their former position responsibilities, they were now on the forefront of the computer revolution. As a trusted employee of the organization, they were trained to understand the business computer of the day and the computer language that best fit the computer. This selection often overwhelmed the individual selected but soon, they knew that something good had just happened to them.

In small businesses in which IBM's "Marketing Representatives" had convinced the business executives that they too could benefit from automation, the programming language most often selected was none other than RPG. This language was better accepted by those just learning about computers as a means to create grassroots solutions for the businesses that had given them this phenomenal opportunity to learn about computers. Once introduced to RPG, the neophytes knew they could master its intricacies and they could use its simplicity to help their organization. They were not computer professionals by design but by destiny they were soon to be.

From their humble beginnings, however, they were not concerned about the latest and greatest in programming facility or function. They did, however, want to know how they could best help the organization become successful using the computer system that had been selected. Those programmers selected in this fashion, and there are many, embraced the RPG language as a means of making their respective organizations successful because they could readily understand what it would take to make it happen. RPG was the intuitive business language that helped many of these initial pioneers believe that they could become professional business programmers. And, they did!

Non-RPG Programmers

During this same time, there were also many engineers and scientists who had begun to tinker with computer technology. These professionals did not necessarily understand a debit from a credit or pick list from a payroll stub. Moreover, they were hoping that nobody would come by any time

soon with an explanation of these that made sense. They simply were not tuned into the business of business.

Being math or science oriented, they flocked to the notion of assembly language (almost as primitive as computer machine language). They also fell in love with the Fortran language. Fortran comes from the two words *Formula* and *Translation* and this language was created by engineers for engineers. With these tools, they were able to solve problems using simultaneous linear equations and other complex mathematical notations that would have taken days in the past. These soon-to-be computer science types had no inclination to program business applications, nor were they attracted to anything that the business oriented RPG language had to offer.

IBM Leads the RPG Way

IBM quickly became the leader in business application programming. Not only did IBM create the RPG language for small businesses but it also perfected COBOL (Common Business Oriented Language) for its mainframe computers. Though Big Blue did offer RPG for mainframes and it eventually offered COBOL for smaller business, RPG became the dominant business language for small to medium sized businesses and COBOL became the dominant business language for larger businesses. First RPG was built for the 1401, then the System/360 model 20 and then the System/3.

The Minicomputer Revolution

While IBM was perfecting the notion of busiess computing on its small business line headed by the System/3 and its mainframe line with the System/370, a new breed of computer systems became very popular without an help from IBM I might add. Companies such as DEC, Hewlett-Packard, Eang, and Data General sprang up and took the engineering / scientific world by storm. These machines cost 75% or more of the price of an IBM System/370 and unlike IBM's processing power-challenged System/3 line, they ran engineering and scientific programs quite well.

Moreover, because these minicomputers were in the price range of s System/3 and even much less, businesses began to use these machines to perform routine business applications. The hardware vendors helped this trend by building their own version of RPG and they added all of IBM's System 3 and later small business machine customers to their prospect list. During these days, I can remember IBM using the term, "They're eating our lunch," to describe how ell the minicomputer vendors were doing in traditional IBM business computer territory.

Though many of the machines provided RPG, it was not the favorite language of the engineers and scientists who often ran the computer departments or the departments in which these machines were installed. To be frank, it was not even on the list. Instead of RPG, they advocated such lower level computer science type languages as C, Pascal, and BASIC. These came into being and soon became pervasive on the minicomputers of the day.

IBM Not Popular in Higher Education

Nobody can say for sure what came first, the chicken or the egg. However, over the last thirty or more years -- perhaps dating back even thirty-years sooner to Harvard's snub of Thomas Watson Sr. re: the Harvard / IBM Mark I Computer in the 1940's, colleges and universities have not held IBM or its products in high regard. Do academics dislike IBM because the company has traditionally made its living using languages such as RPG or COBOL; or do the academics dislike RPG and COBOL, especially the IBM-defined RPG, because they are made by IBM. These questions are at the center of the debate.

Regardless of the reason, there has been little love between IBM and the computer science community in academia for quite some time. Overall, this has hurt IBM to a degree but it has also hurt IT students expecting to be able to be employed in a mostly IBM-oriented business world without having been given the proper business programming credentials.

It is no wonder that the minicomputer quickly became the best friend of the computer-oriented academic community. Colleges and Universities began to stock their new Computer Science programs with the minicomputers of the 1970's rather than IBM System/370s or System/3s

that were available at the time. The apparent bias for these new minicomputers was so strong that IBM was often not even invited to propose its solutions to higher learning institutions. Thus, more and more computerists emerged from colleges and universities without ever having been introduced to RPG business programming.

RPG and AS/400 Kill the Minicomputer Revolution

Though RPG has proven itself well over the years, providing the production data processing for many successful companies, using IBM's small-business computer lines, the unwarranted bias from academia and from the computer science community continues. RPG is the last language to be considered in computer science departments.

Despite this fact, until the mid 1990's, IBM's new account marketing engine with the RPG-oriented AS/400 as its major tool, absolutely defeated all of the minicomputers that ever existed: DEC, Data General, Wang, and others. One must ask how wrong the academicians must have been to embrace technology that was so easily defeated by IBM in the business world, where the computing rubber actually meets the road.

Did the rigidity of the academic computer science community against IBM's unique business computing approach help or hurt students in these institutions? Clearly it hurt students looking for a well-paying computer position in industry. For IBM to overcome the disadvantage of graduating students not being familiar with its best-selling products, IBM developed its own courses and its own education centers and companies trained non-degreed personnel to fill the jobs that would have been easily gained by knowledgeable students. In many academic computer programs today, this anti-IBM scenario persists has not changed and students continue to be short-changed on higher-paying job prospects than those available in the entry Windows market.

IBM's victory over all minicomputer vendors was so complete that even the mighty Hewlett Packard had to give up on its own proprietary line of computers. Even before its merger with COMPAQ, who had bought-out DEC several years earlier, HP had become mostly known as a small printer company. IBM's System/36, System/38, and finally its 1988-introduced AS/400 lines and today's System i5 won the day for Big Blue

and the business oriented RPG language was the major weapon to help make that happen.

Chapter 2

The History of RPG

Filling in the Blanks

In describing RPG, I touched on a number of the historical aspects of the language. In this section, it's time to fill in the blanks to better present the origins of a language known as much for its strong buseiness capabilities and success as it is for the legions of computer science types and hackers who refuse to touch it.

In the Beginning

The beginning of data processing as we know it in themodern era actually began back in the 1890s, some 25 years before Tom Watson Sr.'s, coming as an outsider from NCR Corporation. Watso would waste no time taking over the IBM company as its CEO. While toiling in the 1890's trying to solve a major dliemma of the US Census department in getting its once a decade census tabulated in less than ten years. Herman Hollerith found a solution that would keep the IBM company going for over sixty years. While working for the Computing Tabulating and Recording Company (CTR) one of the predecessor companies of which IBM was founded, he devised a machine to tabulate punched cards to help complete the 1890 census and save the day for the Census Department.

His invention led to many similar and complimentary products by IBM over the next sixty or more yearts until the ocompany launched its first computer in the 1950's. But, way before the 1950's IBM was content making a killing selling what the company called tabulating equipment but

which the industry simply referred to as "Tab" machines. In 1934, for example, IBM introduced its 405 Alphabetical Accounting Machine. This was IBM's high-end Tab offering, and by the way, it was the first one to be called an "Accounting Machine."

Fixed Cycle Accounting Machines

The 405 was "programmed" by a removable plug board with over 1600 functionally significant "hubs", with access to up to 16 accumulators. This high end mechanical monster could tabulate at a rate of 150 cards per minute (CPM), or tabulate and print at 80 cpm. The print unit contained 88 type bars, the leftmost 43 for alphanumeric characters and the other 45 on the right for digits only. The 405 was IBM's flagship product until after World War II. In fact, during the war, retrofitted 405s were used not only as tabulators but also as the print device for top-secret relay calculators built by IBM for the US Army Signal Corps. They were used for decrypting German and Japanese coded messages. Eventually, IBM introduced a model 402, model 403, and a more advanced model 407 for businesses in the 1950's. All of these models were big revenue producers for the IBM Company and they helped many business operations become more efficient.

In the late 1950's to the early part of the 1970's while IBM was beginning to focus on its emerging computer lines, the company was still renting these behemoths to smaller and smaller companies. It was not until 1969 that IBM formally replaced its Tab line with its diskless all-card System/3 and its miniature 96-column card. For a number of years thereafter, IBM's new account sales personnel sold first time computer users on the IBM Tab line since these old war horses rented for about half of what a System/3 cost. After all, IBM only permitted its customers to rent its machines back then so the longer one of those electromechanical marvels was doing its job, the more profitable the experience was for IBM.

The RPG cycle was the key to making the language an initial success. The cycle enabled processes during which an RPG program automatically read a record and performed certain routines. This fixed cycle was at the heart of file processing. Unlike other high-level languages, RPG didn't require a lot of work with file declarations for opening and closing files, and

working with files, nor did it require a complex list of instructions to simply print data. The infamous RPG
cycle took care of all that for the programmer. Those who learned how to work with the cycle were far more productive than those programmers who worked with other languages.

RPG on the IBM 1401

In 1960, the RPG language was built for and made popular with the IBM 1401 transistor based business computer. In 1964, RPG was upgraded to a more usable language. This improvement coincided
with the announcement of the IBM 360 model 20. While RPG did a fine job of handling 80-position cards, it couldn't handle tape or disk processing, not to mention display devices—which were just being introduced. By the later 1960's, however, the call to make RPG a "real programming language," was answered with the introduction of RPG II.

The RPG Cycle

The RPG cycle (the processes during which an RPG program automatically reads a record and performs certain routines) was at the heart of file processing. Unlike other high-level languages, RPG didn't require extensive file declarations for opening and closing files, nor did it require a complex list of instructions to simply print data. The RPG cycle took care of that for the programmer.

Here Comes the IBM System/3

The story of RPG would be incomplete without a discussion of the first computer ever made by IBM to be an RPG-only machine. When IBM decided to retire its older card processing "TAB" units, the company still believed that there was still opportunity in the small to medium business market for a lighter, less costly family of card processing gear. IBM's plant in Rochester Minnesota had been built in 1956 specifically to build card readers and card punches for computer systems.

IBM computers were built only in Endicott and Poughkeepsie at the time, so it was understandable that the Rochester Plant and its research arm, built in 1960 got the nod to design and build the next generation of punch card processing equipment for the IBM Corporation. I repeat, Rochester was not a plant that IBM had selected to produce any type of computer equipment since its New York facilities already had that covered.

Rochester sold the idea of a card-only System/3 using a multifunction card unit (MFCU) to corporate IBM and along with its accessories (96-col keypunch and sorter) the Minnesota plant produced a modern version of IBM's old warhorse tabulating equipment. Though Big Blue envisioned nothing more than newer and better TAB equipment, the Rochester folks had bigger ideas. The machine the company announced in fall 1969 with no disk or tape or any other magnetic or optical media capabilities used 96-column cards-only. Thus, it satisfied IBM's desire for a modern unit-record system but it was clear that there was more to this new machine than met the eye. The underlying architecture of the System/3 eventually permitted it to grow into being lots more than IBM expected or wanted at the time. But, that is another story.

System/3 as a 407

To simulate the functions of fixed cycle of the 407 Accounting Machine, IBM adapted its RPG language. The System/3 had two card hoppers in its multi-function card unit (MFCU) package so RPG II was enhanced to be able to directly access both of these card reader / punches in one program. Since both card hoppers could read and both could punch, RPG II was given the ability to punch data into the same card that was read. Additionally, RPG programs could also punch out new decks of cards from blanks. Since the MFCU could not print on cards, RPG did not gain this ability either. However, any card output that required interpreting was sent through the 96-column card keypunch unit that IBM called its 5496 Data Recorder.

System/3 RPG Was RPG II

Though the RPG language design for the card-only System/3 models was not much different from that which preceded it, technically it became known as RPG II. With its new disk drives, the 1970 version of System/3 RPG II was a much more capable programming language. The major

operators included for DISK were the CHAIN and the EXCPT, which are explained in detail later in this book. By introducing these operation codes, for the first time in RPG, programmers were able to access and update/add disk records both randomly and by key (index files) within the confines of the RPG calculations specification form. These operations, in essence were the very first operations in RPG that did not depend on the RPG cycle for execution. RPG II was well on its way.

By 1970, IBM customers were demanding disk data storage and tape processing for the system. IBM obliged and it introduced its 5444 disk drives which conveniently mounted under the MFCU in two drawers. Each of four disk platters could hold up to 2.45 million bytes of data. A fixed and a removable platter were housed in each of two drawers. Of course, IBM would also sell one drawer units to those needing less disk capacity. Eventually the System/3 grew with 5445 and later 3340 disk drives to hold as much as 180 MB of data on four large disk drives.

The Four RPGs

So, with the IBM 1130 and the disk enhancements to the System/3, the RPG language compiler as originally announced for the 1401 was enhanced and re-announced as RPG II. From the very early RPG "I" language introduced to mimic the electromechanical machines, IBM has been continually improving the language. Historically, these improvements have brought four different RPG languages to market

RPG One

Prior to RPG II of course, with RPG I and the 1401, the programmer used the many statement types defined by the RPG language to interface with what we now refer to as a "fixed logic cycle." With the fixed logic cycle and the RPG code as "written" by the programmer, the 1401 computer could pretend that it really was a 407-style accounting machine and that the program as devised by the programmer provided the variables and logic in much the same fashion as the removable wired circuit panel of its electromechanical predecessor.

It is hard to imagine that an electromechanical machine of the early 20th century could support 150 fixed cycles per minute. Yet, the "advanced"

407 unit was capable on the average of reading a two card records from the one hopper in less than one second, passing the data from the card record to the output print line via "hard wires," and tat the same time, counting, totaling, and printing output lines.

When RPG was used to mimic this operation, just as the 407 itself, the amount of work that was completed was dependent on the speed of the card reader and the speed of the printer. The System/3, though very slow by today's processing standards, was much faster than these mechanical devices which in fact restricted its speed. If the card reader read slower then the printer could print, then the program was said to be card reader bound. If the printer were slower, the program was printer bound. Printer and reader spooling were not yet invented for small computers and were still years away.

The now famous RPG fixed logic cycle performed various types of work during the cycle. If the programmer told the RPG cycle that there was work to be performed in any of these component areas of the cycle, the cycle would cause that work to be accomplished. In its most simplistic terms, the cycle started by being able to print report headings on output, then it read cards, it then calculated and totaled fields and when it started the cycle again it was ready to output a record, which most often was in the form of a print-line.

RPG II Adds More Capabilities

From 1965 with the IBM 1130 and soon after the System/360 Model 20 through the end of the 1960s with its Ssytem/3, IBM made the new RPG II programming language something that had to be noticed. The 1130 for example was marketed by IBM as a scientific computer, having superseded IBM's long standing 1620 unit. However, when the 1130 was equipped with an RPG II compiler, it became a very capable business machine. The IBM plan did not have any business-only customers looking for success by installing an IBM 1130 unit. However, the IBM Company did not refuse these orders. With RPG II as a business mainstay, the 1130 became far more popular than it would have ever been with just Fortran as its guiding light.

Businesses of all kinds soon were able to choose to use this disk-enabled small machine for scientific, business and accounting purposes. For

example, the Administration of Marywood University of which I am a faculty member, back in the early 1970's, chose to use one of these 1130 units as its main administrative system for many years --- until they eventually switched to a System/3. Mercy Hospital in Scranton PA also found the 1130 with RPGII as an ideal system for patient billing and accounts receivable. RPG II was the main business language of the late 1960's and before the System/3, IBM's powerful but little 1130 unit carried the RPG water for Big Blue.

Instead of depending solely on the fixed logic cycle as in the 1401, the major structural improvement that RPG II brought to the fore included the ability to read and write at will during what we will now call "calculation time." The original RPG language did not include an ad hoc read, write or update facility. All reading and writing was done via the RPG cycle. With RPG II, IBM introduced the random read "CHAIN" operation for ad-hoc input and the company introduced the exception output "EXCPT" operation for ad hoc updates, adding records, or printing report lines.

RPG purists have categorized these new facilities in the language as the beginning of its degradation as a powerful report writing tool. In many ways, they are correct since the new capabilities in the language I brought it from its origin as a fixed cycle facility to a real programming language. The enhancements made it substantially easier to code business logic without language constraints. RPGII did not eliminate the RPG cycle, and in fact, required that the cycle be used at least once time in order to arrive at "calculation time." The RPG cycle, for example, still is viewed by those who understand it as the most appropriate vehicle for reports.

In 1972 IBM enhanced RPG II by adding the KEY and SET operations. These were enhanced with the use of the mini CRT on the System/32 desk sized computer announced in 1975. The KEY and SET operations allowed the programmer to accept input and display output. It was simple, fast, and effective and it permitted real interactive work to be done on the IBM System/3 Model 6 and later the System/32. In 1977, This facility was enhanced to work with the System/34. Additionally, IBM introduced the revolutionary workstation file with RPG for the first time with the System/34. This permitted full-panel workstation devices (i.e., dumb terminals) to be deployed as natural devices in an RPG II program.

In 1977 the WORKSTN device was a real phenomenon. IBM defined the notion of a display screen. The displays screen was defined externally to the program and was manipulated within the program using normal RPG operations against screen names. An extension was added to the RPG File description specification to permit what was called a format member to be compiled along with the program. From this member, the programmer could select screen names for output / input to interact with a user.

Prior to the native WORKSTN device support for the System/34, display terminals were not ever integrated into compilers. In fact, terminals were supported only via special add-on support in the form of the Communication Control Program (System/3 CCP) or the Customer Information Control System (System/370 CICS). Both CCP and CICS had their own system generation process and specialized operation codes such as Get and Put. Moreover, these tools required skills above and beyond that of a normal programmer. The WORKSTN file was so easy to use that many who had become adept at CCP or CICS could not believe that it could possibly work. It worked, and it made the RPG language the easiest to use for business full screen at a time interactive processing.

RPG III

In 1978, IBM announced a machine that was so elegant architecturally that it would take the company almost another two years to deliver its first customers shipment. It was called the IBM System/38 and it was a minicomputer class machine but IBM liked to call it a small business computer. The System was built to be the replacement box for the IBM System/3. It was clearly the most advanced general-purpose computer of its day, complete with a built-in relational-like database management system and natural workstation facilities that were far better than even the System/34.

With the System/38, IBM introduced the RPG III language which brought a host of new functions to RPG, among them a nearly complete set of structured programming operations (e.g., IF-THEN-ELSE, DO). With these new features, programmers were able to define RPG programs which did not require even a thread of the RPG cycle. IBM did not eliminate the RPG cycle with the System/38 and in fact, along with all file input/output facilities, the cycle was enhanced to use externally described

files. The System/38 permitted the input and output of files to be described externally such as in a workstation file object or a database file object. At compile time, the programmer merely added a switch in the File Description Specification to tell the compiler that the input and output specs for a file were included in an external object. The compiler would then dutifully go to the object and bring the specs into the program, provide them in the compiler listing and make them available for use within the program. This saved the programmer massive amounts of time.

In 1983, IBM replaced its System/34 line of computers with its brand new System/36. Many of the structured operations that were given to the System/38 in 1978 were made part of the System/36 RPG II compiler. However, because the System/36 was not object based, the notion of externally described data remained a System/38 exclusive. Though in many ways the enhanced System/36 RPG was more like RPG II ½ than RPG III, the name for System/36 RPG stayed as in the System/34 – RPG II.

In 1985, System/38 RPG III was again enhanced to include support for and/or logic within IF and DO operations. This support greatly enhanced program readability and greatly decreased programmer frustration levels. Additionally, those familiar with such structures on other systems and other compilers more readily adapted to learning RPG.

In 1988, IBM introduced the AS/400 and the company provided another new compiler called RPG/400. Because of the many flavors of RPG that were currently being used, IBM packaged all of its exiting compilers into this new edition. Thus, on the new IBM AS/400, the RPG compilers that were available included the following:

1. RPG38 System/38-compatible RPG III.
2. RPG36 System/36-compatible RPG II.
3. RPG AS/400-compatible RPG III.

There was and still is little discernable difference between the System/38 RPGIII and the AS/400 RPG III.

RPG IV – Best Language on any Platform

In 1994, IBM introduced the first significant update to the RPG language in more than 15 years—ILE RPG a.k.a. RPG IV. The introduction of RPG IV marked the first time ever that the RPG specifications have changed. In fact a new data definition specification was added and the long suffering File Extension specification form was eliminated from the new language.

Significantly since the language was originally developed. RPG IV also eliminated virtually all of the perceived limitations of previous versions of RPG. With RPG IV, IBM has added natural expressions to the language in the areas of mathematics and conditioning, and has created leading-edge DATE and TIME arithmetic operations.

This latest version of RPG is, by far, the richest language in existence. Though it is more capable and therefore more complex that the RPG of the 1960's, it is still easy to learn and it offers a rich set of functions for day-to-day, general-purpose business applications. With its reasonably new free form facilities and its many built in functions, and its use of procedures, it also has great affinity to the block structured languages used within the computer science community. As such, it makes it easier for System i5 programming shops to train today's college graduates for a career as a System i5 IT professional. Moreover, the concepts learned are applicable to other programming languages.

So, in 1994 with version 3 Release 1 for CISC and in 1995 with Version 3 Release 6 for RISC architecture, IBM brought forth its fourth iteration of RPG, dubbed RPG IV by those who use it but officially named ILE RPG for its dependence on the more advanced Integrated Language Environment in which it prospers.

The Integrated Language Environment is a programming model that permits highly modular code to perform well on AS/400 type machines. It also enables all languages written within this model to better cooperate when working together. The former model now called the Original Programming Model (OPM) is not as functional and robust as the ILE model, yet it still is supported on the IBM System i5.

Some of the improvements to RPG with RPG IV include the following:

1. New "D Specification: In addition to the specifications for RPG/400, IBM introduced a new specification form called the "D" spec. All non-external data definitions can now be specified in a D-specifications that are new to ILE RPG. In addition you can define "named constants" that greatly simplify coding these in the C-spec's. Also C-spec formats have changed slightly to provide for variable names of up to 10 characters (up from 6 in RPG/400) and longer operation codes.

2. New Operations: Several new operations have been added. One that rovided the ability to code math in a formula-like fashion is the EVAL operation. In essence it permits you to evaluate a mathematical expression similar to Cobol and other mathematical programming languages such as Basic, FORTRAN, PL/1, etc.

3. Modularity: With ILE, you can now write modules (non-executable) in several languages and bind them together into a single ILE program. This program can be an RPG IV program. You can also use RPG modules in other language programs. Thus you can use the best language (ILE C, ILE Cobol, ILE RPG, ILE CLP) for a process or you can use existing modules to write a program.

4. Larger size fields: With RPGIV, the RPG spec has been widened to 100 characters to accommodate up to ten character field names and larger operation codes.

5. Date fields / operations: One of the first major differentiations between RPG/400 and RPGIV is the new compiler's ability to deal with the date data type. For example operations exist to subtract a duration from a date and get a duration or to subtract two dates and have the result presented as a duration.

6. Procedures: IBM has also built into the language the notion of callable procedures.

7. Built-In Functions: Many built-in functions or BIFS have been added to the RPGIV language including *%date, % days, %months, %years, %diff, %abs, %editc, %subdt, %DEC, %INT, %UNS, %FLOAT, %error.*

8. Free format RPG specifications: We may joke that this is not your father's RPG because it isn't. In fact, with RPG FREE form, IBM has given the RPG programmer the opportunity to code without the typical columnar boundaries of RPG/400. As an added benefit, IBM has also provided the ability to add free-form SQL statements within the RPG language to make RPG an even friendlier language for those who choose to use embedded SQL for database access.

The Once and Future RPG

IBM continues to enhance RPGIV for System i5. Yet, the enhanced language is not available on any other platform. Industry consultants have suggested to IBM that it make this very powerful business language available on all of its platforms. Consultants have also suggested to IBM that since RPGIV is more than ten years old, it is time fro IBM to rename RPG. The natural next name, of course would be RPG V but some consultants have suggested that IBM give the language a name that releases it from its "legacy" status.

From its name, the computer science community has pegged RPG as an old language and it is tough for RPG developers to get from under that label. Thus, an all-out assault by IBM is being requested in which three major changes need to occur with the language. These are as follows:

1. New name that includes both business and power connotations – such as The All-Business language.
2. Availability on all other platforms from PC servers to Unix, Linux, and mainframe boxes. After all, it is IBM's best business language.
3. Natural Web operations within the language specification in the form of a browser device file and op codes to send and receive Web pages of any form – HTML, XML, JSP, JSF, etc.

Chapter 3

Understanding the RPG Fixed Logic Cycle

Cycle Operations Are OK!

When RPG got its first face lift in the mid 1960's with RPG II demand (out-of-cycle) operations, the RPG language purists were those who wanted the cycle to be fully preserved at all costs. They were not happy that RPG had lost its innocence and was on its way to becoming a real programming language.

Ironically, those who would be called RPG purists today are those who want to keep RPG on the bleeding edge of compiler functionality. Prior to V5R4, for example, there had been clamor in these ranks for more built-in functions, such as those available in Java and there was insistence that IBM provide a move corresponding type operation in much the same fashion as that available in COBOL. This group is driving the RPG language in such a way that like PL/1 was supposed to be in the 1970's RPG may very well become that one language that offers the full gamut of compiler innovations.

In the latest enhancements, the language draws new facility from both Java and COBOL, two completely different languages. With all of the new facilities being placed with the RPGIV language, perhaps the ultimate destiny for RPG is even more than the All-Business Language moniker that many of us would like. The Once and Future RPG may very well become the All-Everything Language just as the System i5 itself has become the All-Everything machine.

I mention the new purists because they would not approve of me first presenting the RPG cycle in this book or even presenting it at all. These

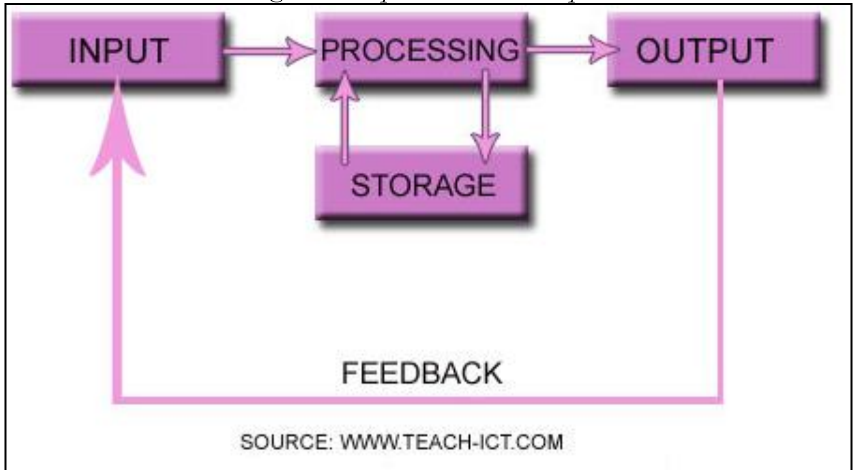
modern purists have wrestled RPG away from its roots so far that many would be pleased that its roots disappear. They would not approve of 4GL or 5GL languages either since their world needs to include pointers and heavy duty built in functions that rival the most complex languages of today.

Despite the purists, I begin this technical work with the RPG cycle because (1) it affords me the opportunity to examine the meaning of the specifications in the RPG/400 language right along with the RPG cycle; (2) Many RPG/400 cycle programs in RPG shops continue to be used and continue to need maintenance; (3) Even those shops that have converted to RPGIV for new development still use the RPG IV cycle for those programs that were migrated since RPGIV fully supports the RPG cycle; and (4) For applications that must produce printed reports, the RPG cycle still provides the most efficient means of report writing short of using 4GL tools. For report-writing in fact, RPG with the fixed logic cycle in use is very much a 4GL unto itself.

A Quick Look at the RPG Cycle

Long before programmers learn their first language, they are introduced to computer concepts. Within the notion of basic computer concepts is the notion of INPUT > PROCESS >OUTPUT as shown in Figure 3-1

Figure 3-1 Input Process & Output



Just about any computer program that you will ever write accepts input, processes the input, and produces a report of some kind. The report may be a line on a screen or a full display panel or a real business report. Moreover, as you can see in Figure 3-1, in addition to producing output in the form of a report or display, a program also can store data in database files (storage) for future use.

So, whether you write the cycle yourself in every program or you choose to use the IBM RPG fixed logic cycle, which by the way is excellent for report-writing, your code will behave as if it is in a big INPUT> PROCESS> OUTPUT cycle for in fact, it is.

So, it is no wonder that the very first data processing machines, the electromechanical behemoths from the 1930's were hard wired to this cycle. It is also no wonder why the first RPG compiler, written to emulate these machines was soft-wired to this cycle. And, because it eliminates coding if you know hat you are doing, it is no wonder why RPG programmers for years had no problem using the RPG cycle for their most complex reporting functions.

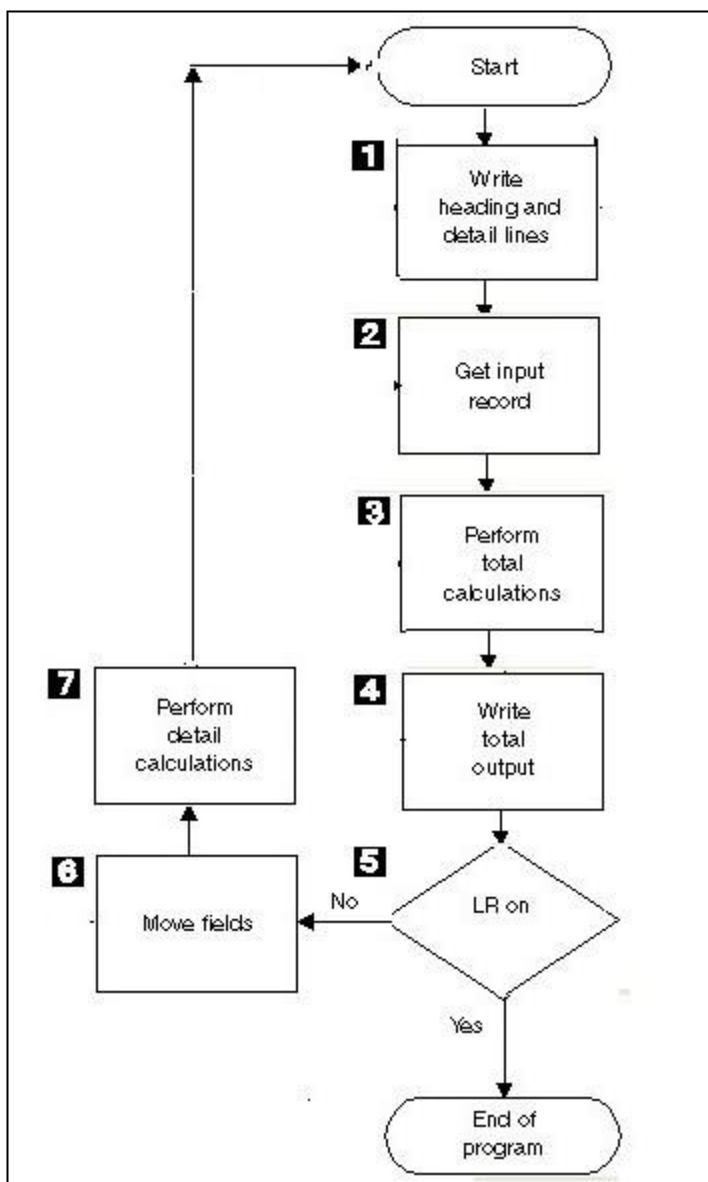
So, now let's take a look at the RPG cycle so we can understand just what it is all about. Before we do that, however, I would like you to think of

what the first thing that a report writing program might do – even before it reads or processes any data. If you guessed that it may put out report titles for the first page of the report since they do not depend on any data being read, you are correct. And, if you carried this notion one step further and suggested that they also prepare the titles so that they can be put out on top of every other page in the report, you would be even more correct.

The RPG cycle is built so that report titles are a natural part of the language and they occur first. Thus as difficult as it may be to realize at first, output happens before input at the beginning of each RPG cycle from the first through the end of the program. Rather than start with input, it starts with output thereby giving the program the opportunity to get those titles on the report before the first data is read.

With that as a backdrop, and without further ado, let's take a look at the RPG cycle first shown in Figure 3-2.

Figure 3-2 The 7 Major Steps to the RPG Fixed Logic Cycle



The various work components of the RPG cycle as shown in Figure 3-2 are as follows:

1A. For reports, first page headings that you specify are printed.

1B. For reports, overflow headings that you specify are printed. Overflow headings are those headings that are repeated on each page after the prior page hits a bottom of the page overflow

1C. Detail output is printed. On the first cycle, since no data has been read, there is no detail data to print. Detail output consists of any output line that can be written or updated to any device – printer, disk, tape, etc. Data fields that are read during the input part of the cycle in Step 2 are held in memory as variables and are not made available to the program until Step 6. These variables may be operated upon in calculations and their contents can be modified. New variables can be defined within calculations and these can store the result of operations. The newly defined variables as well as the input variables or the modified input variables as well as constants can be placed in the output record that is written.

It helps to repeat that since no input data is available during the output part of the first fixed cycle the program produce no output during the first output time of the cycle. If you follow this logic, it also says that any input that is read that must be printed will not print until the next output time in the cycle. Thus, data read on the first cycle in step 2 is not available for output until at the beginning of the second RPG cycle at step 1.

2. The first record is read from the primary file. Though the record has been read, however, RPG does not make the data available to the program until Step 6 of the cycle. When coding with the cycle this needs to be considered since on cycle 3 for example, the record read in this step of the cycle is not the record whose fields are available to the program. The prior record's data is still in the program fields until Step 6.

This may not make sense right now but it will as you study the next parts of the cycle which include total calculations and total output. When RPG is finishing up the totals for a group of data, this event has been triggered by the cycle knowing that the record about to be read into the fields for processing is the first record of a new group. In Step 2, RPG has learned that the next record is from a different group. When writing reports, that

means it is time to perform any special calculations that must be done at this time of the cycle and then it is time to write out the totals.

In many ways, Step 2 is RPG peeking ahead to see what the next record is so that it knows if it should take totals or if it will keep reading records from the current group. This facility also comes in handy when doing matching records so that RPG can look ahead to see the sequence of the matches so that it knows which record to move in for processing.

3. Total calculations are then performed. Total calculations are totals that occur after major, intermediate, or minor control field breaks. Since during the first cycle, no data has been read at this point in the cycle, there will be no control level breaks and therefore, no control level calculations will be performed. However, there will be calculations whenever control fields change in subsequent cycles.

To understand how control level calculations work, consider the following example. If the program is written to total gross sales by city, when the city control field changes from say Kingston to Plymouth when the second record is read, then during the third cycle, this would cause a minor (least important) control break. This is also called Level 1. During the period in which the cycle provides some time for total calculations, it starts by enabling Level 1 calculations followed by Level 2; then Level 3 until finally it reaches Level 9, the maximum number of levels as can be defined in an RPG program. In the city example, during this opportunity to take totals, the programmer would tell the RPG compiler to add the city total of gross sales to the state total of gross sales and store it for the level 2 total on state.

If we decided that we would also like state totals in this report, since the state is a higher level organization than the city, we could use the next highest control break level or L2. Then when the state changes from say Alabama (AL) to Alaska (AK), during the L1 cycle, the programmer would tell the RPG compiler to add the city totals to the state totals and during the L2 cycle, the programmer would tell the RPG compiler to add the state totals to the final total accumulator.

4. Total output is printed. Total output includes totals that have been accumulated, such as the field in which we store the city total and the field in which we store the state total. When the city total changes, for example,

we probably want to print the total on the report with the City name next to it. This type of output would occur during the L1 output cycle. If there were no control breaks during this card cycle, then no output triggered by a control indicator (L1 or L2) would be produced.

If the L2 control field changes (state in this case) that automatically means that an L1 break also occurs. (If the state changes, by definition, you can bet the city changes). So for an L2 break, L2 totals are produced as well as L1 totals. Likewise if an L9 break occurs, all other total levels from L8 to L1, in addition to L9, are prepared to produce output. Likewise if a final total occurred, it would automatically create a level break at the highest level defined in the program – L9 at the highest but just L2 in our case with State and City.

5. The program checks to see if it should end by checking the LR indicator. A special "indicator" called LR causes the program to end if it is set on when all of the files that are being processed are out of records or if this indicator "LR" it is set on in detail calculations from the last cycle. When the program sees that LR is on at this point of the cycle, it ends the program gracefully and closes all of the files.

6. Input is processed by moving input area buffer contents to the RPG fields that you defined to hold them. When RPG has completed performing level (control break) output (This can be disk, tape, card, printer or any output conditioned by a level indicator at total level time), the cycle moves to input processing time and it populates the fields in the file read in Step 2 with the data that is stored in the input buffer.

Step 6 works hand in glove with Step 2 but they occur at different times in the cycle. The following pertains to Step 2 and Step 6 as if theya re one input routine. This is not how it really happens but it will help in understanding the interrelationship.

7. Detail calculations are performed. What are detail calculations? They are calculations that get performed when records are processed. Detail calculations are permitted to occur for each record that passes through Step 6 of the RPG cycle as shown in Figure 3-2. If, for example, each record contained the order quantity and the price, each RPG cycle as coded in the C specifications, the programmer can say to multiply the quantity ordered by the price and create a field called the extended price.

Since basic RPG permits just names of six characters, the result field name might be called XPRICE or something else to properly reflect its meaning.

Since RPG calculations have what are called conditioning indicators, calculations are not necessarily performed each time that the program passes through the detail calculations part of the RPG fixed logic cycle. If, for example, the master and the time card records are being processed through one file, then, assuming that masters are sequenced before corresponding time card records, the programmer may wish to take action only when a time card record is read. In that way, the information from the master such as the pay rate can be used when the time record is read and the program can multiply the two to get the gross pay amount. To set this up, the programmer would assign an indicator to the master and an indicator to the transaction file and those RPG cycles, in which a master record is read, the calculations can be conditioned not to occur.

Additional INPUT Processing Information

For example, if there is one card reader / hopper defined to the program, after RPG reads a record the cycle would keep repeating Steps 2 and 6 in turn and would in essence keep going back to read another card or disk record. If there are two card readers as in the old System/3 and both are defined in the program, or there are two disk files defined, RPG will keep going back to the first hopper or first file until it is empty or the disk file is at end of file. At this time, RPG would begin reading data from the other hopper or the second disk file until it is empty. When both hoppers are empty or the disk files are at end of file indication, RPG automatically turns on a switch called the last record indicator or LR and it declares that the program can end and then it ends the program as described in Step 5 above. .

It is unusual to have two files that need to be processed and the job is such that one can be processed fully before the other is even begun. Yet, with no special coding in RPG, that is what occurs. In real business processing situations, however, the program requirements most often dictate that the records from one file be interspersed with the records from the second file.

RPG Matching Records Processing

For example, you may have master payroll records specified as *File 1*. You may have time cards specified as *File 2*. To prepare for this computer run, which depends on both files being in sequence by say, employee number or a field called EMPNO, you would sort the both files by employee number so they are in the same sequence. Your program logic would want to be able to read the master payroll record for employee 1 and then the time card for employee1 and then it would want to read employee 2's master and then employee 2's time card and so on until all employees were processed. Once the program reads the master and the time card for each employee, it has enough information to calculate the payroll.

During the input part of the cycle in Step 2, RPG provides for a period in which two or more files of records that are all in sequence by the fields specified for the match can be read and be declared a match. In this scenario, the time cards and the employee master file would be in different card hoppers in old systems or they would be in different disk files in more modern System i5 units. This notion is referred to in RPG as "matching records" and it is a big plus for using the RPG cycle for report writing functions when two or more files are needed for processing. It is also handy when updating master files from transaction files.

To use the matching records facility within the RPG cycle, the programmer must have a very good understanding of the detailed RPG cycle since the needs of the program are more involved than when using just one input file. In addition to telling the compiler which file should be read first (primary file – pay master) and which should be read second (secondary file – time card), the programmer must also designate the name(s) of the field(s) that must match. In this example, the one field name is EMPNO in both files.

RPG provides a designator that can be placed next to the field names in both files within the RPG Input Specifications. In other words, the programmer must place a designator in a field from each of the files to be matched. This can be referred to as a matching designator. If there is just one field to match, then the only matching designator required is known to RPG as "M1." However, unlike the present example, which needs just one field (EMPNO), if there is more than one field that must be matched

(and up to nine fields), then RPG makes available other matching designators --- M2 through M9. When more than one matching designator (M2 to M9) is specified, all match fields that are specified must be matched from each file before the special "indicator" known as "MR" for match indicator is as we say, "turned on."

So, let's say in our simple example that we have two files. We mark the employee number fields (EMPNO) in both files in the RPG program input specs with the M1 designator. From here on, when this program runs, RPG itself makes sure that both files are in sequence and when there is a match, it turns on a special matching switch called the matching record indicator or MR. The programmer can then use the status of this switch (on or off) to cause desired events to occur in the program that otherwise would be difficult to achieve. Additionally, if by any chance the files are found by RPG to not be in sequence by the matching designators, the program will halt with an error condition.

Record Identifying Indicator Processing

There is another phenomenon in the RPG fixed cycle called the record identifying indicator. Each time RPG reads a master record with the cycle, for example, it tells the programmer from which record from which file it read the record by turning on an indicator that the programmer associates with the file. If RPG turns indicator 01 on for example, that may mean a master record has been read and if RPG turns on indicator 02, that may mean that a time card record has been read. This powerful, yet simple communication gives RPG a simple way of permitting telling programs what is happening so that the program can take the proper action.

What is an indicator?

The RPG language provides a programmer with a tool box of 99 indications to use for their own purposes. The indication that a particular record type has been read can be stored in a special type of "field" called an indicator. To make it easy to remember, most programmers use various indications to change the value of the indicator fields from 0 to 1. Moreover, because these are special fields that exist in no other programming language, RPG also permits just two values in the field – either a "0" or a "1." When the value is "0," RPG programmers say the

indicator (numbered 1 through 99) is "off." Just like a light switch, when the value is one, RPG programmers say that the indicator is "on." So all 99 indicators in RPG can be tested to see if they contain values of "0" or "1" as well as being tested for "on" or "off."

With the notion of indicators and the RPG cycle, the compiler provides some additional facility. If, next to the input record on the RPG Input Specification, you choose to specify an indicator to turn on if a particular record ID is read, RPG will gladly turn on that indicator if it reads a record meeting the criteria you have specified. In the time card file and master file example, since we would only have one time card for one master, it would suffice to do no testing for record contents. Though RPG can look at the contents and turn on an indicator if the contents of a column contain a "T" (time cards) instead of an "M" (master) since we are reading the data from two different files, we already know that if it is read from the primary it is a master and if it is read from the secondary, if it is a time card. Thus, there is no further need to definemarkings inside the records to identify them.

So, for the record that defines the master file on the RPG Input specifications in the RPG program, the programmer assigns an indicator. Let's say the programmer choose indicator 01. Now, for the record that defines the time card in the RPG program, let's say the programmer chooses 02.

Once the programmer has done this, independent of the matching status or the reading sequence of the program, whenever the program reads a master record, regardless of its EMPNO value, record identifying indicator "01" will be turned on and record identifying indicator "02" (time card) will be off. That is because only one record identifying indicator can be on at one time. RPG processes just one record each cycle. Likewise, whenever the program reads a record from the time card file, independent of the matching status or the reading sequence of the program, and regardless of its EMPNO value, record identifying indicator "02" will be turned on and record ID indicator 01 will be turned off.

Primary and Secondary Files

The second last phenomenon that we will discuss regarding matching records is the notion of primary and secondary files. The 407 Accounting

Machine had just one hopper so it did not need anything on a wiring board to signal a match from two card readers. RPG is more sophisticated. It says that one file can be designated as primary and a second file can be designated as secondary for processing purposes within a program. If there are three card readers and/or many tape and many disk devices (files) in play, then theoretically, there can be as many – even twenty or more files defined in just one program. If these files are not defined as random or keyed-access files – in other words, they are to be processed sequentially, the programmer would need to designate all but one as secondary files in terms of matching records processing. Just one file in all cases can be defined as primary.

Internally, based on the order defined in the program, RPG would designate these files as tertiary, quaternary, quinary, senary, septenary, octonary, nonary etc. based upon the order in which the programmer specified them in what is called the File Description Specification in RPG. In today's processing, I rarely if ever see a tertiary sequential file. Multiple files other than the primary and the designated secondary are most often defined to RPG as something other than secondary – such as keyed index files or random files. However, you may run into someone else's code in which they have chosen to use more than one secondary file for matching records purposes.

The MR Indicator

The last phenomenon in understanding the RPG cycle with matching records is that the MR indicator comes on at a certain times in the cycle as does the record identifying indicators. They are turned off by RPG at a designated time in the cycle. For the novice RPG programmer, this is really a big pain to understand. "Real programmers" and RPG purists like to be in control of the program. Great programmers, however can take a powerful tool like "Matching Records," learn it inside and out, and be substantially more real and more productive than a real programmer. More importantly, by understanding the cycle, as noted earlier in this book, programmers can get much more done in report programs than by having to write the code themselves. To be good at the cycle, however, you have to understand the cycle. You have to know when RPG does what it does. You have to understand when RPG turns on record id and matching indicators and when it turns them off.

For example, if the pay rate is in the master record, we know that when it is read, indicator 01, the designated master record identifying indicator, is turned on. If the next master record has the same EMPNO field value as the record just read, RPG will read another record from the master file (primary). Since we have a one to one relationship of master to time card, this cannot happen. So, we know that the next record that RPG will read will come from the time card file (secondary) one input cycle after the matching master was read. Right before RPG reads the record for the time card record; it cleans up its act and turns off the indicator (01) that recognized that a master had been read. Prior to the time card record being read then, there are no record identifying indicators in the on condition. When RPG goes ahead and reads the time card record from the secondary file, it turns on indicator 02.

It helps to know that RPG can actually peek ahead at the cards or sequential disk records or tape records to see what they contain even if it is not going to read them on this particular input cycle. No, I am not kidding. RPG knows for example when it reads the master record from hopper 1 for EMPNO # 1 that there is a matching time card record for employee 1. It already knows that because it has peeked ahead. So, even though it is processing the payroll master record from the primary file with indicator 01, it is smart enough to turn on this special indicator called "MR" to indicate that the next time card record what will be read from the secondary file will match this master that is now being processed. The fact is that when the master is read, RPG "sees" the matching time card record sitting over there in the other file even though it has yet to read it. Yes, that is neat!

If the pay rate is in the master and the hours are in the time card record, when the master is read and indicator 01 comes on, the hours are still not yet available for employee # 1. So, we cannot calculate gross pay at that time in the cycle. So a condition that we might call "MR" and "01," means that we have a master read and being processed and its ***matching*** time card has not yet been read. Notice the word matching in the prior sentence. When we eventually read the time card, RPG has already turned off the master indicator (01) and then it turns on indicator 02. So, if you are keeping score at home, you might say that when the record identifying indicator "02" and the "MR" indicator is on, the primary record just read matches the secondary record that is now being processed. That is great

information for a programmer to gain without having to write the code for it. All you must do is be conversant with the RPG cycle.

Multiple Output Records per Cycle.

As an additional plus, for those systems such as the 1130 and the System/360 that were equipped with disk drives, as well as the later System/3 machines and every machine since, when the RPG language read an input record, it could also write a record in the same cycle while it was also producing a print line as what we call the detail output part of the cycle.

Actually, RPG could outdo its electromechanical predecessor by punching out many more cards in one cycle than it could read. Each of these card records, however, would have to be described in detail in the RPG output area. Each record described would be written as long as the conditioning indicators were satisfied. No, we have not described conditioning indicators in any degree of detail yet. Let's just say that they make output occur conditionally on whether certain indicators are on or off. When all output is done, the program with its fixed logic cycle always go back to read another record (96-column card in System/3s). When there are no more records (cards on older systems) to read, as noted in Step 5 of the Cycle above, RPG sets on its infamous ***last record*** indicator called "LR" and the program ends and is removed from memory.

RPG Fixed Logic Cycle Summary

If we go back to Step 6 and Step 7 of the cycle, once a given input record is read, the next step is that it is time again for detail calculations. This is the classic processing part of the RPG cycle that we have just described. The next step, of course is output at Step 1 and thus we have a complete cycle:

INPUT >> Processing >> OUTPUT

We learned that detail calculations occur in that part of the RPG cycle after the detail input has been read. The word "detail" describes what happens in each normal RPG cycle as an individual record is processed

(not counting total calculations or total output time). As noted above in the matching example, a detail record ID indicator of 02 means we have read in a time card. If we also have an MR match with the master record, then we know that the master record for that time card is also in memory.

If we are trying to assure that our output occurs after all of the information for one employee has been read, RPG helps again. We know that the detail time card record turns on indicator 02. We know that because this record matches a master, RPG is keeping indicator MR on for us. So, if indicator 02 is on and indicator MR is on, we know that RPG is processing a time card record and the time card record matches the master record that RPG read in the prior cycle. Thus, it is a logical time to condition some calculations that should occur when the time card is being processed for a matched master.

In other words, it would be quite appropriate to condition detail calculations to occur when both indicators 02 and MR are both on. In this case, we specify a calculation to multiply the rate times the hours to produce the first gross amount for the employee. Since the first gross amount per employee would not be an input field, the program would use RPG calculations to create this field for us within the detail calculation specifications.

So, as we wrap up this example in summary form, RPG will have read all of the data and performed the calculations and more than likely, because we have told it so, it will print an output line on a piece of paper that is in the printer. Once this happens, it is time to recycle – run the cycle again and again and again – until we run out of input.

So, the next generic cycle step is to skip back to the beginning of the RPG cycle to that spot that we designated above as the first step (1) in the cycle. Of course, we ignore the headings now since we are not on the 1st page since they occur just once in a program – to be able to print the first page information. They were done in cycle 1. Since RPG at this point is continuing to processing its first input record, it is highly unlikely that we printed enough detail records that the first page is full. Therefore, there will be no overflow processing on the second cycle and the next record cane be brought in at Step 2 of the cycle and then we finish up total calculations from the prior input record in Step 3 in the cycle and the cycle keeps moving. In step 1, it helps to remember that RPG will print

anything it can such as the contents of the last record read– not just headings as long as the output lines are either not conditioned or are conditioned with indicators that are turned on.

That actually about does it for our treatment of the RPG cycle in this book. To strengthen your knowledge of the RPG cycle, however, we provide a living example of the processing we have just explained in words. We defer this example to Chapter 4 in which we show the code to achieve what we have discussed regarding the cycle and as we fill them out, we describe the RPG specification forms so that you can get a better appreciate the form of this phenomenal language.

Chapter 4

Your First RPG Program

The Specs for Your First Program

Now that you have been introduced to the original notion of RPG programming using the RPG fixed logic cycle, let's use this knowledge to solve a simple business reporting problem. Suppose we have two files from which we would like to gain information – a payroll master and a time card file; and we would like to print a report in a very understandable fashion including totals and subtotals.

For this program, let's say we have the two files above – the very same files that we used to describe the RPG cycle, level breaks, matching records, and record identifying indicators above. Our objective, in the form that a programmer often receives a request for a program from as systems analyst is to print a report that looks like the report shown in Figure 4-1. The input for the report comes from two database files, the descriptions of which are shown in Figures 4-2 and 4-3.

Figure 4-1 Program Output from Running Sample Learning Program

```
          THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE          2/21/06

ST     CITY           EMP#     EMPLOYEE NAME         RATE     HOURS      CHECK

PA     Wilkes-Barre    001     Bizz Nizwonger        7.80     35.00     273.00
PA     Wilkes-Barre    002     Warbler Jacoby        7.90     40.00     316.00

                               TOTAL CITY  PAY FOR Wilkes-Barre         589.00

PA     Scranton        003     Bing Crossley         8.55     65.00     555.75

                               TOTAL CITY  PAY FOR Scranton             555.75

                               TOTAL STATE PAY FOR PA                 1,144.75

AK     Fairbanks       004     Uptake N. Hibiter     7.80     25.00     195.00
AK     Fairbanks       005     Fenworth Gront        9.30     33.00     306.90
                       006 NO MATCHING MASTER                 40.00
AK     Fairbanks       007     Bi Nomial             8.80     39.00     343.20

                               TOTAL CITY  PAY FOR Fairbanks            845.10

AK     Juneau          008     Milly Dewith          6.50     40.00     260.00
AK     Juneau          009     Sarah Bayou          10.45     40.00     418.00

                               TOTAL CITY  PAY FOR Juneau               678.00

                               TOTAL STATE PAY FOR AK                 1,523.10

NJ     Newark          010     Dirt McPug            6.45     35.00     225.75

                               TOTAL CITY  PAY FOR Newark               225.75

                               TOTAL STATE PAY FOR NJ                   225.75

                               FINAL TOTAL PAY                        2,893.60
```

To be able to print this report, you must be able to define the two files to RPG with the master as primary and the time card file as secondary. During first page or overflow time on the report, you want the report heading as well as the one line of column headings printed. For each master, there is just one time card. Print an error message if there is a time card and no master. If there is a master and no time card, let's just let that alone. When the time card record is read and there is a match with a master record, calculate gross pay and print a detail line on the report. Each time the city changes, total the columns shown on the sample by city. Each time the state changes, total the city and then total the state. As shown on the report. Each time the program ends – when the LR indicator turns on (only once in a program) print out the city totals, followed by the state totals, followed by the final totals.

This constitutes a summary of the written specs for the program you are about to write. Systems analysts are often more formal in their communications with programmers, but at a basic level, that about does it in informal shops. To know how to write the program, it will help to review the information I presented on the RPG cycle but, other than that, you're ready to go.

So, in the next chapter, we show you line for line how to write the RPG fixed cycle program that accomplishes the mission as defined. Well, not exactly! The fact is that I have not even offered you one thought on the various and many specification sheets that are the blood and guts of this powerful language. So, maybe you can't begin just yet. First, it seems that you must find out about those infamous RPG form types in which all you have to do is fill-in the blanks.

Well, not exactly! Sure, I could start there, but then this would be like every other book that has attempted to overwhelm the learner with every option on every RPG form – the easy and the difficult; the often used, the less-often used and the hardly ever used. I am not going to do that to you if you bought this book so that you could take a quick course or you could very simply gain the basics of the RPG language. Form what you learn in this book, you can grow into a more knowledgeable RPG programmer but when you leave this book, you will already know how to get a lot done with the language.

So, I will explain the option that you need to choose on the RPG specification sheet to perform the functions (each and every one and no more) that you need to do in this program for it to meet the specs as given and to produce the report as shown. By the way (BTW), systems analysts most often provide mock-ups of printer output, often in the form of a printer spacing chart (like graph paper) that shows where each constant and each field on the report is to appear. So, we have actually given a little more in this instance to show you how your output is to look.

Now is the time, if you are extremely adventurous to feel free to go out to IBM's Web site or another to learn as much as you can about the components of this program as coded on RPG sheets. You may even want to try to code the program with what you pick up on your Web trip. This will not hurt even if you mess it up badly. Even if you try to code the

program in pseudo code before you move on to the following mini-tutorial for this program, it would help you learn this topic more precisely. You'll learn why all of your documented assumptions are wrong – and that will help you learn RPG.

But, even if you choose to take the less difficult way out, which is also OK in this book, and in fact, which this book enables, after you sit back and relax and you read-on, soon you will have coded your first RPG program, and you will understand it. How about that for a challenge?

A Description of the Data

As those who have become acquainted with the iterations of AS/400 to the System i5 over the years will attest, the platform has a nice, proprietary, even snappy database language of its own in support of the relational database. So, instead of showing the data for these two files in an CREATE TABLE format, I have chosen to show it in a DDS format since that is the natural database language of the AS/400 historically.

Since most of the data defined in these two layouts are self descriptive (even without column headings and text), let me take a brief opportunity to describe the data in Figure 4-3 and Figure 4-3 as presented to you in these figures.

Figure 4-2 DDS Layout of EMPMAST Record

```
        *************** Beginning of data ******
FMT PF  .....A..........T.Name++++++RLen++TDpB..
0001.00      A          R PAYR
0002.00      A            EMPNO          3S 0
0003.00      A            EMPNAM        20
0004.00      A            EMPRAT         5S 2
0005.00      A            EMPCTY        20
0006.00      A            EMPSTA         2
0007.00      A            EMPZIP         5S 0
        ****************** End of data *********
```

Figure 4-3 DDS Layout of TIMCRD Record

```
FMT PF   .....A..........T.Name++++++RLen++TDpB.
0001.00      A          R TIMR
0002.00      A            EMPNO          3S 0
0003.00      A            EMPHRS         4S 2
        ****************** End of data ********
```

DDS provides a means of defining a field name and a length and an attribute of say alphabetic or numeric, and if numeric, then it also demands to know how many decimals. So, in the two DDS descriptions above, let's just say that PAYR and TIMR are known as record format names and these take the format of the file (all the fields) and permit all of the fields to be referenced in any program by this one name.

Following the record format name for a physical file, as shown in the DDS in Figures 4-2 and Figure 4-3, the database designer gets to describe all of the fields that make up the format of the record in the file – ie – the record format. EMPMAST (employee master file) has six fields defined for it and TIMCRD (time card file) has just two. IN all cases in the DDS shown in these two figures, the data being described is self evident.

The first number shown after the field name in the figures is the length of the field. If the field is numeric, more information is given. If it is character, such as a name, no more needs to be described. The length is all DDS needs and it will determine the start and finish positions in the records. The "S" shown in these examples stands for decimal data that has no strange IT format. In other words it is not packed data or binary or floating point or fixed point. It is merely numeric data – one number for storage position. You can say what it is without being a computer scientist and that is one of RPG's strengths.

Then in all cases, you see a number or a zero after the "S" in these field definitions. It appears only when there is an "S" specified. This is where the database designer tells the DDS compiler the number of decimal places that the database must remember about the numeric value when it is stored. The number "2" for example, as you would expect means that two positions of the length are reserved for the decimal positions. If the a value with three or more decimal position is stored in this database field, the value is truncated to two decimal positions. And, that's about all there

is to defining homes for data field values in the natural database for the System i5 – a.k.a. DDS.

The Data Itself

When you program in RPG or any other language, it is easy to make assumptions about how the data will look. As many times as not, when programmers are trying to figure out why the code of which they labored and labored does not work, and it does not appear logical that it does not work. Experienced programmers probably have just had a big laugh about this phenomenon. Beginning programmers always blame the compiler or the system for messing them up when a program does not work.

In almost all cases, the beginning programmer is wrong and they find that there was something not quite right with their logic. Then, as the programmers gets more experienced and they no longer are amused at blaming the compiler and being wrong, they rightfully check and double check their code to make sure they have not made a mistake that a peer would recognize immediately. In this stage, programmers become programmers. In fact, most who adopt the rule that the compiler is never wrong and this helps them in their quest for excellence. However, when they become excellent from this rigid discipline to which they subject themselves, when they hit a problem they absolutely cannot solve, they continue to blame themselves and look for a solution without seeking help or crying foul.

The irony is that as a programmer actually becomes excellent, other programmers may in fact be responsible for their apparent failings. If you are following along, you may have already figured out that not all compiler writers are not tenth degree knights. In fact, some of them are just out of college. And, though IBM and others have checks and balances to prevent bad code from entering the compiler or OS realm, sometimes it happens. Otherwise, there would be no IBM PTFs. When an experienced RPG programmer relearns that sometimes the compiler is wrong, they become even better at their trade.

But, to become good at what they do, they first had to subsume their initial arrogance and they had to discipline themselves into assuring that nothing was wrong with their code. They knew that they would not be

treated well by their peers if they continued to rant about the failings of others when the failings were theirs.

Most RPG programmers are amazed at their first taste of understanding their own excellence. Being trained to "blame me first," there is an initial shock in learning that the compiler or the OS was bad, and not the RPG code. Along their path to excellence all of them have learned about the problems with data.

No matter how foolproof the logic, if the data appears in what a programmer would offer is an "illogical sequence," the most meticulous code will not perform correctly. So, along the way to excellence, after they really understand the language, programmers learn to explain most of the inexplicable by taking a hard look at the input data during the debugging cycle.

It is in this vein that we offer Figure 4-4 and Figure 4-5 for your reading and digestion pleasure. These are pictures of the actual data that are in these samples. Since this is a sample world in which we are forced to live -- in learning a programming language, the data used to produce the results is the key to solving any issues that are not because our logic, as prescribed by the analyst is flawed. Often, the work of the systems analyst is necessarily incomplete or the phenomenon of analysis paralysis (over analysis) subverts the project. Only through major testing with real data can anybody – even the best programmers -- actually determine that bad data has entered the system.

Figures 4-4 and Figures 4-5 show the data, in query form to help you in your own desk-check logic, understand the real flow.

Figure 4-4 Query Listing of EMPMAST File Data

```
                                  Display Report
                                              Report width
Position to line  . . . . .                  Shift to column
Line   ....+....1....+....2....+....3....+....4....+....
       EMP EMPNAM              EMP EMPCTY       EMP   EMP
       #                       RAT              STA   ZIP
000001   1 Bizz Nizwonger     7.80 Wilkes-Barre PA  18702
000002   2 Warbler Jacoby     7.90 Wilkes-Barre PA  18702
000003   3 Bing Crossley      8.55 Scranton     PA  18702
000004   4 Uptake N. Hibiter 7.80 Fairbanks    AK  99701
000005   5 Fenworth Gront     9.30 Fairbanks    AK  99701
000006   7 Bi Nomial          8.80 Fairbanks    AK  99701
000007   8 Milly Dewith       6.50 Juneau       AK  99801
000008   9 Sarah Bayou       10.45 Juneau       AK  99801
000009  10 Dirt McPug         6.45 Newark       NJ  07101
****** ********  End of report  ********
```

Figure 4-5 Query Listing of TIMCRD File Data

```
                                  Display Report

Position to line  . . . . .                 Shift
Line   ....+....1....
       EMPNO  EMPHRS
000001    1    35.00
000002    2    40.00
000003    3    65.00
000004    4    25.00
000005    5    33.00
000006    6    40.00
000007    7    39.00
000008    8    40.00
000009    9    40.00
000010   10    35.00
****** ********  End of report  ********
```

Chapter 5

The Specifics of RPG Coding – Control Specification – by Example

From Coding to Decoding

Though the textbooks would show otherwise, the fact is that most programmers today do not get their programming requirements from a systems analyst who is designing a new application system. In fact, most programmers get little to no documentation at all from the requester. In most cases, no systems analyst exists in an RPG shop. Thus, the programmers in the mid-sized to small shops are referred to as programmer / analysts. The requester of a program change is typically a user whose change application has been approved by a steering committee or perhaps the IT Director.

Most small shops do not believe they have the time to deal with the formality of the natural change process as documented in the Systems Design and Analysis text books. Therefore, the information that comes from the requester or the IT manager in most cases is much lighter than the specs outlined in Chapter 3 under the heading, "the specs for your first program." The burden rests on the programmer /analyst to find the program(s) that must be changed and then, go ahead and make the change, test the change to make sure it works, and then put the change into production.

Once the program is found, the process of decoding must begin. Decoding an existing program is actually as important a skill as coding a program originally. Trying to appreciate the art that a brother or sister programmer deployed when making the code work the first time is most often more difficult than making the change itself.

In today's IT environment RPG detractors with computer science backgrounds will quickly point out their opinions that RPG is irrelevant as a language. Others will tell you that the RPG cycle is irrelevant in a modern programming language. The fact is, once you get out of college, unless you work for the most rigid and the most precise shops, in your role as a programmer, the only thing that matters is that you can read code and that you can fix code. Being able to write code from scratch, though a valuable skill, is just not as important. .

Once you begin a career in RPG programming, you will find code written in each and all of the techniques available in a computer programming language. For RPG programmers, this includes the RPG fixed logic cycle, RPG II type code, RPG III style, as well as RPG IV. Additionally, ther are externally described and internally described variants. Though many would tell the neophyte to stay clear of the cycle, it not only has its place in new development, but it also is an extremely valuable skill in being able to decode the many applications that once in a while come up for air and need to be fixed or adjusted to meet a new need.

So, we begin our description of the coding necessary to deal with the specs of our payroll sample by decoding the program from scratch.

Check the page count. I have almost hit page 50 and I have not shown one line of RPG code. I'm setting you up, of course for learning RPG by learning a little than by being introduced to a lot at a time. By now, you have the flavor of early RPG and you have a notion that it may be very good to solve business problems that you have not even begun to discuss.

So, before we move into the mundane of describing the program solution to Chapter 3, let's consider that all RPG coders who use any form of the language believe they are RPG coders. That means that you do not have to be at the bleeding edge of RPG IV to be serving your company well.

In this book, of course, we assume nothing. Many original RPG programs exist (cycle and otherwise) in the code inventories and packages within long-standing RPG shops today. The inventories include cycle programs that were coded in RPG, RPG II, RPG III or RPG IV. Since it is extremely important to be able to update such code, RPG ***"decoding"*** of cycle programs and other older RPG forms is a necessary skill for an

RPG programmer. Therefore, in this book, to help you become a guru in RPG decoding techniques, we show several methods of achieving the same end in the programming examples.

Well Not Exactly!

Once you learn RPG itself via RPG/400, it will be easy to relate to the same facilities as delivered within RPGIV. This book will show you how to do that all along the way to learning RPG. For the RPGIV specific advanced language features, since they are almost as different as COBOL is different from RPG, we have included several chapter that addresses the new facilities and we show specific coding examples for RPGIV.

However, in its basic form, RPGIV and RPG/400 are very similar. Therefore, from a teaching/learning perspective, this book focuses on RPG/400 in our early examples and then show the same code in RPGIV. When it is appropriate to explain a feature of RPGIV that is substantially different from the RPG/400 model, we will take the time in place to make the explanation.

RPG is a wonderful language and you will soon see why. The fact is that Java and C programmers and even old-time procedural COBOL programmers have not typically liked RPG. Likewise, RPG purists of all genres do not like Java, C, or COBOL. Old-time RPG programmers do not have that warm of a feeling either for free format RPG and those features that make the language look more like Java and C. And, as would be expected, those who have adopted the RPG IV language as their first language or who like the types of constructs that have been added to RPG IV that make the language more universally appealing are in conflict with the old-line RPG crowd. Something less than half of the RPG programmers out there have wholesale adopted RPGIV and a lesser percentage use the newest bells and whistles.

Rather than suggest that one faction in the RPG fight is correct and another is off-base, since decoding for maintenance purposes is an absolute necessity, we will start with the basics of all RPG and move on from there. It would be inappropriate to teach just free-format RPG with the built-in block language structures since the code libraries that you will find in practice do not have much of this. So, we have chosen to help you learn to be a real RPG programmer first. You can then stretch those skills

to be an advanced RPGIV programmer after you gain knowledge of RPG as practiced in most IT shops.

Figure 5-1 RPG Cycle Program PAREG with State Totals and Matching Records

```
        *************** Beginning of data ***********************
        6789112345678921234567893123456789412345678951234567896123456 78
0001.00 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
0002.00 H
0003.00 F*
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FEMPMAST IPEAE                    DISK
0007.00 FTIMCRD  ISEAE                    DISK
0008.00 FQPRINT  O   F      77     OF     PRINTER
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IPAYR        01
0013.00 I              EMPNO                           EMPNO   M1
0014.00 I                                              EMPCTYL1
0015.00 I              EMPSTA                          EMPSTAL2
0016.00 ITIMR        02
0017.00 I              EMPNO                           EMPNO   M1
0018.00 C*
0019.00 C* RPG CALCULATION SPECIFICATION FORMS
0020.00 C*
0021.00 C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72
0022.00 C   02 MR   EMPPAY    ADD  CTYPAY    CTYPAY  92
0023.00 CL1         CTYPAY    ADD  STAPAY    STAPAY  92
0024.00 CL2         STAPAY    ADD  TOTPAY    TOTPAY  92
0025.00 O*
0026.00 O* RPG OUTPUT SPECIFICATION FORMS
0027.00 O*
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0030.00 O                                   32 'THE DOWALLOBY COMPANY'
0031.00 O                                   55 'GROSS PAY REGISTER BY '
0032.00 O                                   60 'STATE'
0033.00 O                         UDATE Y   77
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0036.00 O                                    4 'ST'
0037.00 O                                   13 'CITY'
0038.00 O                                   27 'EMP#'
0039.00 O                                   45 'EMPLOYEE NAME'
0040.00 O                                   57 'RATE'
0041.00 O                                   67 'HOURS'
0042.00 O                                   77 'CHECK'
0043.00 O        D 1      02NMR
0044.00 O                                   46 'NO MATCHING MASTER'
0045.00 O                         EMPNO     27
0046.00 O                         EMPHRS1   67
0047.00 O        D 1      02 MR
0048.00 O                         EMPSTA     4
0049.00 O                         EMPCTY    29
0050.00 O                         EMPNO     27
0051.00 O                         EMPNAM    52
0052.00 O                         EMPRAT1   57
0053.00 O                         EMPHRS1   67
0054.00 O                         EMPPAY1   77
0055.00 O        T 22     L1
0056.00 O                                   51 'TOTAL CITY  PAY FOR'
0057.00 O                         EMPCTY    72
0058.00 O                         CTYPAY1B  77
0059.00 O        T 02     L2
0060.00 O                                   51 'TOTAL STATE PAY FOR'
0061.00 O                         EMPSTA    54
0062.00 O                         STAPAY1B  77
0063.00 O        T 2      LR
0064.00 O                         TOTPAY1   77
0065.00 O                                   50 'FINAL TOTAL PAY'
        ***************** End of data ****************************
```

Figure 5-2 RPG Cycle Program PAREG – Internally Described Data

```
0001.00 F* RPG HEADER SPECIFICATION FORMS
0002.00 H
0003.00 F*
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FEMPMAST IPEAF      55            DISK
0007.00 FTIMCRD  ISEAF       7            DISK
0008.00 FQPRINT  O   F      77     OF     PRINTER
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IEMPMAST AA  01
0013.00 I                                         1   30EMPNO   M1
0013.01 I                                         4  23 EMPNAM
0013.02 I                                        24  282EMPRAT
0014.00 I                                        29  48 EMPCTYL1
0015.00 I                                        49  50 EMPSTAL2
0015.01 I                                        51  550EMPZIP
0016.00 ITIMCRD  AB  02
0017.00 I                                         1   30EMPNO   M1
0017.01 I                                         4   72EMPHRS
0018.00 C*
0019.00 C* RPG CALCULATION SPECIFICATION FORMS
0020.00 C*
0021.00 C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72
0022.00 C   02 MR   EMPPAY    ADD  CTYPAY    CTYPAY  92
0023.00 CL1         CTYPAY    ADD  STAPAY    STAPAY  92
0024.00 CL2         STAPAY    ADD  TOTPAY    TOTPAY  92
0025.00 O*
0026.00 O* RPG OUTPUT SPECIFICATION FORMS
0027.00 O*
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0030.00 O                                    32 'THE DOWALLOBY COMPANY'
0031.00 O                                    55 'GROSS PAY REGISTER BY '
0032.00 O                                    60 'STATE'
0033.00 O                         UDATE Y    77
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0036.00 O                                     4 'ST'
0037.00 O                                    13 'CITY'
0038.00 O                                    27 'EMP#'
0039.00 O                                    45 'EMPLOYEE NAME'
0040.00 O                                    57 'RATE'
0041.00 O                                    67 'HOURS'
0042.00 O                                    77 'CHECK'
0043.00 O        D 1      02NMR
0044.00 O                                    46 'NO MATCHING MASTER'
0045.00 O                         EMPNO      27
0046.00 O                         EMPHRS1    67
0047.00 O        D 1      02 MR
0048.00 O                         EMPSTA      4
0049.00 O                         EMPCTY     29
0050.00 O                         EMPNO      27
0051.00 O                         EMPNAM     52
0052.00 O                         EMPRAT1    57
0053.00 O                         EMPHRS1    67
0054.00 O                         EMPPAY1    77
0055.00 O        T 22     L1
0056.00 O                                    51 'TOTAL CITY  PAY FOR'
0057.00 O                         EMPCTY     72
0058.00 O                         CTYPAY1B   77
0059.00 O        T 02     L2
0060.00 O                                    51 'TOTAL STATE PAY FOR'
0061.00 O                         EMPSTA     54
0062.00 O                         STAPAY1B   77
0063.00 O        T 2      LR
0064.00 O                         TOTPAY1    77
0065.00 O                                    50 'FINAL TOTAL PAY'
```

Decoding the PAYREG RPG Program

The PAREG program that we will be decoding in this chapter and the next several chanters is chapter is shown first in Figure 5-1. The RPGIV version of the same program as converted using IBM's CVTRPGSRC facility is shown in Figure 7-1.

Still, a third version of this program is necessary in order to cover the notion of externally described files and internally described files in RPG. Figure 5-1 shows the program as it would be written today using externally described data. Figure 5-2 shows the same program using internally described data, which is often called program described data.

You will notice that the only difference between the two programs is in the Input (I) specifications. Program described data use two coding formats for input known as the I and the J. The I part captures the record identification information. The J part captures the input field information. For externally described files, the formats are called IX and JX respectively. The X designation is for eXternal. Overall there is not much difference between the I and IX and the J and JX formats.

Internally & Externally Described Data

Internally described RPG data means that the data is described within the program. The field names and attributes are assigned within the program. The length is specified within the program. That is the big difference functionally between external and internal descriptions. When a program is compiled that uses internal descriptions, the compiler gets the exact record layout from the program itself. When a program is compiled that uses external descriptions, the compiler fetches the data descriptions from the database and manufactures the input and output specifications in the compiled program to save the programmer the work of all that coding.

Besides the input area, there are also changes in the PAREG program in the File Description area. Column 19 of the File Description specification asks if the input and output descriptions from the file should be fetched from the data base (coded as E) or whether they should use the fixed format within the program (coded as F). Based on this switch the compiler knows how to do its job. Additionally, since using the "F"

switch means the compiler does not visit the external file description to get the data definitions, it has no means of knowing the record length. So, you can see by examining Figure 5-1 and Figure 5-2 that the PAREG program in Figure 5-2, which uses internally described data definitions has a record length of 55 specified for the EMPMAST file and a record length of 7 for the TIMCRD file. This coding is unnecessary for the File when using external descriptions.

Let's begin our decoding by describing the form types used in the all RPG programs including the PAREG program.

Looking at the code in Figure 5-1, you can see that there are numbers in columns 1-7, then there is a space, and then there is a letter. Walking down the program line by line, in the column that holds the letter, you can see that there are 2 H's, 6 F's, 9 I's, 7 C's, and 41 O's. The program described version in Figure 5-2 uses the same sequence numbers with suffixes to distinguish new lines. It has 13 input lines. When present, the forms are always presented to the RPG compiler in sequence.

The names for these specifications are as follows:

H Header (Control) Specification Form
F File Description Specification Form
I Input Specification Form
C Calculation Specification Form
O Output Specification Form

Two other RPG/400 forms not used in this program are as follows:

E File Extension Specification Form (after F)
L Line Counter Specification (after E or F if no E)

The Extension form is used to describe arrays and tables and the Line Counter form is used to specify the length and overflow lines of forms suing special sized (not 8.5 X 11) paper. The Extension form is covered in Chapter **** and the Line Counter is covered in Chapter 6.

The vision of the programs in Figure 5-1 and 5-2 is deceiving in that the column shown that has the specification type is presented as column 8 in

the figure. However, this is not the case. To create the figure, I used the Source Entry Utility, described in Appendix ****. I positioned column 6 using a Window command so that it would be the first column of the program shown. SEU provides the numbers on the left side as a guide for editing. Those numbers do not exist in the RPG statement. So, even though it looks like column 8, please make the mental adjustment so that just as the format line above shows, the RPG form ID is in column 6.

Columns 1 through 5 of all RPG statements are no longer used. In the past when each line in a program was typed on a punch card, it was important to number the cards in case they fell on the floor and needed to be machine sorted back in sequence. RPG/400 defines 80 columns of each form for your use. Since we do not have to deal with the first five columns at all, you can see we are making terrific progress. All of the form types that we are about to study below are typed in column 6 of the forms in Figure 5-1.

H-- Header (Control) Specification Form

Only one control specification is permitted in a program. Yet, as shown in statement 1 and statement 2 of the PAREG program, we clearly have two H forms in this program. Whenever an * is placed in column 7 of any RPG form type, the source line becomes a comment line. It does not matter whether column 6 contains a valid form type. Whenever there is an asterisk (*) in column 7 the line is a comment line. Thus, in our example there are not two H specifications. The first one is a comment line in which we have placed a comment. The second is the control statement which has no entries in this case.

The control (Header – H) specification provides the RPG/400 compiler with information about your program and your system. This includes the following information:

- ✓ Name of the program
- ✓ Date format used in the program
- ✓ Alternative collating sequence or file translation if used
- ✓ Debug Mode (1 in column 15)

.

The control (H) specification is optional and thus in program *PAYREG*, since it has no entries, it was not necessary to specify it at all..

The detailed format of the RPG/400 Header specification is as follows:

H Columns 7-14 (Reserved)

H Column 15 (Debug)

Place a 1 in column 15 to turn on Debug.

H Columns 16-17 (Reserved)

H Column 18 (Currency Symbol)

Any character except zero (0), asterisk (*), comma (,), period (.), ampersand (&), minus sign (-), the letter C, or the letter R may be specified as the currency symbol.

H Column 19 (Date Format)

Specify the format of the RPG/400 user dates. The allowable entries are as follows:

Blank Defaults to month/day/year if position 21 is blank. Defaults to day/month/year if position 21 contains a D, I, or J.
M Month/day/year.
D Day/month/year.
Y Year/month/day.

H Column 20 (Date Edit)

The entry in this position specifies the separator character to be used with the Y (date) edit code – typically slash (/), period (.), or dash (-).

H Column 21 (Decimal Notation)

Specify the notation to be used for the user date. This entry also specifies the decimal notation and the separator used for numeric literals and edit codes. The term decimal notation refers to the character that separates whole numbers from decimal fractions.

H Columns 22-25 (Reserved)

H Column 26 (Alternate Collating Sequence)

The allowable entries are as follows:

Blank	Normal collating sequence is used.
S	Alternate collating sequence is used.

H Columns 27-39 (Reserved)

H Column 40 (Sign Handling)

A blank entry is required. The sign is always forced on input and output of zoned numeric fields.

H Column 41 (Forms Alignment)

Should forms be aligned on first page printing? The allowable entries are as follows:

Blank First line is printed only once.
1 First line can be printed repeatedly allowing the operator to adjust the printer forms.

If the program contains more than one printer file, the entry in position 41 applies to each printer file that has 1P (first-page) output. This function may also be specified by the CL command OVRPRTF (Override to Print File) or in the printer device file and can be affected by the ALIGN option on the STRPRTWTR command. Use column 41 only when the first output line is written to a printer file.

H Column 42 (Reserved)

H Column 43 (File Translation)

A blank entry says no translation occurs. An entry of F indicates that a file translation table is to be used to translate all data in specified files.

H Columns 44-56 (Reserved)

H Column 57 (Transparency Check)

Sometimes data composition messes up the best coding. There are few instances but they exist in which transparency of characters needs to be checked. The allowable entries are as follws:

Blank No check for transparency.
1 Check for transparency.

If you specify 1 in position 57 of the control specification, the RPG/400 compiler scans literals and constants for DBCS characters. It does not check hexadecimal literals.

H Columns 58-74 (Reserved)

H Columns 75-80 (Program Identification)

Considering that most RPG programs are written today with No H specification whatsoever, the typical way of naming an RPG program is the default of taking the name of the source member and making it the name of the program object.

The symbolic name entered in these positions is the name of the program that can be run after compilation. You can override this name with the PGM parameter in the CL command CRTRPGPGM (Create RPG Program) which is used to create an object from your source RPG program.

If you do not specify a name in positions 75 through 80 of the control specification or on the CRTRPGPGM command, but the source is from a database file, the member name is used as the program name. If the source is not from a database file, the program name defaults to RPGOBJ.

If you specify the program name on the control specification, its maximum length is 6 characters. If you specify the program name in the CRTRPGPGM command, its maximum length is 10 characters.

RPGIV Header (H) Specification

The Header specification is also called the CONTROL specification. It is seldom used in RPG/400 programs written today. Just as in RPG, the H spec provides information about generating and running programs. However, with RPGIV, there are three different ways in which this

information can be provided to the compiler and the compiler searches for this information in the following order:

1. A control specification included in your source with an H in column 6
2. A data area named RPGLEHSPEC in *LIBL
3. A data area named DFTLEHSPEC in QRPGLE

Once one of these sources is found, the values are assigned and keywords that are not specified are assigned their default values.

Header option 1 is similar to RPGIV in that you specify the options in the program. With option 2, you get to include a prewritten H spec called RPGLEHSPEC in a data area object in your program merely by permitting the compiler to find the object in your library list. Header option 3 is similar to option 2 and option 1 with nothing specified. In other words, RPG finds the default H spec data area object that it stores under the name of DFTLEHSPEC in the QRPGLE library from which the compiler itself gets launched.

The fact that there is an option 1 is where the similarities end between the standard H spec and the new RPGIV H spec.

As an introductory book to RPG, it is not the author's intention to provide all of the information that exists in IBM manuals on a given topical area. Some of the keywords are self explanatory and thus will be helpful immediately. However, some of them are not self explanatory and demand that the reader be well versed in other topics that are not given extensive treatment in this book. For example, there are a number fo keywords that require a high level of understanding of the Integrated language Environment as well as the notion of procedures in RPGIV. Though these topics are touched on lightly in this book, we do not supply enough information for a beginner to have a command of these features and thus the keywords that may be used to invoke certain features and options. The good news as with most of RPG is that the defaults do a fine job of covering the options that a beginner would need to get a quick start in programming RPGIV.

The format of the new RPGIV H spec is very simple"

Column 6	H
Columns 7-80	Area for header spec keywords
Columns 81 – 100	Comments

The RPGIV Header spec is keyword driven in much the same way as other System I artifacts, such as physical files, logical files, and display files are specified to their respective compilers.

The keyword format is simply as follows:

`Keyword(value)`

A list of the RPGIV keywords and sample values is shown below:

Figure 5-2 shows each of the RPGIV keywords that can be used in the H specification with a brief explanation. The table also shows one option specified as a sample and other options in a separate column..

Figure 5-2 Header Keywords and Options

Keyword / Sample	Other Options	Meaning
ACTGRP(*NEW)	*CALLER; 'activation-group-name'	Type of activation group
ALTSEQ{*NONE)	*SRC; *EXT	Alternate sequence?
ALWNULL(*NO)	*INPUTONLY; *USRCTL)	Nulls allowed?
AUT(*LIBRCRTAUT)	*ALL; *CHANGE; *USE; *EXCLUDE	Authority given to users for use of program
BNDDIR('BINDER')	:'binding-directory-name'...	List of binding directories
CCSID(*GRAPH: Ignore)	*SRC; number	Set default graphic
CCSID(*UCS2: 13488)	Number	Set default UCS-2 CCSID
COPYNEST(32)	Number	Maximum nesting depth for COPY directives (1 – 2048)
COPYRIGHT('Kelly Consulting')	'copyright string'	Set copyright info

CVTOPT(*DATETIME)	***{NO}DATETIME *{NO}GRAPHIC *{NO}VARCHAR *{NO}VARGRAPHIC**	**How compiler handles dates, etc.**
DATEDIT(*DMY)	***MDY, or *YMD**	**Format for Y edit code**
DATFMT(*ISO)	**Valid date formats**	**Internal date format**
DEBUG{*YES)	***NO**	**Debug on or off**
DECEDIT(*JOBRUN)	**Value**	**Char used for dec. pt.**
DECPREC(30)	**31**	**Decimal precision**
DFTACTGRP(*YES)	***NO**	**Default activation**
DFTNAME(MYPROG)	**RPG Name**	**Specify pgm name**
ENBPFRCOL(*PEP)	**ENTRYEXIT; *FULL**	**Is full performance collection enabled?**
EXPROPTS (*MAXDIGITS)	***RESDECPOS**	**Precision rules**
EXTBININT{(*NO)	***YES**	**Internal or external binary format is used.**
FIXNBR(*ZONED)	***{NO}ZONED; *{NO}INPUTPACKED)**	**Should decimal data errors be auto-fixed?**
FLTDIV{(*NO)	***YES**	**Use floating divide**
FORMSALIGN{(*NO)	***YES**	**Repeat 1p output for forms alignment**
FTRANS{(*NONE)	***SRC**	**File translation?**
GENLVL(10)	**Number 1 – 20**	**Errors > this value will stop compile**
INDENT(*NONE)	**character value**	**Should structured operations be indented?**
INTPREC(10)	**20 (10 or 20)**	**Intermediate precision**
LANGID(*JOBRUN)	***JOB; language-identifier**	**Language ID**
NOMAIN	**For module**	**No main proc**
OPENOPT (*NOINZOFL)	***INZOFL**	**Set OF indicator to off when file opened**
OPTIMIZE(*NONE)	***BASIC | *FULL)**	**Level of optimization**
OPTION(*XREF)	***{NO}XREF; NO}GEN; *{NO}SECLVL; *{NO}SHOWCPY; *{NO}EXPDDS; *{NO}EXT; *{NO}SHOWSKP); *{NO}SRCSTMT); *{NO}DEBUGIO)**	**Specifies compiler options**
PRFDTA(*NOCOL)	***COL**	**Collect profile data?**
SRTSEQ(*HEX)	***JOB; *JOBRUN; *LANGIDUNQ; *LANGIDSHR; 'sort-table-name'**	**Sort sequence table**
TEXT(*SRCMBRTXT	***BLANK; 'description'**	**Descriptive text**
THREAD(*SERIALIZE)	**Not specified**	**Serialize thread?**
TIMFMT(*ISO)	**Time format options**	**Time format**
TRUNCNBR(*YES)	***NO**	**Use truncated value or**

USRPRF(*USER)	***OWNER**	**produce an errormsg. Which authority?**

Chapter 6

The Specifics of RPG Coding –File Description & Line Counter Specifications – by Example

Talking to the Outside World

As in all programs that use files, the language must provide a means of linking from the program to the outside world. The File Description Specification, a.k.a. the "F" spec is the way this communication is achieved in RPG.

F-- File Description Specification Form

We continue our decoding adventure for File Desctiptions and Line Counters in this Chapter. A sample filled-in RPG/400 File description coding sheet is shown in Figure 6-1. Column 66 of the line in which the CUSMSTF file is defined has an A in it. For viewing purposes we moved that column two columns away from the end of the form so it could be seen.

Figure 6-1 Sample File Description form (amalgamated & truncated)

File Description Specifications

RPG Program Using a Physical, Display, and Printer File

```
0 2 0F*
0 3 0F* SAMPLE RPG PROGRAM
0 4 0F*
0 5 0FCUSMSTF UF E K DISK A
0 6 0FCUSDSPF CF E WORKSTN
0 7 0FCUSPRTF O E 10 PRINTER
```

*** Note – The A in column 66 next to "DISK" has been moved for viewing.**

Before SEU (See Appendix A ****), programmers painstakeingly coded their RPG programs on coding sheets that were 8 ½ X 11" in size. Figure 5-3 reflects a hand-written coding sheet.

Notice in the area marked as 2 that column 19 contains an E for all of these files. That means that the files are externally described and no record length is required on the File Description statement for these three files. Notice that there is no primary and no secondary file defined. That means that this program does not depend on the RPG cycle.

The customer master file CUSMSTF and the customer display file CUSDSPF each use an F designator in column 16 rather than the P or S that we have used in the PAREG program. This means that the file is fully procedural. Fully procedural (F) files can be processed by the many RPG operations such as read (READ), read equal (READE), chain (CHAIN) etc. that we will be covering in Chapter *****. The fully procedural file gives the programmer great latitude in the calculations specification to determine which operations occur at what times.

In the PAREG program, column 15 was coded with either an I for input or an O for output. The sample lines for the first two files in Figure 6-1 are coded with a U and a C respectively. The U means that this file can be used for input and that the records read will be locked for update. The

program can then update the data in the record and with an update operation write the changed data back to disk. The record gets unlocked when the update is complete or when the program reads another record in the same file.

The C (combined) designation is very similar to the U (update) operation but it is reserved for device files such as terminals. For the program to be able to write the screen and then read the data back, the display file must be coded with a C in column 15 to designate that combined operations of output and input are available.

Notice on the right of the form that the device type is coded as WORKSTN. PAREG used just the DISK device and PRINTER device. So now you know how to define a simple workstation device in your programs. Workstation (WORKSTN) files can be designated as input (I) or output (O) also but most programmers prefer to use the C designator regardless of the type of operations that they will use against the file.

The CUSTMSTF file also has a K designation in column 31. This is to tell the compiler that the file will be processed by a key, such as customer number. It also says that the file has an index associated with it. The file can be a keyed logical or physical file. Looking out on the right to column 66, you will notice an A designation. This tells the RPG compiler that this file can not only be used for input and update operations but it can be used for WRITE operations which add records to the file.

That's a lot to know about file description specifications and it should give you a head start in decoding and understanding the File description specifications used in the PAREG example.

Why File Descriptions?

File description specifications describe all the files that your program uses. The information for each file includes the following:

- ✓ Name of the file
- ✓ How the file is used
- ✓ Size of records in the file

- ✓ Input or output device used for the file
- ✓ If the file is conditioned by an external indicator.

For your convenience we have collected the six "F" specs from the external and internal versions of PAREG and have listed them again in Figure 6-2 and 6-3 respectively. It may help to recall that the only difference between the external and internal versions in this program is the record length and the F instead of an E in column 19.

Figure 6-2 PAREG RPG/400 File Descriptions

```
        678911234567892123456789312345678941234567895123
0003.00 F*
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FEMPMAST IPEAE                   DISK
0007.00 FTIMCRD  ISEAE                   DISK
0008.00 FQPRINT  O   F      77     OF    PRINTER
```

Figure 6-3 PAREG – File Descriptions Internally Described Data

```
        678911234567892123456789312345678941234567895123
0003.00 F*
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FPAYMAST IPEAF      55           DISK
0007.00 FEMPTIM  ISEAF       7           DISK
0008.00 FQPRINT  O   F      77     OF    PRINTER
```

Statements 3 through 5 contain asterisks in column 7, therefore, they are comments used for documenting the program. Statements 6 through 8 are very important to the functioning of this program as they define both the two input files and the printer file.

F Column 7-14 File Name

Columns 7-14 of the F spec are where you make the link to the outside world in RPG. Every file must have a unique file name that is defined to the i5/OS system. The file name can be from 1 to 8 characters long, and must begin with an alphabetic character. As you can see in Figures 6-2 and 6-3, the three file names we define in the PAREG program are EMPMAST, TIMCRD, and QPRINT.

Using the DDS that we described in Figures 3-2 and 3-3, we created two database files named EMPMAST and TIMCRD respectively. See Appendix **** for the instructions on how to create a database file from DDS. In statement 8, we see the name QPRINT which is the name of a reserved printer file in i5/OS that IBM makes available for your use in RPG. So, each time you create a program that prints a report, you may use the print file QPRINT to permit this to happen. You may also create your own print file. This is shown in Appendix ****. When you create your own print file, you would use the name of that file in place of QPRINT.

In RPGIV, the file name can be ten characters and it occupies columns 7 to 16.

F Column 15 File Type – I, O, Etc.

Column 15 is where you define the file type. The question you ask in order to know how to fill in this column is: "How will I use this file – input or output?" The choices of entries for Column 15 are as follows:

I Input file.
O Output file.
U Update file.
C Combined (input/output) file.

In our example program, we defined EMPMAST and TIMCRD as I for input. These two database files will provide the input for our program. We defined QPRINT as O for output since we will produce an output report from the input data read from the database files. If we were going

to write back records to the EMPMAST or TIMCRD databases, we would have coded it as U for update. We have no use for the C for combined in this program. Later when we work with display files in which we write a program to interact with a terminal user, we will use the C designation to indicate that we both write to the display and read from the display using the same file description specification.

In RPGIV, the File Type is specified in position 17.

F Column 16 File Designation – Primary, Secondary, etc.

Column 16 is used to designate that a file will use the RPG cycle or not and it is used to provide a means for special files to be loaded into the system to control processing. It also provides for files to be processed outside the RPG cycle

The entries that can be used in column 16 and their meaning are presented below:

Blank	Output file
P	Primary file
S	Secondary file
R	Record address file
T	Array or table file
F	Full procedural file

Blank, P, S-- Output, Primary, and Secondary

The QPRINT file defined in the File Description specifications of the PAREG RPG program is output-only and thus, we describe it in column 16 with a blank entry. The EMPMAST file is designated as primary. This means that it will be the first file read and it will be the first file read when there is a matching time card record (secondary.) The TIMCRD file is designated as secondary since it is not read until ***all*** of the matching primaries are read for an employee. Since in our example, there will be

just one payroll master for each time card, the actuality is that that after a matched master, the matching time card record, designated as secondary will be read. More than one secondary file can be specified but this is not necessary in our example.

Though the PAREG program does not include any other entry types, the possible choices are explained below for completeness.

R-- Record Address File

A record-address file (RA) is a sequentially organized file used to select records from another file. Only one file in a program can be specified as a record-address file. One of the frequent uses for an RA file is to process the results of what is called and address sort. In other words, you can run the FMTDTA AS/400 command to sort a file and instead of a big file with big record lengths as your output, you can ask the sort to create a file of record addresses. If your program were working with he output of such a sort, you would designate it with an R in column 16 of the F specification. In this book, since logical files have mostly replaced the need for RA files in RPG, this book does not offer any examples of using RA files.

T-- Array or Table File

Array and table files are specified by placing a T in position 16. Arrays and tables are small files, often of codes that enable the code to be checked for existence in some cases or to find a matching explanation for a code. A table for code lookups might be used for payroll codes as an example. The code M in the table might have a matching explanation called Married and the code F might have a matching explanation called Female. RPG provides a means of compiling these codes at the back of the program and this is very convenient when the codes are not expected to change frequently.

For codes that change frequently, such as the tax tables in a payroll program, it is not wise to compile these with the program since each time the government makes a change, you must alter the program contents and you must recompile (retranslate into machine language) the program. For situations such as this, RPG provides this nice facility in which the table

or array can be pulled in from disk right before the program is executed. In this way, the changes can be made to the table or array on disk without the program having to be modified or recompiled.

F-- Full Procedural File

As the name implies, full procedural files are those that give the programmer full procedural control of happenings in a program. This is a very important entry and will be explored further in this book in far greater detail. This entry is used when the input, output and update functions are controlled by calculation operations. File operation codes such as CHAIN or READ or UPDAT or WRITE are used to do input, output, or input/output functions.

In RPGIV, the File Designation is specified in position 18.

F Column 17 (End of File)

This entry is used with primary and secondary files to instruct the program about how it can end naturally. For RPG programs to end, the LR indicator must be turned on. When just one file is used for input, the entries are moot since LR will naturally be turned on when all records from the file have been read. If a primary and a secondary file were defined for input such as in our example, we would not want the LR indicator to turn on and end the program after all the primaries are read because we may still have a matching time card record to be processed.

The entries for this column are as follows:

E All records from the file must be processed before end

Blank This file does not have to be fully processed to end

However, if position 17 is blank for all files, then RPG defaults that all records from all files must be processed before end of program (LR) can occur. If position 17 is not blank for all files, all records from this file may or may not be processed before end of program occurs in mult-ifile

processing. In PAREG, both input files must finish being read before the program can end.

In RPGIV, the End of File is specified in position 19. The Add records column of RPG/400 (column 66) is specified in RPGIV in position 20.

F Column 18 (Sequence)

You use this handy facility to sequence check your input, combined our update files when they are being read as primary and/or secondary files.

The possible entries for this column are as follows:

A	Match fields are in ascending sequence.
Blank	Same as A
D	Match fields are in descending sequence.

This column works hand in glove with the match field indication on the Input Specification form that is used to assure a match between two files such as the EMPMAST and TIMCRD files. Both input files defined in PAREG in Figure 6-2 have an A designation meaning that the match fields (EMPNO) must be in ascending sequence. Jumping ahead just a little, to show you exactly what I mean by match fields, I have duplicated the Input lines in Figure 6-4 for your reading convenience:

Figure 6-4 PAREG Match Field Input Specifications

```
        67891123456789212345678931234567894123456789512345678961234567 8
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IPAYR        01
0013.00 I              EMPNO                            EMPNO   M1
0014.00 I                                               EMPCTYL1
0015.00 I              EMPSTA                           EMPSTAL2
0016.00 ITIMR        02
0017.00 I              EMPNO                            EMPNO   M1
```

On line 12, the record format name PAYR from the primary file called EMPMAST is specified. Following this on line 13 the EMPNO field from EMPMAST is referenced. The M1 in column 61 next to the EMPNO

tells the compiler that this field is a match field and it tells the compiler that if there is an “A” or “D” entry in column 18 of the File description, the file EMPMAST must be in sequence – either ascending or descending. In PAREG of course it is ascending. If the file records as they are read by the program are not in sequence, the RPG program will halt with an error condition. The M1 in line 17 does the same thing for the TIMCRD file.

In RPGIV, the Sequence works the same but is specified in position 21.

F Column 19 (File Format)

This column is used to tell the compiler from what source the input and output will come for this particular file in the program. The input / output definitions may come from within the program on input and output specification sheets. These files formats are referred to as program described or internally described files. The definitions may also come externally from within a database, workstation, printer or other file object that is referenced within in the program. These file formats are referred to as externally described files.

The possible entries for File Format are as follows:

F Program described file
E Externally described file

An F in position 19 indicates that the records for the file are described within the RPG/400 program on input/output specifications. An E in position 19 indicates that the record descriptions for the file are external to the RPG/400 source program. The compiler obtains these descriptions when the program is compiled and it then includes them in the source program and in the accompanying source listing.

As you can see in line 8 of PAREG, the QPRINT print file is described with an “F” entry in column 19. This means that the record length will be provided by file description and the format of the output records will be provided within the program as you can see in lines 25 through 65 of Figure 5-1. .

As you can also see in lines 6 and 7 of the external version of the program, Figure 6-2, both the EMPMAST and the TIMCRD files have an "E" entry in file description column 19 and thus they are externally described. The internally described PAREG shown in Figure 6-3 has an F in column 19 for internal fixed format. The external definitions will be brought into the program at compilation time. A snapshot of a compile listing showing the full inclusion of the data definitions brought in from the external database file defined in the RPG program is shown in Figure 6-5.

Figure 6-5 Compile Listing of External Input Expanded

```
   1000  I* RPG INPUT SPECIFICATION FORMS
   1100  I*
   1200  IPAYR        01
   1300  I              EMPNO                          EMPNO   M1
   1400  I                                             EMPCTYL1
   1500  I              EMPSTA                         EMPSTAL2
   1500   INPUT  FIELDS FOR RECORD PAYR FILE EMPMAST FORMAT PAYR.
A000001                 EMPNO                     1    30EMPNO   M1
A000002                                           4   23 EMPNAM
A000003                                          24   282EMPRAT
A000004                                          29   48 EMPCTYL1
A000005                 EMPSTA                   49   50 EMPSTAL2
A000006                                          51   550EMPZIP
   1600  ITIMR        02
   1700  I              EMPNO                          EMPNO   M1
   1700   INPUT  FIELDS FOR RECORD TIMR FILE TIMCRD FORMAT TIMR.
B000001                 EMPNO                     1    30EMPNO   M1
B000002                                           4    72EMPHRS
```

The sequence designators in Figure 6-5 that begin with A are the specifications that were brought in from the EMPMAST file from record format name PAYR. The sequence designators that begin with B were brought in from the TIMR record format from the TIMCRD file.

As a point of note, when external files are used with fully procedural files, no input specifications need to be presented to the RPG program at all so this can be a great saving in coding. However, when a program uses level breaks or match fields as the PAREG program does, the program uses the input form in order to tell the compiler what the match fields are and what fields should be used for various level totals. Notice in Figures 5-1 and 5-4 that just the lines of input with the M1, L1, and L2 designated input fields needed to be specified. The other input field definitions were brought in from the external file definitions.

In RPGIV, the File Format is specified in position 22.

F Columns 20-23 (Reserved)

As you can see in the program, there are no entries in positions 20 through 23. In fact, these must be blank. In older RPG's these positions were available to specify a block length. Since the database itself is used to determine blocking during execution, there is no need for this in RPG. When a block length is allowed, for example, there are parameters in the CL command OVRDBF that permit a blocking factor to be specified. Our simple program, PAREG uses not such facility.

F Columns 24-27 (Record Length)

Programmers use positions 24 through 27 to indicate the length of the logical records contained in a program-described file. The maximum record size that can be specified is 9999; however, record-size constraints of any device may override this value.

Notice that the QPRINT file has a 77 as its record length. That is because the longest print line ends printing in column 77. The EMPMAST and the TIMCRD files are both externally described in Figure 6-2, therefore, their record lengths are provided by the external object and this entry must be blank for externally described files. For the internally described version, the record length for EMPMAST is specified as 55 and for TIMCRD as 7.

In RPGIV, the record length can be longer by one column and is specified in positions 23 to 27.

F Columns 28-39 (Other Entries)

Positions 28-39 are not needed for the PAREG program. We show the other entries here, however, so that this book can also be used for you as a reminder guide and thus being complete at this point of the book is important. We will revisit File Descriptions in later chapters when this information is germane to a problem example we are solving.

F Column 28 Limits Processing

The entries for column 28 are as follows:

L Sequential-within-limits processing by a record-address file
Blank Sequential or random processing

You specify an L in column 28 to indicate limits processing otherwise let the column blank. The default then is no limits processing. The limits file is a file of limits in which each record contains a set of limits that consists of the lowest record key and the highest record key from that particular section of the indexed file to be processed. You may have multiple limits records in the file. When a limits record is read, the corresponding record with the key value in the file is processed.

The record address file with the limits record would require its own File Description statement. The L in 28 is for the keyed file that will be processed via the limits file.

In RPGIV, Limits processing is also specified in position 28.

F Column 29-30 Length of Key or Record Address

The entries specified in columns 29 to 30 pertain to program described keyed (indexed) database files. If you are not coding a program described file or the file is not keyed, then leave these positions blank. Otherwise, place the total length of the key field(s) in 29-30.

In RPGIV, the length of key or RAF is specified in position 29 – 33..

F Column 31 Record Address File Type

A record address file is a file used to process another file. See limits file explanation for column 28 above. Whereas the file that would have an L specified in column 28 is the keyed master file, the entries in column 31 describe the RAF file that will control the processing of that file.

The FMTDTA command (a.k.a. the System I sort) has the ability to produce a record address file of relative record numbers (also called an address-out or ADDROUT file or record address file [RAF]). ADDROUT sorts were very prevalent when disk was very expensive. Instead of a fully sorted file that included all of the records in the file that were part of the sort output, the ADDROUT file contained the addresses of the records such that if the records were brought into the program in the sequence of the ADDROUT records, the file data would appear to be sorted in the program. In addition to being very clever, this saved much disk space. CL programs using ADDROUT sorts continue to run in System I shops today.

The entries for RAF include the following:

Blank Relative record numbers are used to process the file.
Records are read consecutively.
Record address file contains relative-record numbers.
Keys in record-address-limits file are in the same format as keys in the file being processed.

A Character keys (valid only for program-described files specified as indexed files or as a record-address-limits file).

P Packed keys (valid only for program-described files specified as indexed files or as a record-address-limits file).

K Key values are used to process the file. This entry is valid only for externally described files.

Figure 6-1 shows the K entry for the CUSTMSTF designating that a key would be used and that the file was indexed.

In RPGIV, the RAF type is specified in position 34.

F Column 32 File Organization

Column 32 is used for Internally Described Files only.
.
Other than the blank entry which is for both program described and externally described files, the two entries that may be specified in this column are for program described files only. The possible entries are as follows:

Blank The file is processed without keys, or the file is externally described.
I The file is an Indexed file. (program-described files).
T The file is a Record address file that contains relative-record Numbers (valid only for program-described files).

In RPGIV, the File Organization is specified in position 35.

F Columns 33-34 Overflow Indicator

When a printed form passes the last line of print on a form, RPG can sense that the overflow has occurred and it can communicate that fact to the programmer through a number of indicator choices OA-OG, and OF. By habit, I use OF because it reminds me of the word overflow. However, with some programmers writing programs that produce two or more reports on different printers, it is good that RPG permits eight different overflow indicators to be used – one for each of up to eight printer files. This is the place in the program in which you tell RPG what the overflow indicator is going to be for a particular printer file.

The entries for overflow are as follows:

Blank No overflow indicator is used.
OA-OG, OV Specified overflow indicator conditions the lines to be printed when overflow occurs.

01-99 Set on when a line is printed on the overflow line, or

the overflow line is reached or passed during a space or skip operation.

Indicators OA through OG, and OV are not valid for externally described printer files. Use positions 33 and 34 to specify an overflow indicator to condition which lines in each PRINTER file will be printed when overflow occurs. This entry is valid only for a PRINTER device. Overflow only occurs if defined.

Only one overflow indicator can be assigned to a file. If more than one PRINTER file in a program is assigned an overflow indicator, that indicator must be unique for each file.

F Columns 35-38 (Key Field Starting Location)

This entry if for internally described files only. The possible entries for these RPG/400 columns are as follows:

Blank Key fields are not used for this program-described file, or the file is externally described.
1-9999 Record position in a program described indexed file in which the key field begins.

This area is not used for externally described files. For program described index database files, use positions 35 through 38 to identify the record position in which the key field for the indexed file begins. An entry must be made in these positions for a program described indexed file. The key field of a record contains the information that identifies the record. The key field must be in the same location in all records in the file. The entry in these positions must be right-adjusted. Leading zeros can be omitted.

In RPGIV, the information for the key field starting location for program described files is now in a keyword with this format:

KEYLOC(number)

F Column 39 (Extension Code)

As RPG was developed, the File Description specification could no longer hold all of the information that was needed to handle the requirements of the language. Since much of the information that needed to be added had to do with data, IBM created what was originally called the File Description Extension form. It was an extension to File Description. We cover this form in detail in Chapter *****. Over time, since the File Description Extension itself grew in size, and some of the material did not relate at all to files, IBM reduced the size of the File Description Extension moniker to just Extension. Without a historical perspective, the word Extension has no real meaning.

As internal line control facilities began to take over in printers from the old carriage control tapes, the line counter specification was also devised as another file extension. However, since the word extension had already been taken, IBM fashioned the L spec or Line Counter specification to help with printer files.

To link the File with the extension or line counter, the compiler writers included a column in File Descriptions that alerted the compiler that there was an extension specification expected for the file being defined. The entries for the Extension Code therefore are as follows:

Blank No extension or line-counter specifications are used.
E Extension specifications further describe the file.
L Line counter specifications further describe the file.

Use position 39 to indicate whether the program-described file is further described on the extension specifications or on the line counter specifications (printers). An E in position 39 applies only to array or table files or to record-address files; an L in position 39 applies to files assigned to the PRINTER device.

Extension and Line Counter specifications in RPGIV have been replaced by a combination of keywords and the new "D" specification. This entry is therefore moot in RPGIV.

F Columns 40-46 (Device)

Positions 40-46 of file description is where you specify the name of the generic device type that will be used in the program. In essence this is where the file name is linked with the specific device type that the file name references.

The possible entries for the device in RPG programs are as follows:

PRINTER	File can have control characters for printers
DISK	File is a database file on a disk device
WORKSTN	File is a workstation file – terminal I/O
SPECIAL	User supplies special routine for device

In the PAREG program, we used positions 40 through 46 to specify the RPG/400 device name to be associated the three files in the program. The file names EMPMAST, TIMCRD, and QPRINT were specified in positions 7 through 14. EMPMAST and TIMCRD are designated as DISK files and QPRINT is designated as a PRINTER files.

Sine the AS/400 has natural print spooling. The RPG program will not ever have to communicate directly with a printer. However, it will send the appropriate control characters for printing through the device file in play (QPRINT in this case). The output will be in an output queue associated with the user's job and it can be printed by using the operating system facility known as spooling by starting a "writer" against an output queue. If this presents an issue for you in your shop, see your system administrator since each shop may have different printers and different printer standards.

The **WORKSTN** device in RPG (Figure 6-1) permits files that are created using a tool called SDA (Described in Appendix *****) to be linked with the RPG program. Using this facility, RPG programmers can use the WRITE or READ operations to WRITE or READ full screen panels of data to terminal or PC emulated displays. RPG also has an operation that we will discuss that performs both a WRITE to a Display and a READ from a display in just one operation. The RPG operation

code for this is EXFMT. Since there are no interactive devices in the PAREG program we defer this discussion until Chapter ******.

We have completed all we need to know about the very valuable File Description Specification for our PAREG problem. There are some more entries that can be made on file descriptions so, for completeness these are examined below.

In RPGIV, the Device is specified in positions 36-42.

F Positions 47-52 (Reserved)

Positions 47 through 52 are not used in PAYREG and for all programs, these positions must be blank.

F Position 53 (Continuation Lines)

RPG supports the ability to have special things defined in File Descriptions for which there is no room on the F spec. So, a continuation of the F spec is permitted by placing a K in position 53. This indicates a continuation line. We need no continuation lines in the PAREG program.

The functions implemented via File Description continuation line entries are covered after column 80 of File Descriptions. Most of the facilities provided via continuation entries are provided in RPGIV via keywords.

F Positions 54-59 (Routine)

When you must use SPECIAL as the device entry (positions 40 through 46), you also must name a routine in positions 54 through 59 to handle the support for the special device. The routine name must be left-adjusted within these columns. This entry is used by the compiler to produce the linkage to the routine. The PAREG program used no special routines and therefore these entries were not needed.

F Positions 60-65 (Reserved)

Positions 60-65 are not used in PAYREG and for all programs, these positions must be blank.

F Position 66 (File Addition)

Sine the PAREG program is database input-only, there was no need to specify to the compiler that the file described in positions 7-14 might have records added to it during processing. However, in future programs in this book, we will be adding records to files, and when we do, this entry will be needed.

The allowable entries for column 66 are as follows:

A	Records will be added to the file
Blank	Records will not be added to the file

An "A" in position 66 indicates whether records are to be added to a DISK file during processing. A blank means records will not be added For an output file (O in position 15), however, a blank is equivalent to an A since by definition output means added records.

F Positions 67-70 (Reserved)

Positions 67-70 are not used in PAREG and for all programs, these positions must be blank.

F Positions 71-72 (File Condition)

Programmers sometimes try to get all they can into one program and they will often use the same program to provide two functions that are similar, rather than write a second program. Sometimes, the difference in program is the input or output form that the job requires. To help

programmers in their efforts to be efficient, RPG provides the ability to include or exclude certain files at execution time withng a facility called an external program switch. The PAREG program does not use an external switch nor do any other programs examples in this book. However, since you may find it a handy tool, we are providing a description of the function and how to make it work for you.

The entries that can be used in 71-72 are as follows:

Blank The file is used by the program, if input file, it is opened.
U1-U8 The file is used by the program only when the indicator is on.
UC The programmer controls the first open of the file

When switches 1 to 8, also known as U1 to U8 (U for user switch) are used, the effect on the file description is ignored when the indicator is off. Thus it is like a blank being specified. When UC is sued, RPG does not automatically open the file for use as it usually does. Instead, the programmer controls the first open using the OPEN operation in calculations.

F Positions 73-74 (Reserved)

Positions 73-74 are not used in PAREG and for all programs, these positions must be blank.

F Positions 75-80 (Comments)

Since most of the file description form is used with real, live potential entries, there is not much left for comments. Positions 75 to 80 can be used for comments, or left blank.

FC File Description Continuation Lines

Even with the File Description Extension specification as well as the Line Counter specification, (column 39) both of which extend or continue the File Description specification, there still is not enough room in the File

Description area to provide for all of the options that must be specified. Therefore, IBM has defined a continuation column (53) by which the entries for File Description can be "extended even further."

A continuation line can be specified on the main file-description specification line if the functions use positions 54 through 65 for their definition. However, the use of certain keywords defined below, such as SFILE, RENAME, IGNORE, and PLIST cannot be defined on the same spec with the main file description line. For these and for situations in which multiple keywords need to be used, additional lines cane be used to continue the File Description form. Any number of continuation lines can be specified. A continuation line is indicated by a K in position 53.

FC Columns 7 – 18 Unused

These positions must be blank for a separate continuation line.

FC Columns 19 – 28 External Name of Record Format

These columns positions are used to specify the external name of the record format that is to be renamed (RENAME) or ignored (IGNORE).

FC Columns 29 – 46 Unused

These positions must be blank for a separate continuation line.

FC Columns 47 – 52 Record # Field for Subfile

When programming to use interactive workstation capabilities in RPG/400, there is a facility known as a subfile that can be appended to the display file through the use of DDS and specific RPG entries to

support the use of the subfile. A subfile in essence is a memory file of repeating lines (records) that would appear on a display. The memory file can be accessed within a program by record number. In columns 47 – 52, you specify the name of the numeric field that will contain the relative record # associated with the subfile.

This filed name gets specified ona subfile iptions keyword (SFILE). For the SFILE options, these positions must specify the name of a Relative Record Number (RECNO) field. For other continuation line options, these positions (47 – 52) must be blank.

FC Column 53 Continuation Character (K)

A "K" indicates a continuation line.

FC Columns 44 – 59 & 60 – 67; "K" Options

These positions are used together. Positions 54 through 59 specify the option, while positions 60 through 67 provide further explanation of the option. See "Continuation Line Options Summary Chart" in Figure 6-6 for a look at the available continuation options and coding requirements.

FC Columns 68 – 74 Unused

These positions must be blank for a separate continuation line.

FC Columns 75 – 80 Optional Comments

This space is available for comments

FC Options and Entries for Continuation

Many of the entries in Table 6-6have to do with device files used for data communications or for WORKSTN files. Though there is a place for these options and entries, this is not a beginner's topic so if this stuff is as clear as mud as you look at the table, that's OK for now. One day, when you need a tool, such as those described in the table, you will know they

exist, and you will know where to find them. The function and purpose of the tools below are provided in very brief terms with the intention of providing a light familiarity. When the reader chooses to use this material, what is provided below will help you but for a complete explanation, you will need to consult the IBM manuals to get the grit and detail. The valid entries for positions 54 through 67 are shown in the chart in Figure 6-1.

Figure 6-6 Continuation Line Options Summary Chart

Option (54-59)	Entry (60-67)	Function & Purpose
COMIT	**Blank**	**This entry is specified to indicate that the file is opened for commitment control. This enables the COMIT and ROLBK operation codes in RPG/400.**
ID	**Field Name**	**Provide the name of a 10 character field to capture the name of the device providing the record to the program.**
IGNORE	**Blank**	**Permits you to selectively ignore specific record formats from files. The record format name would be specified in positions 19 to 28 of the continuation line**
IND	**Indicator**	**General indicators from 01 to the indicator specified in 60-67 are saved and restored for each device that is attached to this program.**
INFDS	**DS Name**	**Provide the name of a data structure that RPG can use to contain the exception / error information associated with device operations. The DS name is entered in columns 60-65 and left justified. If multiple INFDS are used in a program, each must be given a unique name in this area.**
INFSR	**Subroutine Name**	**The file exception/error subroutine named (left justified) in positions 60 through 65 may receive control following file exception/errors. The subroutine name may be *PSSR, which indicates that the user defined program exception/error subroutine is to be given control for errors on this file.**
NUM	**Maximum**	**Workstation programs can acquire workstation. In most programs, just one device, the requester of the program, is used with a program. However, some programs are written to acquire devices and begin communication**

		with them for a program purpose.
PASS	***NOIND**	**This facility exists mostly to accommodate programs coming from other systems such as System/3 or System/36. If you were to write a program fresh today, you would not use program described workstation input and this keyword would be moot. However, to use this facility, you would specify PASS *NOIND on the file description specification continuation line for a program described WORKSTN file if you are taking responsibility for passing indicators on input and output. With PASS *NOIND, the RPG/400 language does not pass indicators to data management on output and does not receive them. In this scenario, you would pass indicators by describing them as fields (in the form *INxx, *IN, or *IN,xx) in the input or output record.**
PLIST	**Paramete r List Name**	**This entry is valid only when the device name specified in positions 40 through 46 of the main file-description line is SPECIAL. Positions 60 through 65 give the left-justified name of the parameter list to be passed to the special routine. The parameters identified by this entry are added to the end of the parameter list passed by the program.**
PRTCT L	**Data Struc- ture Name**	**There is a facility for program described priner files called dynamic printer control. If this keyword is used, the option is being used**
RECNO	**Field Name**	**This entry is optional for disk files to be processed by relative-record number. A RECNO field must be specified for output files processed by relative-record number, output files that are referenced by a random WRITE calculation operation, or output files that are used with ADD on the output specification.**
RENAM E	**Rec Format Name**	**This entry, which is optional, allows you to rename record formats in an externally described file. Positions 19 through 28 of the continuation line specify the external name of the record format that is to be renamed. Positions 60 through 67 specify the left-justified name of the record as it is renamed for use in**

		the program.
SAVDS	**Data Struc-ture Name**	**Positions 60-65 contain the left-justified name of the data structure saved and restored for each device. Before an input operation, the data structure for the device operation is saved. After the input operation, the data structure for the device associated with this current input operation is restored.**
SFILE	**Record Fmt Name**	**If the main file-description line contains E in position 19 and WORKSTN in positions 40 through 46, this option must be used to define any subfiles to be used in the file. Positions 60 through 67 must specify, left justified the RPG/400 name of the record format to be processed as a subfile.** **Positions 47 through 52 must specify the name of the relative-record number field for this subfile. The relative-record number of any record retrieved by a READC or CHAIN operation is placed into the field named in positions 47 through 52. This field is also used to specify the record number that RPG/400 uses for a WRITE operation to the subfile (memory file) or for output operations that use ADD.**
SLN	**Field Name**	**Positions 60-65 contain the left-justified name of a start line number (SLN) field. The SLN field determines where (which line #) a record format is written to a display file. The main file-description line must contain WORKSTN in positions 40 through 46 and a C or O in positions 15. The data description specifications for the file must specify the keyword SLNO(*VAR) for one or more record formats.**

RPGIV File Description Keywords

Just like with the Header Specification, RPGIV handles many of the items that were required to be specified with columns in RPG/400 File Descriptions using keywords. In Figure 6-7, we show a sample of six lines of RPG code with some keywords being used.

Figure 6-7 Sample RPGIV File Descriptions with Keywords

```
FFILE1P   O   E            DISK     EXTMBR('FILE1ALL')
F                                   RENAME(FILE1PR:ALLPR)
FFILE2P   O   E            DISK     EXTMBR('FILE2LST')
F                                   RENAME(FILE2PR:LSTPR)
FFILE3P   O   E            DISK     EXTMBR('FILE3OTH')
F                                   RENAME(FILE3PR:OTHPR)
```

Table 6-8 shows the keywords, samples in action, options, and the meaning of File Description keywords. RPGIV provides positions 44 to 80 of the F specification for stringing out keywords. Multiple keywords can be placed on each line and multiple lines can be used to accommodate necessary keywords.

Table 6-8 RPGIV File Description Keywords

Keyword / Sample	Other Options	Meaning
BLOCK(*YES)	***NO**	**Should records fro this file be processed in a block?**
COMMIT(YESORNO)	**RPG_name**	**Enable commit with a "1" value**
DATFMT(*ISO)	**Format{separator}**	**Specifies the date fromat**
DEVID(devname)	**Field name**	**Field contains the name fo the device supplying the last record processed.**
EXTFILE(RPGBOOK/ TIMECD)	**Filename; libname/filename *LIBL/filename**	
EXTIND(*INU1)	**(*INUx)**	**Open file if external indicator is on.**
EXTMBR(MEMBER2)	***First; *All, member name**	**Specify specific member name in file to be opened**
FORMLEN(50)	**number**	**Specify forms length in lines for line counter function**
FORMOFL(44)	**number**	**Specify line # which turns the overflow indicator on.**
IGNORE(RFMT01:	**(recformat{:recform**	**Ignore one or many record**

RFMT02)	at...})	formats from this file.
INCLUDE(RFMT01: RFMT02)	(recformat{:recformat...})	Include one or many record formats from this file.
INDDS(INDICATORS)	DS_Name	Load workstation indicators into this structure during execution
INFDS(FEEDBACK)	DS-Name	Name the info DS that will be associated with this file
INFSR(SUBNAME)	*PSSR; (SUBRname)	Name subroutine to get control if error.
KEYLOC(45)	Number	Specify the location of the key in an program described file.
MAXDEV(*ONLY)	*FILE	Restricts # of acquired devices such as workstations
OFLIND(OF)	OA-OG, OF	Specify the overflow indicator for a printer device file
PASS(*NOIND)	No other option	Do programmers control indicator passage?
PGMNAME(SPECDEV)	program_name	Provide name of program to handle a special device.
PLIST(BIGPARMS)	(Plist_name)	Provide the name for a parameter list to be used with called or calling programs.
PREFIX('GRP1.')	(prefix{:nbr_of_char_replaced})	Prefix is appended to the beginning of each field in this file to avoid duplicate names when used with other files.
PRTCTL(FRMCTLDS)	(data_struct{:*COMPAT})	Provide name of forms control data structure
RAFDATA(RAFFILE)	Filename	Specify the name of the RAFFILE to control record processing for this file
RECNO(RRN)	(Fieldname)	Specify the field name to contain the record # to write output records to a direct file
RENAME(FM1:FM2)	(Ext_format:Int_format)	Rename the external record format for program use to avoid conflicts
SAVEDS(SAVEFIELDS)	DS Name	Provide a DS name so that RPG will save fields for each device before each input operation.
SAVEIND(55)	Number	Similar to DAVEDS. Specify how many indicators you want saved.
SFILE(SFL3:RRN3)	(recformat:rrnfield)	Specify name of subfile format and the field name to be used for subfile record

		processing.
SLN(15)	**Number (1 to 24)**	**Starting line # to place record formats for this workstn file. At least one screen panel must use SLNO(*VAR)**
TIMFMT(*ISO)	**format{separator}**	**Specify time format**
USROPN	**No other values**	**This file will be opened in the program – RPG will not automatically open it.**

L -- Line Counter Specification Form

The Line Counter specification was designed as an extension to File Description for printer files. Its purpose is to provide the programmer with a means of defining necessary things such as the forms length and overflow position of special forms such as invoices, statements or checks. The form is quite simple, and though we did not need a line counter in the PAREG program, the entries in the example shown below reflect the default line counter used in PAREG.

When you line up all the forms for an RPG program, you place the Line Counter specifications right after File Description specifications unless there are Extension specifications. If there are Extension specifications, then Line Counter follows Extension. Figure 6-9 shows the Line Counter specification default that was used in the PAREG program.

Figure 6-9 Default Line Counter Specification

```
FMT *   ..... *. 1 ...+... 2 ...+
0004.00      L*
FMT L   .....LFilename066Fl060Ol.
0005.00      LQSYSPRT 066FL060OL
```

L Column 7-14 File Name

In columns 7-14, specify the file name of the PRINTER file as previously specified on file description specification form.

L Column 15-17 Lines Per Page

In Columns 15 – 17, specify the number of printing lines on the form. The available entries are 2 through 112.

L Column 18-19 Form Length

Specify the letters "FL" in columns 18 – 19 to indicate that the entry in positions 15 through 17 is the form length. Positions 18 and 19 must contain `FL` if positions 15 through 17 contain an entry.

L Column 20-22 Overflow Line Number

The line number you specify in columns 20-22 is the overflow line number. When printing hits this point on a page, the RPG program turns on the overflow indicator OA-OG, OF to inform the program that the last print line on the form has been reached..

L Column 23-24 Overflow Line Indicator

Specify the letters "OL" in columns 23 – 24 to indicate that the entry in positions 20 through 22 is the overlow line. Positions 23 and 24 must contain OL if positions 20 through 22 contain an entry.

L Column 25 - 74 Blank

L Column 75 - 80 Optional Comments

RPG IV Line Counter Information

In RPGIV, the columns of the Line Counter specification are handled by the FORMLEN and FORMOFL keywords. They are specified in File Description for the printer file. The File Description specification and the RPGIV keywords are described in the next section.

Chapter 7

The Specifics of RPG Coding – Input – by Example

The Many Faces of RPG Input

RPG is an evolving language. At the time of its introduction in 1988, for example the base RPG in RPG/400 was well over 30 years old. Each year as the specifications for RPG got better and better, the challenge for the designers was hot to fit the new function in the same number of specifications and the 30-year old 80 column limitation.

From your reading of Chapter 6, you now know that File Descriptions were expanded in three different ways. Line Counters specs (L) were built to better describe more modern printers. Extension (E) specifications were built to handle arrays and tables (– covered in Chapter ***), and the File continuation was introduced to deal with the nuances of workstation and telecommunications files. In fact, for a brief period, File Description actually spawned a third spec called the Telecommunications (T) specification which was used to talk to batch terminals in the 1970's before the System/38 was introduced.

So, you will not be surprised that the Input specification is also overloaded with baggage. First of all, input specifications always had two purposes: (1) identify records as they are read and (2) define the data. So, there were always two formats to the Input specification. In 1978 with the introduction of the System/38, IBM introduced RPGIII. This flavor of RPG permitted the RPG input specifications to be fetched and formatted from the database itself by the compiler at compile time.

This saved tremendous amounts of coding but it added a new wrinkle to input specs. When input needed to be defined or changed for one reason or another, such as reducing the size of a database field from 10 characters to 6 for the RPG limitation, a means on the INPUT coding sheet needed to be created. Since Database files do not like more than one record type, the old way of identifying records needed to be replaced in such a way that RPG could create a place in input for each different screen format or each format in a logical file. Additionally, Input needed to be provided for Control Fields and for Match fields even when the rest of the data descriptors were fetched from the database.

IBM answered the call with two more forms of the RPG Input specification, both to handle what is called externally described data. Since the new formats handled externally described data, the old formats were then said to use internally described data or program described data – synonyms for the notion that the data elements used in the program were the ones described in the program – regardless of what was in the database. Just as with the program described data, the one format handled record IDs and the other handled field definitions.

The System/38 also introduced new uses for the Input specification: Data structures and named constants. Unlike COBOL which has a Data Division that does not care about whether a data element is used for input or output, RPG had only input and output specs for major data definitions and operations. Since it did not make sense to define data in output, the named constant took on a new format of the input spec as did the data structures. Named constants are covered in Chapter 8 and Data structures are covered both in Chapter 8 and in Chapter ****.

Though named constants and structures are important, we would be getting ahead of ourselves at this point since the PAREG program which is still our simple guiding example, gets along fine without either.

Internally & Externally Described Input

Externally described input files are far more efficient to code than program described (internally described) input files. However, because there are still many RPG programs that use program described data, especially in shops that have migrated from System/36 machines, we

cover program described data for completeness. Since PAREG uses externally described data, this chapter first defines the structure of the external form for input and then goes back for completeness and fills in the blanks for the internal / programmed described format of Input.

I – Input Specification Form

For an externally described file, input specifications are optional. Yet, they have value in instances in which the programmer would like to add RPG/400 functions to the external description – such as matching records and level breaks. The PAREG program is an example. So we use Input specifications as we did in the PAYREG program, even though its input is externally described. This enabled us to describe the matching fields as well as the control fields needed in this program to create the report.

To make it easier for you to follow along with the role and the format of the input specification, we show the input portion of the compile listing of the PAREG program in Figure 7-1. This is important in understanding internal and external data descriptions in RPG, since as you can see in the with the input specifications and the expanded source.

Figure 7-1 Compile Listing of External Input Expanded

```
   1000  I* RPG INPUT SPECIFICATION FORMS
   1100  I*
   1200  IPAYR        01
   1300  I              EMPNO                          EMPNO   M1
   1400  I                                             EMPCTYL1
   1500  I              EMPSTA                         EMPSTAL2
   1500   INPUT  FIELDS FOR RECORD PAYR FILE EMPMAST FORMAT PAYR.
A000001                 EMPNO                   1   30EMPNO   M1
A000002                                         4  23 EMPNAM
A000003                                        24  282EMPRAT
A000004                                        29  48 EMPCTYL1
A000005                 EMPSTA                 49  50 EMPSTAL2
A000006                                        51  550EMPZIP
   1600  ITIMR        02
   1700  I              EMPNO                          EMPNO   M1
   1700   INPUT  FIELDS FOR RECORD TIMR FILE TIMCRD FORMAT TIMR.
B000001                 EMPNO                   1   30EMPNO   M1
B000002                                         4   72EMPHRS
```

We have already learned that any statement with an asterisk in column 7 is a comment and thus, statements 10 and 11 are comments. Statement 12 is a record format identifier, whereas statements 13 through 15 are the field definitions for the payroll master. As you may be able to tell by looking at Figure 6-1, the format of the record identifier and the format of the field definitions are substantially different. So, unlike the File Description Specification (F- spec), which has just one format, for externally supplied input, there are two very different types of input specifications.

For the externally described files such as the two we have coded in the PAREG program, entries on the input specifications are divided into the following categories:

1. Record identification entries
   (positions 7 through 14, and 19 and 20)
   These identify the record (the externally described record format) to which RPG/400 functions are to be added.

2. Field description entries
   (positions 21 through 30, 53 through 62,and 65 through 70).
   These describe the RPG/400 functions to be added to the fields in the record.

Field description entries are always written on the lines following the corresponding record identification entries. For data structures, which are fully described in Chapters 8 and *****, the entries on input specifications are divided into the following categories:

1. Data structure statements
   (positions 7 through 12, 17 through 30, and 44 through 51).
   These entries define data structures to RPG.

2. Data structure subfield specifications
   (positions 8, and 21 through 58)
   These entries describe subfields of the data structures.

Just as the field descriptors for input, data structure subfield specifications are written on the lines following the data structure statements.

Input Spec Quick Summary

As a quick summary on the externally described versions of the record and field oriented input specification, it is clear to see that the I spec is a real workhorse for RPG. Ironically, its overuse for non-input functions is one of the major reasons why IBM created the data specification ("D" spec) for RPG IV. This "D" spec resembles the Data Division in COBOL. For RPG/400, however, since there is no "D" spec, you will find machinations of the good old input spec helping you with the coding for all of the following RPG facilities:

- ✓ Entries for program described files
- ✓ Entries for externally described files
- ✓ Entries for data structures
- ✓ Entries for named constants

RPG Input Form Types

To help you gain a better appreciation for the many combinations of input specification formats, a picture is worth a thousand words. Years ago, RPG programmers would code their programs on paper sheets such as the one in Figure 7-2. As you can see by examining Figure 7-2, there are many different formats for RPG for both program described and externally described formats

Figure 7-2 RPG Input Specification Form – Entry Choices

Filename or Record Name
Data Structure Name
Sequence
O R
A N D
Number (1/N),E
Option O,U,S,I
Record Identifying Indicator, **, or DS
Subfield Initialization Value
Named Constant Value
External Field Name
Record Identification Codes
1 — Position, Not (N), C/Z/D, Character
2 — Position, Not (N), C/Z/D, Character
3 — Position, Not (N), C/Z/D, Character
P/B/L/R/C
Field Location
From To
Data Structure
Occurs n Times
Length
Decimal Positions
RPG Field Name
Constant Name
Control Level (L1-L9)
Matching Fields or Chaining Fields

Program Described Files
Record Identification Entries
Field Description Entries
Externally Described Files
Record Identification Entries
Field Description Entries
Data Structures
Data Structure Statement
Data Structure Subfield Specifications
Named Constants

PAREG Record and Field Statements

The PAREG program needs the external form for both the Record ID Entries and the Field Description Entries. As we examine the PAREG coding in light of these forms, you will see how the Record ID part helped us identify the primary and secondary files with record identifying indicators. You will also see how the Field part provided space for the entries necessary both match fields and control levels and you will see that the match fields help assure that there is an in-sequence time card record for each payroll master record. So, there is no doubt they both halves of the external input format are necessary in RPG and no doubt that both are absolutely necessary for the success of the PAREG program. Let's first look at the record ID part of input and then we'll move to the field description part.

I Externally Described Record ID Entries

Externally described file input specifications provide additional coding for records to be processed in an RPG program. Since their typically is no Record ID character in database files with multiple record types, input specs are used to differentiate records. Additionally, the record format can be renamed as needed using input specifications specifically designed for externally described files. On the field side, of course input specs permit fields to be referenced so that level indicators and matching fields indicators can be specified.

The following section describes in reasonable detail the entries for the Record ID part of the External Input form. Since RPGIV in many cases uses a different column for the same function, immediately after the description of the RPG/400 entry, the RPGIV function location is provided. The designator **IEDRI** is used to begin each header so it is easy to spot the entries that belong with Input, Externally Described, Record Identification.

IEDRI Columns 7-14 Record Name

For the record name specified in columns 7-14 of the Input specification, the allowable entries include the following:

1. The external name of the record format. The file name cannot be used or an externally described file.

2. The RPG/400 name specified by the RENAME option on the file description specifications continuation line if the external record format was renamed. A record format name can appear only once in positions 7 through 14 of the input specifications for an RPG program.

In RPGIV, the Record name is specified columns 7 – 16.

IEDRI Columns 16 – 18 Reserved

Columns 15-18 of the externally described input form are reserved for future use. No entries should be placed in these columns.

IEDRI Columns 19 – 20 Record Identifying Indicator

When an optional record identifying indicator is specified in columns 19-20, it will be turned on if the record format name in 7-14 is the one read by the program.

In RPGIV, the Record ID Indicator is specified in columns 21 – 22.

IEDRI Columns 21 – 41 Unused

Columns 21 to 41 are unused for externally described input specifications. They must be coded as blank.

IEDRI Columns 42 – 74 Reserved

Columns 42-74 are reserved for future use for externally described input specifications. They must be coded as blank.

IEDRI Columns 75 – 80 Reserved

Positions 75-80 of the externally described record ID input form can be used for comments, or left blank.

Applying INPUT Record IDs to PAREG

Now, let's take a look at the input records coming in from the EMPMAST and the TIMCRD files. In Chapter 3, we showed the DDS

in Figure 3-2 for EMPMAST. You may recall that the first line of the DDS described the record format. This line looked like this:

```
0001.00      A          R PAYR
```

DDS specs have their own spec type "A" in column 6. The R in column 17 says this is a record format line. In DDS, you place an R in this column to identify the record format by name. The name PAYR in column 19 is the name we gave to the record format of the PAYMAST file. Look at the similarities of this with the following RPG Input spec as originally typed in the PAREG program (Figure 5-1)

```
0012.00      IPAYR        01
```

The notion of a record format name is very similar to the record id notion within the RPG language as shown immediately above in the blown up input specification from line 12. In most cases, the coding is that simple.

There is a notion in System i5's DB2 native database coding called Logical files or views. These logical views can be built over one or several database files and can present data from multiple database files. All relational databases can present a JOIN view in which pieces of multiple records in multiple files make up one record in the joined logical view.

System i5's DB2 is the only database that can provide a hierarchical view of physical files with different formats in much the same way file systems handled multiple record types years ago. When a multiple format logical view is created over a number of physical files, the format name in the physical file, such as EMPMAST's PAYR and TIMECRD's TIMR becomes the format name used to differentiate the records in input specs when processed through the logical file. Instead of two files being defined to the program as in PAREG, for example, just the Logical File would be coded and it would presents both record types with their differing formats to the program in a pre-designated sequence such as by the EMPNO field..

So, if PAYR and TIMR record formats were in a logical view sequenced by EMPNO, the 55 character PAYR format would come in and then the 7 character TIMR format would come in through the one file description spec. The programmer would identify each of these with the IEDRI form – one for each record format and then would specify each of the format names in IEDRI exactly the same way as they are specified in the PAREG program. When PAYR is read, indicator 01 would be turned on as identification and when record TIMR is read, indicator 02 would be turned on.

Therefore, both logical files with multiple formats as well as primary and secondary files with one format each can be read by the fixed logic cycle of an RPG program such as PAREG. Therefore it is incumbent on RPG to have a vehicle to differentiate one record format from another. The record format input specification IEDRI provides this vehicle.

To help us remember what the Input Specs are like in PAREG, we present Figure 7-3. We have seen this Figure before as Figure 6-5 in Chapter 6. You may recall it is the compiled input specs. Therefore it includes external fields from the database that are not described in the program code. The program code has regular statement #s to the left whereas the code fetched from the database has long numbers such as A000001 beginning with a vowel to the left.

Figure 7-3 Compile Listing of External Input Expanded

```
   1000  I* RPG INPUT SPECIFICATION FORMS
   1100  I*
   1200  IPAYR        01
   1300  I              EMPNO                            EMPNO   M1
   1400  I                                               EMPCTYL1
   1500  I              EMPSTA                           EMPSTAL2
   1500   INPUT  FIELDS FOR RECORD PAYR FILE EMPMAST FORMAT PAYR.
A000001                 EMPNO                      1   30EMPNO   M1
A000002                                            4  23 EMPNAM
A000003                                           24  282EMPRAT
A000004                                           29  48 EMPCTYL1
A000005                 EMPSTA                    49  50 EMPSTAL2
A000006                                           51  550EMPZIP
   1600  ITIMR        02
   1700  I              EMPNO                            EMPNO   M1
   1700   INPUT  FIELDS FOR RECORD TIMR FILE TIMCRD FORMAT TIMR.
B000001                 EMPNO                      1   30EMPNO   M1
B000002                                            4   72EMPHRS
```

When an RPG programmer decides that matching or control fields must be specified, the language requires that for an externally described file, a record format must precede the field specs. After the "I" in column 6, (columns 7 through 14) the programmer specifies the name of the format that matches the name of the format in DDS, which to be more exact is the format name in the physical database file itself.

After the format name, if you keep moving right along the RPG form, you will come to an '01" starting in position 19. This is very important for this program. The "01" is the record identifying indicator for the EMPMAST file. Just what does this mean? It means that whenever a record is read during the fixed logic cycle, RPG will check to see if it is a record from the PAYR record format. in the database. If it is, then RPG will turn on indicator "01" so it can be used within the program.

In older RPG's, and for program described data, 7 – 14 would contain the name of the File (EMPMAST) as defined in the File Descriptions this the. For externally defined files, however, RPG demands that the programmer specifies the same format name as the name used to build the database.

Once we know how to describe an input record format for one RPG file, we can describe it for any. Let's take a shot at decoding the record format for the TIMCRD file which is found on the PAREG program at statement 16.

```
0016.00 ITIMR          02
```

Other than the sequence number and the I in column 16, again the programmer needs just two entries to make this external record ID work in the PAREG program The DDS for the TIMCRD database file were originally shown back in Figure 3-3. Just as PAYR is the record name specified in DDS in Figure 3-2 for the EMPMAST database file, TIMR is the record format name specified in DDS for the TIMCRD database file. Thus in the input statement defined at statement 16 in the PAYREG RPG program, TIMR is the record format specified. Additionally, the programmer has told RPG to turn on indicator "02" as a record identifying indicator whenever a time card record is read.

That's' the essence of the record format input specification. For the PAREG program, by turning on indicator 01, it tells RPG that a EMPMAST record was read and it is available for processing in this detail cycle. By turning on indicator 02, it tells RPG that an TIMCRD record was read and that it is available fro processing this detail cycle. As you may recall from Chapter 1, right before RPG reads from the file last processed, it does housekeeping by turning off all of the matching and record identifying indicators so that when it turns on indicator "01" or indicator "02" there is no residue left from other detail cycles. If the status of Indicator 01 in the PAREG program is "ON," then the Master has just been read. On the other hand, if the status of indicator "02," is on, then the time card record has just been read.

I Externally Described Field Description Entries

Unlike the program described record ID and field description portions of the RPG input specification, which is really an 80 column form cut just about in half, the externally described versions each start at position 7 of the input specification. In this section, we examine the field description entries for externally described files.

The designator **IEDFD** is used to begin each header so it is easy to spot the entries that belong with Input, Externally Described, Record Identification.

Field Specifications in RPG/400

There are more entries in the field specification form in RPG/400 than we need for PAREG so while we are examining the entries in PAREG, we will also explain any other entries on the form type. The field entries from the EMPMAST are listed below followed by blank line and the fields from the TIMCRD file.

Figure 7-4 Field Entries as Coded in PAREG

```
        678911234567892123456789312345678941234567895123456789612
0013.00 I              EMPNO                            EMPNO   M1
0014.00 I                                               EMPCTYL1
0015.00 I              EMPSTA                           EMPSTAL2

0017.00 I              EMPNO                            EMPNO
```

IEDFD Columns 7 – 20 Reserved

Columns 7 – 20 of the externally described field description form of the input specification has reserved positions for future use, thus none of these positions were coded for PAREG

IEDFD Columns 21 – 30 External Field Name

For the PAREG program, no entries need to be made in this area. Yet, if you look at the four field entries specified in this program, you will see that three are entries for three of the four input fields. This area needs to be used only when you have chosen to rename a field from the database. Say, you want to reference the field called EMPNO, for example as EMP# in the program. By specifying EMPNO in this area and then specifying EMP# in the program field area (53-58) the field name will be EMP# when referenced in this program

A big reason for renaming fields is that sometimes database administrators use all ten positions of the field name in DDS. Since RPG/400 field names can be just 6 characters in length, a rename would be necessary to use the field in the program.

There are no such examples in the PAREG program. However, we did add an entry for all fields but EMPCTY of the EMPMAST file in this external name for rename area. This was unnecessary. It was done merely to show you how a rename of a field could readily be accomplished.

In RPGIV, the External Field Name is also specified in columns 21 – 30.

IEDFD Columns 31 – 52 Reserved

Columns 31 – 52 of the externally described field description form of the input specification has reserved positions for future use.

IEDFD Columns 53 – 58 Field Name

For files that are externally described, no input specifications are required unless the programmer uses special RPG functions that depend on input lines being marked appropriately. In other words, the field name entry is made only when it is required for the RPG/400 function (such as control levels or matched fields or renamed fields) that must be added to the external description.

The four field name entries (lines 13 to 15 from the EMPMAST database and line 17 from the EMPTIN database) as defined in PAYREG are easy to spot in Figure 7-4. The field name entry can be specified with one of the following:

The name of the field as defined in the external record description (if 6 characters or less).

The name specified to be used in the program that replaced the external name specified in positions 21 through 30.

In RPGIV, the Field Name is specified in columns 49 – 62.

IEDFD Columns 59 – 60 Control Level

The entry in position 59 and 60 indicates whether the field is to be used as a control field in the program.

The allowable entries for positions 59 and 60 are as follws:

Blank This field is not a control field.
L1-L9 This field is a control field.

The input specifications for the fields in the EMPMAST file used to define control level totals are shown below:

```
0014.00 I…                            EMPCTYL1
0015.00 I…  EMPSTA                    EMPSTAL2
```

The code above tells the RPG compiler that the State field (EMPSTA) is a second level control field and any change in this field as primary (Payroll Master) records are being read will trigger a control break from which second level (L2) control level calculations and second level control level output can be created during the RPG cycle. If the City (EMPCTY) field changes while master records are being read, a first level break is triggered and those calculations and output designated to occur during an L1 break can then occur. A higher level break always triggers a lower level break. Thus in this example, if the state field changes, the city calculations (L1) will occur, followed by city total output (l1), followed by state total calculations (L2) followed by state total output (L2).

In RPGIV, the Control Level is specified in columns 63 – 64.

IEDFD Columns 60 – 61 Match Fields

This entry indicates whether the field is to be used as a match field. The allowable entries in columns 60 – 61 are as follws:

Blank This field is not a match field.
M1-M9 This field is a match field.

Match fields are the key to making this PAREG program work. The two lines that specify the match fields in the PAREG program follow this paragraph. The data records in both files are in sequence by EMPNO in order for this to work. They get in sequence using the IBM fort known as Format Data or FMTDTA. When the records match, RPG turns on a special indicator called MR to help control the operations of the program. Knowing a match has occurred is an extremely valuable piece of information as you will see in the rest of this program. When the “MR” indicator is turned on by RPG detecting a match between the primary and

secondary files, it can be used to further condition operations in the program.

```
0013.00 I...    EMPNO...        EMPNO     M1

0017.00 I...    EMPNO...        EMPNO     M1
```

The two fields above are shown in context in Figure 7-3 within their specific record formats (from the EMPMAST and TIMCRD database files respectively). Statement 13 belongs with EMPMAST and statement 17 belongs with TIMCRD.

As you can see on line 12 in Figure 7-3, the EMPNO field from the EMPMAST file (format PAYR) is matched on line 17 with the EMPNO field from the TIMCRD file (format TIMR from line 16) By placing M1 on each field, this entry is used to match the records of the one file with those of the other.

Sometimes the names for fill-in items in RPG do not make sense in a broad context. For example, the designator M1 can also be used to sequence check match fields within one file. Obviously with just one file specified, there could be no match and thus the designation of matched fields does not really work in that case. Nonetheless, columns 61-62 would be where the programmer would specify matching for just one file if the programmer would like the RPG compiler to sequence check the incoming data (ascending or descending – depending on the code in the file description specification). For PAREG, which has two input files, we get sequence checking of both files as well as matching.

What about M2 through M9?

M1 is the lowest order field that can be specified in a multi-field match. If, for example the EMPNO field were duplicated in departments in an organization, then the only way to assure a real match on employee would be to include the department number as a match field. Assuming that both databases were populated with a field called EMPDNO for employee department number (**Note lines 12.01 and 16.01**) the code would look like that shown in Figure 7-5.

Figure 7-5 Duplicate Employee # within Department #

```
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IPAYR        01
0012.01 I              EMPDNO...     EMPDNO   M2
0013.00 I              EMPNO...      EMPNO    M1
0014.00 I                            EMPCTYL1
0015.00 I              EMPSTA...     EMPSTAL2
0016.00 ITIMR        02
0016.01 I              EMPDNO...     EMPDNO   M2
0017.00 I              EMPNO...      EMPNO    M1
```

Match field designators M3 to M9 work in the same fashion. Thus, in RPG you can have up to nine match fields involved in a sequence check or a match. If they are in match relationship, all fields specified, M1 through M9 must match in order for the MR indicator to be turned on.

One more point on this example. To keep matters simple we chose to use the same field names in both databases for the EMPNO and EMPDNO fields. This is not necessarily a good idea in practice but it helps in teaching and learning. For the EMPMAST file, for example, it may be appropriate in our shop standards to begin each field with the three letters, EMP. However, for the TIMCRD file, using EMP for each field can create confusion when both files are used in the same program. Therefore, it would probably be more appropriate to use a prefix of TIM, rather than EMP for the fields in the TIMCRD file. This is fine with RPG. The field names do not have to be the same. In fact, the input specifications to accommodate this are shown in Figure 7-6..

Figure 7-6 Input Specs shown with Better Naming Conventions

```
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IPAYR       01
0012.01 I             EMPDNO…     EMPDNO  M2
0013.00 I             EMPNO…      EMPNO   M1
0014.00 I                         EMPCTYL1
0015.00 I             EMPSTA…     EMPSTAL2
0016.00 ITIMR       02
0016.01 I             TIMDNO…     TIMDNO  M2
0017.00 I             TIMNO…      TIMNO   M1
```

Some other helpful rules are as follpws:

1. Match fields can be specified only for fields in primary and secondary files.

2. Match fields within a record are designated by an M1 through M9 code entered in positions 61 and 62 of the appropriate field description specification line. A maximum of nine match fields can be specified.

3. The match field codes M1 through M9 can be assigned in any sequence.

4. When more than one match field code is used for a record, all fields can be considered as one large field. M1 or the lowest code used is the rightmost or low-order position of the field. M9 or the highest code used is the leftmost or high-order position of the field.

In RPGIV, the Match Fields are specified in columns 65 – 66. .

IEDFD Columns 63 – 64 Reserved

Columns 63 – 64 of the externally described field description form of the input specification has reserved positions for future use.

IEDFD Columns 65 – 70 Field Indicators

Though the program PAREG does not have a use for field indicators, columns 65 to 70 of the I specification field spec is where they are specified.

The entries for this area are as follows:

Blank	No indicator specified
01-99	General indicators for programmer use
H1-H9	Halt indicators – cause the machine to halt
U1-U8	External indicators – externally supplied
RT	Return indicator. – causes return to calling program

When you choose to supply entries in positions 65 through 70, you can test the status of a field or of an array element as it is read into the program during the input phase of the RPG cycle. The field indicators are specified on the same line as the field to be tested. Depending on the status of the field (65-66 - plus, 67-68 minus, 69-70 “zero, or blank”), the appropriate indicator is set on and can be

In RPGIV, the Field Indicators are specified in columns 69 – 74.

IEDFD Columns 71- 74 Reserved

Columns 71-74 of the externally described field description form of the input specification has reserved positions for future use

IEDFD Columns 75 – 80 Comments

Columns 75 through 80 can be used for comments, or left blank.

Now, to put the whole INPUT specification, let's start with the Record ID area of program described files, work our way to the filed area of program described files, then let's look at record IDs in externally described files and field entries for externally described files. Following this, we'll examine the Data Structure and the data structure subfield formats of the input specification. In many ways, you can easily conclude that there are in fact six different formats for the input specification and some for them have nothing to do with input. If you have come to that conclusion, you are correct.

I Program Described Record Identification Entries

Now, to put the whole INPUT specification, let's start with the Record ID area of program described files, and work our way to the field area of program described files. Following this, in Chapter 8, we examine the data structure and the data structure subfield formats of the input specification. From what we've done so far and where we are heading, you can easily conclude that there are in fact six different formats for the input specification and some of these have nothing at all to do with input. If you have come to that conclusion, you are mostly correct. There is actually one more variant of input that we cover in Chapter 8. This is the named constant and clearly a constant is something that does not get read in from an external device.

Program described input specifications describe everything about the types of records within the file, the sequence of the types of records, the fields within a record, the data within the field, the indicators based on the contents of the fields, control fields, fields used for matching records, and fields used for sequence checking.

In Chapter 5, we showed both the externally described version and the internally described version of the PAREG program. Figure 7-7 shows the input specifications for the internally described version of PAREG. The designator **IPDRI** is used to begin each header so it is easy to spot the entries that belong with Input, Program Described, Record Identification.

Figure 7-7 Internally (Program) Described Input for PAREG

```
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IEMPMAST AA  01
0013.00 I...                  1   30EMPNO   M1
0013.01 I...                  4  23 EMPNAM
0013.02 I...                 24  282EMPRAT
0014.00 I...                 29  48 EMPCTYL1
0015.00 I...                 49  50 EMPSTAL2
0015.01 I...                 51  550EMPZIP
0016.00 ITIMCRD  AB  02
0017.00 I...                  1   30EMPNO   M1
0017.01 I...                  4   72EMPHRS
```

IPDRI Columns 7-14 File Name

For **program described files**, you specify the File name of the input file in 7-14 – EMPMAST and TIMCRD in PAREG. It must be the same file name that you already described in the File Description area. In File Descriptions, this file must have been described as an input file, an update file, or a combined file. The file name must be entered on the first record identification line for each file and it can be entered on subsequent record identification lines for that file (lines 12 & 16 in Figure 7-7) but the file name can be skipped if the record id being defined is from the same file (such as in a multi-format logical file or a display file). All entries describing one input file must appear together (12 – 15.01 and 16-17.01); they cannot be mixed with entries for other files.

In RPGIV, the File name is in positions 7-16.

IPDRI Positions 14-16 (Logical Relationship)

For program described input, multiple tests can be performed on input data to determine which record has been read. The logical relationship is either coded in an RPG statement following a statement that has the file name in 7-14 or it follows a subsequent record definition. The logical relationship links multiple input tests together. The entries are as follows:

AND More than three identification codes are used.
OR Two or more record types have common fields.

PAREG has no logical relationships defined. If there were, the code might look as shown in Figure 7-8 for EMPMAST.

Figure 7-8 Logical Relationship – One Record

```
        4567892123456789312345678941
IEMPMAST AA  01  56 CA  57 CB  58 CC
I       AND      59 CD
Field Definitions here…
```

In this case, we are looking for real codes in the record (ABCD in 56 – 59) since that's how internally described programs are most often designed. If there were a second master record format in the file, say with a record code of ABCE, the two program described record IDs would be coded as in Figure 7-9.

Figure 7-9 Logical Relationship – Multiple Records

```
        4567892123456789312345678941
IEMPMAST AA  01  56 CA  57 CB  58 CC
I       AND      59 CD
Field Definitions here…
I        BB  03  56 CA  57 CB  58 CC
I       AND      59 CE
Field Definitions here…
```

In this example, indicator 01 would turn on for an "ABCD" master format and indicator 03 would turn on for an "ABCE" master format. The field from and to positions of each format with internally described data can be substantially different. The other parts of the specifications are about to be explained.

In RPGIV, the Logical Relationship is in positions 16-18.

IPDRI Positions 15-16 (Sequence)

The entries for Sequence are as follows:

AA-ZZ The program does not check for special sequence.
01-99 The program checks for special sequence within the group.

The Internally described PAREG program uses AA and AB for Sequence (Figure 7-7). An alphabetic sequence entry tells RPG to not check the record sequence of the input data. A numeric sequence entry does a lot of work for the programmer if sequencing of records is important to the application. Most often, especially with single record format disk files, the alpha sequence is used.

The numeric sequence entry works in combination with the number (position 17) and option (position 18) entries. It causes the program to check the sequence of input records within a file. If the sequence is not correct, control passes to the RPG/400 exception/error handling routine.

If AND or OR lines are specified, the sequence check is performed on the main record line of the group, not on the AND or OR lines.

Let's suppose that our data is coming from a program described file with multiple record types – payroll master file, time card file, and deduction file. In this scenario, each record is the same length since each is in the same file. Shorter record designs are merely padded with blanks at the end of the record. But the fields in the records do not have to line up and in fact will not. Each has its won layout though it exists in the same file. Many former System/36 applications are built like this. Let's look at the abbreviated record layouts in Figure 7-10. As you can see, besides a record identification field, we defined fields for each of these files. This figure shows how the format of each record is different

Figure 7-10 Three Record Types from Same File

Emp Master	From / To	Time Card	From / To	Deduction	From / To
RECID	**1 / 1 / A**	**RECID**	**1 / 1 / B**	**RECID**	**1 / 1 / C**
EMPNO	**2 / 5**	**EMPNO**	**2 / 5**	**EMPNO**	**2 / 5**
EMPNAM	**6 / 35**	**HOURS**	**6 / 8**	**DEDNO**	**6 / 7**
EMPAD1	**36 / 65**	**NA**	**9**	**DEDAMT**	**8 / 12**
# per EMPNO		**1**		**Up to 4**	

In addition to the length of the RECID field, we also show the constant contents. In this case, the master record has an A, the time card has a B, and the deduction record(s) has a C. Thus, each of the record types in the file are uniquely identified so that RPG can differentiate them on input and be able to turn on the appropriate record identification indicator.

Each of the files also has an EMPNO (employee number) field which permits the data to be presorted in record id within EMPNO sequence prior to running the program. After the EMPNO field the records begin to take different shapes. EMPNAM (employee name) for example is 30 positions in length, HOURS (hours) is 3 positions and DEDNO (deduction number) is two positions. The next field defined in each record starts in a different position – 36, 9, and 8 respectively. In fact, there is not a third field defined for time card in this design since there is already enough information in the record.

The preprocessing sort is key to understanding sequencing. All records are sorted within EMPNO sequence. If we have 100 employees and 3 records are processed for each employee for each payroll, then 300 records are processed. In groups of three each – one group of A, B, and C record types for each. In other words, one master, one time card, and one deduction record for each employee. The sequencing capabilities of RPG are very powerful in that for every group of records, A, B, and C, you get to say with a numeric value, whether the A should be first, the B, or the C.

In this example we defined our records in the sequence A,B, C that we would like them processed in each group. Since pay rate is typically in the master, we would have the master record (A record type) read 1st, then the time card record (B record type). Once the time card is read, we can calculate gross pay and "net pay before deductions." The deductions, (C record type) such as United Way and Savings bonds can then be processed (deducted) from the net pay before deductions to get the payroll check amount.

So, for RPG to assure that our records are in sequence, we can use the sequence entry and we would assign a 1 sequence to master, a 2 sequence to time card, and a 3 sequence to deductions. If ever the record types were not in this sequence for each employee, RPG would halt the program with an error message.

In RPGIV, Sequence is in positions 17 – 18.

IPDRI Positions 17 (Number)

For PAREG, since an alphabetic sequence # was used, no number entry is required for column 17. The allowable entries for the number position (column 17) are as follows:

Blank The program does not check record types for a special sequence (positions 15 and 16 have alphabetic entries).

1 Only one record of this type can be present in the sequenced group.

N One or more records of this type can be present in the sequenced group.

This entry must be used when a numeric entry is made in positions 15 and 16. If an alphabetic entry is made in positions 15 and 16, this entry must be blank.

To code for the example shown in Figure 7-10, the programmer would place a 1 in column 17 for the A record (sequence 01) since there is to be just one master; a 1 in column 17 for the B record (sequence 02) for the time card since there is to be just one time card; and an N in column 17 for the deduction record since there can be one or many of them. In other words, for each payroll group, we are asking RPG to check for one and only one master as the first record (A) type of the group as well as one and only one time card as the second record type (B) of the group. It gets trickier for the deduction record as we are saying that there can be multiples of the third record type . As your own experience with payroll indicates, employees can have many deductions – loans, United Way, Savings bonds etc.

In RPGIV, Number is in positions 19.

IPDRI Positions 18 (Option)

For PAREG, since an alphabetic sequence # was used, no Option entry is required for column 18. The possible entries for column 18 are as follows:

Blank The record type must be present if sequence checking is specified.

O The record type is optional (that is, it may or may not be present) if sequence checking is specified.

This entry must be blank if positions 15 and 16 contain an alphabetic entry. Sequence checking of record types has no real meaning when all record types within a file are specified as optional (alphabetic entry in positions 15 and 16 or O entry in position 18).

Going back to the three record type sample in our payroll example from Figure 7-10, we know that if there is a group of records for a particular employee (EMPNO), there must be at least a master and a time card record. The programmer would code column 18 as blank meaning that

these records are required in each group. If it is possible, such as with part-time workers, that an employee (even just one) would not have any deductions, then this column must be coded as O for optional so that RPG will not generate an error for these employees. So, if we code the deduction record as optional, if it (C record type) is not there, it is OK but of it is there, it must follow the B record type.

In RPGIV, Option is in position 20.

IPDRI Positions 19-20 (Record identifying Indicators)

Both forms of the PAREG program use record identifying indicator 01 for the EMPMAST and record identifying indicator 02 for the TIMCRD file. The possible entries for positions 19-20 (Record Identifying Indicator, or **) are as follows:

Blank	No indicator is used.
01-99	General indicator
L1-L9 or LR	Control level indicator used for a record identifying indicator.
H1-H9	Halt indicator.
U1-U8	External indicator.
RT	Return indicator.
******	Lookahead field (not an indicator). Lookahead can be used only with a primary or secondary file.

The whole purpose of these columns and the record identification codes in positions 21 through 41 are to be able to inform the program as to which record has been read so it can be processed uniquely compared to all other record types. For example, the program would code to test for an A in the payroll example records first shown in Figure 7-10 and if the record code is an A, the programmer would code an indictor in 19-20 so the program would turn it on to indicate that an A record was read – So also for a B and a C record.

The indicators specified in these positions are used in conjunction with the record identification codes (positions 21 through 41).

When ** is used with a primary and secondary file, RPG is able to peek ahead in the file at the next record to be processed in a file.

In RPGIV, Record identifying Indicator is in position 21 – 22.

IPDRI Positions 21-41 (Record Identification Codes)

The entries a programmer makes in positions 21 through 41 provide a means for the program to identify each record type in an input file consisting of multiple types of records. Up to three test sets can be included on one statement and the tests can be extended by using the AND/OR logical relationship column described above (columns 14-16). The best example for this for a PAREG variant is shown in Figure 7-9.

If the file contains only one record type, which is the case with most files used with System i, the identification codes can be left blank. However, a record identifying indicator entry (positions 19 and 20) and a sequence entry (positions 15 and 16) must be made. It is OK for the sequence entry to be alphabetic meaning no sequence.

Three sets of entries to be tested can be made in positions 21 through 41: Set 1 is 21 through 27; set 2 is 28 through 34; and set 3 is 35 through 41. Each set is shaped the same and each is divided into four groups. For want of better names, we will call these parts: (1) position, (2) not, (3) code part, and (4) character.

It helps to remember that we are coding at this point to test input record types so that RPG will turn on an indicator that we can use in our program to process a particular record type. We are not defining anything here. We are merely testing conditions to see if they meet a particular record type's criteria.

(1) Position refers to the specific position in a data record that we will be testing for its contents. If a record is 100 characters long, we can test in

any of 100 places for a record code that differentiates this record from others. If we use the three record types defined in Figure 6-3, we can see that there is just one record code that differentiates each record and it is located in position 1 of the record.

There are applications that build standard messages so that data can be passed from program to program and depending on the code in the message, the data with the message is formatted specifically for that message and it differentiates that message from all other messages. If an organization chose a four digit code for example for that message ID, the ID itself could be examined in this input area provided there were fewer than 99 different combinations of codes that were deployed in the application. To test for four codes, all three groups on one input statement would need to be used and the AND relationship would need to be used to examie the fourth position.

(2) Not refers to the negative of the test. If the test, for example is looking for an A in column 1, and the N was specified in the record ID group, if there was **NOT** an A in column 1, the indicator specified in columns 19 and 20 would turn on to ID the record.

(3) Code part refers to the part of the one character that is being tested. You can choose the whole character, the zone of the character or just the digit portion.

(4) Character refers to what you are looking for in that particular area of the record. In our example, if we were looking for an A record code we would test position 1 of the record with not being blank and a C in the Code part and an A in the character part so that an indicator – let's say indicator 1 in 19 and 20 would get turned on each time the program read a master record.

Table 7-11 shows which input test sets use which positions in each set.

Table 7-11 Test Set Positions

Test Set	**21-27**	**38-34**	**35-41**
Position	**21-24**	**28-31**	**35-38**
Not	**25**	**32**	**39**
Code Part	**26**	**33**	**40**
Character	**27**	**34**	**41**

Entries in these sets need not be in any particular sequence. For example, the programmer can make an entry in positions 28 through 34 without requiring an entry in positions 21 through 27. Entries for record identification codes are not necessary if input records within a file are of the same type.

A catch-all can be coded as an input specification containing no record identification code. This defines the last record type for the file, thus allowing the handling of any record types that may be in the file but have not specifically been coded in the program.

If no record identification codes are satisfied during the an input cycle, control passes to the RPG/400 exception/error handling routine.

In RPGIV, Record Identification codes are in positions 23 – 46.

I Program Described Field Description Entries

Column 42 is reserved for future activity in RPG/400 but since there is no future development of the RPG/400 compiler in RPG, it is s safe bet we do not have to learn about whatever column 42 is about. From column 43 over to the right for program described data, RPG provides space to describe the fields and their attributes as they are to be read in from files.

The data description part of RPG begins with column 43 of the input specification as described below. Just as the record identification entries were all prefixed with IPDRI, field entries for the **I**nput **P**rogram **D**escribed **F**ield **D**escriptions shall be prefixed with IPDFD:

Figure 7-7 is repeated here as Figure 7-12 for your convenience Take another look at the field definitions before you continue.

Figure 7-12 Internally (Program) Described Input for PAREG

```
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IEMPMAST AA  01
FMT                          PFromTo++DField+L1M1FrPlMnZr
0013.00 I...                    1   30EMPNO   M1
0013.01 I...                    4  23 EMPNAM
0013.02 I...                   24  282EMPRAT
0014.00 I...                   29  48 EMPCTYL1
0015.00 I...                   49  50 EMPSTAL2
0015.01 I...                   51  550EMPZIP
0016.00 ITIMCRD  AB  02
0017.00 I...                    1   30EMPNO   M1
0017.01 I...                    4   72EMPHRS
```

IPDFD Position 43 (Data Format)

The PAREG program uses two files which have two data types – character and signed decimal. Neither of these types needs to be coded in column 44 and therefore, PAREG has no entries. The allowable entries for the data format area in position 43 are as follows:

Blank The input field is in zoned decimal format or is a character field.
P The input field is in packed decimal format.
B The input field is in binary format.
L The numeric input field has a preceding (left) plus or minus sign.
R The number input field has a following (right) plus or minus sign.

The RPG programmer uses position 43 of program described input to specify the format of the data in the records in the file that are to be read. This entry has no effect on the format used for internal processing of the

input field in the program. For example, a nun-packed numeric entry will be read from disk and converted to internal packed decimal form for processing since that is how the System i likes to do its math.

In RPGIV, Data Format is in position 36.

IPDFD Position 44-51 From / To Record Positons

The PAREG program described input is shown in Figure 7-12. As you can see, it is best to think of this 8 position as two four position areas. The programmer specified the beginning position of the record in the first half and the ending position of the record in the second half. The allowable entries in these two "half areas" are as follows:

From:
Columns 44-47

1-9999	Specify a 1- to 4-digit number. This entry is for the beginning of a field (from position)

To:
Columns 48-51

1-9999	Specify a 1- to 4-digit number This entry is for the end of a field (to position).

The from and to location entry describes the location and size of each field in the input record. Most other programming languages require a beginning position and length or just a length in describing input. RPG is much easier to deal with for the novice since the field specifications also serve as a nice and neat record layout.

In positions 44 through 47 the programmer specifies the location of the field's beginning position in the record being read; and in positions 48 through 51 the programmer specifies the location of the field's end position in the record being read. To define a single-position field, just enter the same number in positions 44 through 47 as in positions 48 through 51. Numeric entries must be right-adjusted; leading zeros can and most often be omitted.

Additional Information re: from / to length

The maximum number of positions in the input record for each type of field is as follows:

# Pos.	Type of Field
30	Zoned decimal numeric (30 digits)
16	Packed numeric (30 digits)
4	Binary (9 digits)
256	Character (256 characters)
31	Numeric with leading or trailing sign (30 digits)
9999	Data structure.

In RPGIV, the From / To positions are 37 to 46.

IPDFD Position 52 Decimal Positions

The PAREG program has three different entries for the # of decimal positions. For EMPNO and EMPZIP, for example the entries are zero. This means that the field is numeric (signed decimal format) but it has no decimal places. For EMPNAM, EMPCTY, and EMPSTA, the entries are blank. This means that these fields are of character type. For EMPHRS and EMPRAT, the entries are both 2, This means that the fields are numeric (signed decimal) and that two positions of the field length is reserved for decimal positions. The allowable entries for column 52 are as follows:

Blank Character field
0-9 Number of decimal positions in numeric field.

In combination with the data format entry in position 43, this entry describes the full format of the field. This entry indicates whether the field described on this line is a character field or a numeric field. If the field is numeric, an entry must be made (0 to 9). This entry represents the number of decimal positions to be carried for the field coming in. For a

numeric field, obviously the number is limited by and cannot exceed the length of the field.

In RPGIV, Decimal positions are in positions 47 – 48.

IPDFD Position 53-58 Field Name

The Field names for the files defined in PAREG are clearly shown in Figure 7-12. The allowable entries for field name in positions 53 to 58 are as follows:

Symbolic name	Field name, data structure name, data structure Subfield name, array name, array element, PAGE, PAGE1-PAGE7, *IN, *INxx, or *IN,xx.

These positions name the fields of an input record that are used in an RPG/400 program. This name must follow the rules for RPG symbolic names.

In RPGIV, Field Name is in positions 49 – 62.

IPDFD Position 59 – 60 Control Level

The following entries in EMPMAST have control level indication specified.

```
29   48 EMPCTYL1
49   50 EMPSTAL2
```

City has an L1 indicator and State has an L2 indicator. The allowable entries in columns 59-60 for control level are as follows:

Blank This field is not a control field. Control level indicators cannot be used with full procedural files.
L1-L9 This field is a control field.

Specify the control level indication in positions 59 and 60 to indicate the fields that are used as control fields. A change in the contents of a control field causes all operations conditioned by that control level indicator and by all lower level indicators to be processed.

Sometimes it is appropriate to use two or more fields for the same control field. This is called a split control field. It is a control field that is made up of more than one field, each having the same control level indicator. The first field specified with that control level indicator is placed in the high-order position of the split control field, and the last field specified with the same control level indicator is placed in the low-order position of the split control field.

In RPGIV, Control Level indicators are specified in positions 63 – 64.

IPDFD Position 61 - 62 Matching Fields

The following asterisked entries in EMPMAST have matching fields specified.

```
0012.00 IEMPMAST AA  01
FMT                 PFromTo++DField+L1M1
**13.00 I...             1   30EMPNO   M1
0016.00 ITIMCRD  AB  02
**17.00 I...             1   30EMPNO   M1
```

This shows that the EMPNO field in EMPMAST file is set up to match the EMPNO field in the TIMCRD file. Just one field is sued for the PAREG match definition.

The allowable entries for positions 61-62, matching fields are as follws:

Blank This field is not a match field.
M1-M9 This field is a match field.

This entry is used to match the records of one file with those of another or to sequence check match fields within one file. Match fields can be specified only for fields in primary and secondary files.

To specify that you are using matching records or match fields, place an M1 through M9 code in positions 61 and 62 of the appropriate field description specification line. A maximum of nine match fields can be specified.

The match field codes M1 through M9 can be assigned in any sequence. For example, M3 can be defined on the line before M1, or M1 need not be defined at all.

When more than one match field code is used for a record, all fields can be considered as one large field. M1 or the lowest code used is the rightmost or low-order position of the field. M9 or the highest code used is the leftmost or high-order position of the field.

If match fields are specified for only a single sequential file (input, update, or combined), match fields within the file are sequence checked to assure they are in ascending or descending sequence. In this case, the MR indicator is not set on (just one file) and cannot be used in the program. An out-of-sequence record causes the RPG/400 exception/error handling routine to be given control.

In addition to sequence checking, match fields are used to match records from the primary file with those from secondary files. When all the specified match indicators in the primary match those specified for the secondary, a special indicator called MR is turned on by RPG.

In RPGIV, Matching Fields are specified in positions 65 – 66.

IPDFD Positions 63 - 64 Field Record Relation

The PAREG program does not use field record relation.

The possible entries for field record relation includes blank which means that the field is common for all record types. The entries can be any of the various indicators that are available in RPG/400 from 01 to 99, l1 to l9, MR, U1 to U8, H1 to H9, to RT.

Field record relation is a means of reducing the coding required when the same field exists in multiple record formats of the same file. Indicators are used to associate fields within a particular record type when that record type is one of several in an OR relationship. This entry reduces the number of lines that must be coded. The field with the same indicator specified in 63 - 64 as an ORed record identifying indicator gets used if the corresponding record is read..

It is a simple concept. The field described on a line is extracted (included in the input fields) from the record by the RPG/400 program only when the indicator coded in positions 63 and 64 is on or when positions 63 and 64 are blank. When positions 63 and 64 are blank, the field is common to all record types defined by the OR relationship.

In RPGIV, Field Record Relation is specified in positions 67 – 68.

IPDFD Positions 65-70 Field Indicators

The PAREG program does not use Field Indicators.

The entry can also be any of the various indicators that are available in RPG/400 from 01 to 99, MR, U1 to U8, H1 to H9, to RT. Level indicators and MR are not allowed..

Entries in positions 65 through 70 save coding in calculations for field values. By specifying an indicator in the +, -, Or 0 area of 65 – 70, each entry is examined for positive, negative or zero as it is read into the program. Field indicators are specified on the same line as the field to be

tested. Depending on the status of the field (plus, minus, zero, or blank), the appropriate indicator is set on and can be used to condition later specifications.

Positions 65 and 66 (plus) and positions 67 and 68 (minus) are valid for numeric fields only. Positions 69 and 70 can be used to test a numeric field for zeros or a character field for blanks.

In RPGIV, Field Indicators are specified in positions 69 – 74.

IPDFD Positions 71 – 74 Unused

The area in the input spec from columns 71-74 are unused for program described files.

IPDFD Positions 76 – 80 Comments

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

RPGIV Program Described Files

RPGIV has several input field specifications that do not exist in RPG/400. These are captured in the section below with the header IPIV.

IPIV Columns 31-34 Data Attributes

Positions 31-34 specify the external format for a date, time, or variable-length character, graphic, or UCS-2 field. If this entry is blank for a date or time field, then the format/separator specified for the file (with either DATFMT or TIMFMT or both) is used. If there is no external date or time format specified for the file, then an error message is issued.

For character, graphic, or UCS-2 data, the *VAR data attribute is used to specify variable-length input fields. If this entry is blank for character, graphic, or UCS-2 data, then the external format must be fixed length.

IPIV Column 35 Date/Time Separator

Position 35 specifies a separator character to be used for date/time fields. The & (ampersand) can be used to specify a blank separator. For an entry to be made in this field, an entry must also be made in RPGIV input positions 31-34 (date/time external format).

Chapter 8

The Specifics of RPG Coding – Input Structures & Constants – by Example

What is a Data Structure?

A data structure is simply a packaging of data elements as in a record. In fact, a record is a data structure. In programming for computer science applications and business alike, a data structure is a way of storing data in a computer so that it can be used efficiently. In computer science, a carefully chosen data structure will allow a more efficient algorithm to be used. In business programming careful design of data structures can provide an ease of understanding and application standardization. A well-designed data structure allows a variety of critical operations to be performed using as little resources, including programmer coding time, as well as both execution time and memory space, as possible.

In this chapter we show how to code both program-described and externally described data structures. A program described data structure is identified by a blank in position 17 of the data structure statement. The subfield specifications for a program-described data structure must immediately follow the data structure statement. An externally described data structure is identified by an E in position 17 of the data structure statement. The subfield descriptions for this are contained in an externally described file with one record format. The file merely serves as a means of grabbing the data definition.

The data contents of the file are irrelevant to the data structure. To bring the data structure definition from the external file into the program, at

compile time, the RPG/400 program uses the external name to locate and extract the external description of the data structure subfields. An external subfield name can be renamed in the program, and additional subfields can be added to an externally described data structure in the program.

In RPG, Data structures are very powerful data configurations that have a number of purposes. For example, a data structure can be used to:

1. Allow the division of a field into subfields without using the MOVE or MOVEL operations.
2. Operate on a subfield and change the contents of a subfield.
3. Redefine the same internal area more than once using different data formats.

Data Structure Record ID Entries

Data structures are defined on the input specifications in RPG/400 and in the "D" spec in RPGIV. In RPG/400, they are defined the same way records are defined. The record specification line contains the data structure statement (DS in positions 19 and 20) and the data structure name is optional. The field specification lines contain the subfield specifications for the data structure.

Though Data Structures (DS) use the Input specification, programmers must use care when arranging the structures so that they appear in the program after the normal input specifications for records. All entries describing a data structure and its subfields must appear together

In RPGIV, the data structure is defined on the new "D" type specification which is covered in Chapter **** along with any additional detail and sample code using RPG/400 Data Structures. The PAREG program that we have been working with uses no data structures. Therefore, the examples we choose will not be completely reflective of code that is found in the sample program to this point. However, they will be similar. The designator **IDSRI** is used to begin each header so it is easy to spot the entries that belong with Input, Data Structure, Record Identification.

Let's start our examination of data structures with a hypothetical example shown in Figure 8-1 that has some basis in our EMPMAST file, but as you can see this data structure has a different format than the EMPAST file and it has more fields defined. Thus, it makes a better example for us for this topic.

Figure 8-1 Sample Employee Data Structure

```
IEMPDS1    DS...   100
I...                 1   50EMPNO
I...                 6  30 EMPNAM
I...                31  60 EMPAD1
I...                61  90 EMPAD2
I...                91 110 EMPCTY
I...               111 112 EMPSTA
I...               113 1170EMPZIP
I...               118 1252EMPPAY
I...               126 1310EMPHIR hiredate
I...               126 1270EMPYR
I...               128 1290EMPMO
I...               130 1310EMPDA
```

This data structure is internal. It is named EMPDS1. It has 100 occurrences of a record layout that has 12 fields. One field, EMPHIR (Employee date of hire) is subdivided by the structure into separate YEAR, MONTH and DAY fields. See how convenient it is to redefine a structure within RPG/400.

IDSRI Positions 7-12 Data Structure Name

Positions 7 through 12 of the DS format of the Input Spec can contain the name of the data structure being defined. The name in the example in Figure 8-1 is **EMPDS1.** The data structure name is optional, and is limited to the 6 character spaces provided. A data structure name can be specified anywhere a character field can be specified. If the data structure is externally described and positions 21-30 are blank, this entry must contain the name of an externally described file.

IDSRI Positions 13-16 Reserved

Columns 13-16 of the data structure record format form of the input specification has reserved positions for future use

IDSRI Positions 17 External Description

The example in Figure 8-1 is internal so this column is blank. The External Description area in column 17 can have the following entries:

Blank Subfield definitions for this data structure follow this specification.

E Subfield definitions are described externally. Positions 7 through 12 must contain the name of an externally described file if positions 21 through 30 are blank. The file name must be limited to 6 characters.

IDSRI Positions 18 Option

The example in Figure 8-1 uses no Option entry.

The Option field has just a few entries but it enables a number of very powerful facilities of data structures – initialization, program status, and data area data structure. Initialization involves the filling of subfields with zeros or blanks; program status provides information about program operations in a handy special purpose data structure. A data area is akin to a one record disk file that is external to the program and brought in during program startup or during data area operations. Adding a data structure to a data area provides the one record with a layout of fields and field names that are available to the program.

The allowable entries include he following:

Blank This data structure is not a program status or data area data structure, and this data structure is not globally initialized.

I	Data structure initialization. All subfields in the data structure are initialized; characters to blank, numerics to zero, in the order in which they are defined, during program initialization.
S	This data structure is the program status data structure. Only one data structure can be specified as the program status data structure.
U	This is a data area data structure. The external data area (named in positions 7 through 12) is retrieved when the program starts and rewritten when the program ends. If you put blanks in positions 7 through 12, the local data area is used. It is important to note that the data area specified by the data structure is locked for the duration of the program.

IDSRI Columns 19 – 20 Record Identifying Indicator

Since the example in Figure 8-1 reflects a real data structure, positions 19-20 contain the letters "DS." In the position typically reserved for an input record identifying indicator to be specified, for a data structure record ID, the letters "DS" must be provided in columns 19 – 20.

IDSRI Positions 21 – 30 External File Name

The example in Figure 8-1 is internal so this column is blank. The external name of the database file from which the data structure definitions are pulled is specified in columns 21 – 30. Using External structures accommodates standardization and it reduces the amount of coding – especially for long data structures. The allowable entries are as follows:

Blank	The data structure subfields are defined in the program.
File name	This is the name of the file whose first record format contains the field descriptions used as the subfield descriptions for this data structure.

IDSRI Positions 31 – 43 Reserved

Columns 31 – 43 of the data structure record format form of the input specification has reserved positions for future use

IDSRI Positions 44-47 Data Structure Occurrences

The example in Figure 8-1 is set up to contain 100 records (occurrences) in memory. A simple data structure is similar to a one record memory file. Data structures however can have multiple memory records called occurrences. In columns 44-47, you specify the number of occurrences that this particular data structure should have. The allowable entries are as follws:

Blank This is not a multiple-occurrence data structure.
1-9999 The number (right-adjusted) indicating the number of occurrences of a multiple-occurrence data structure.

These positions must be blank if the data structure is the program status data structure (indicated by an S in position 18), a file information data structure (INFDS), or a data area data structure.

IDSRI Positions 48 – 51 DS Length

In the example in Figure 8-1, the length of the data structure is 131 but the compiler must calculate that by adding up the subfields. The length of a data structure can either be specified in positions 48-51 of the Data Structure Record Format or it can be calculated by the compiler. It is optional. The possible entries are as follows:

Blank Length of the data structure is either the length specified on the input field specifications if the data structure is an input field or

the highest To position specified for a subfield within the data structure if the data structure is not an input field.

1-9999 Length of the data structure.

If the length is specified, it must be right-adjusted. If this entry is not made, the length of the data structure is one of the following:

A. The length specified on the input field specifications if the data structure name is an input field.

B. The highest To position specified for a subfield within the data structure if the data structure name is not an input field.

IDSRI Positions 52 – 74 Reserved

Columns 52 – 74 of the data structure record format form of the input specification has reserved positions for future use

IDSRI Columns 75 – 80 Comments

Columns 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

I Data Structure Subfield Entries

In this section, the subfield entries required for data structures are examined in detail. See the I Data Structure Record Format header above for more information on data structures as well as Chapter **** The designator **IDSSF** is used to begin each header so it is easy to spot the entries that belong with Input, Data Structure, SubField..

IDSSF Column 7 Reserved

Column 7 of the data structure subfield form of the input specification has reserved this position for future use

IDSSF Column 8 Initialization Option

The example in Figure 8-1 is basic and it uses no initialization options.

That which the record format giveth the subfield can taketh more. Position 18 of the DS record format provides an I option to initialize the subfields to either zeroes or blanks. This option trumps that option by permitting the subfield to be initialized with a real value, rather than a zero or blank. The allowable entries for this option include the following:

Blank No subfield initialization other than that specified in record format.
I Subfield is initialized to value specified in positions 21 to 42 of this statement.

IDSSF Columns 9 – 20 Reserved

Columns 9 – 20 of the data structure subfield form of the input specification has reserved these positions for future use

IDSSF Columns 21 – 30 External Field Name

The example in Figure 8-1 is internal so this column is blank.

When an externally described data structure is coded in a program, sometimes there is a field name conflict and the subfield in the external structure must be renamed to be able to be used properly in the program. To rename a subfield in an externally described data structure, specify the external name in positions 21 through 30, and specify the name to be used in the program in positions 53 through 58. The remaining positions of the DS subfield form must be blank.

IDSSF Columns 21 – 42 (Initialization Value)

The example in Figure 8-1 uses no initialization. When a subfield is to be initialize with a specific value, specify a literal value or a named constant in these positions. If no value is specified and position 8 contains I, the subfield is initialized to zero or blanks, depending on the field type. The value may be continued on the next line. Obviously the initialization value cannot be sued with an externally described DS since the fields are mutually exclusive.

IDSSF Columns 31 – 42 Reserved

Columns 31 through 42 of the data structure subfield form of the input specification must be blank, if an external field name is specified in positions 21 to 30.

IDSSF Column 43 Internal Data Format

The subfields in the example in Figure 8-1 use only character or zoned decimal format and therefore, for all subfields described , this column is blank.

Since data fields can be defined with the DS subfield form, the programmer must specify what type of data is to be stored in each subfield of a data structure. Unlike the external data format field, the entry determines the internal format of the data. The allowable entries include:

Blank Subfield is in zoned decimal format or is character data if position 52 (decimal positions) is blank.
P Subfield is in packed decimal format.
B Subfield is in binary format.

IDSSF Column 44 – 51 Field Location

The subfields in the example in Figure 8-1 use these columns to specify the from and to positions of each subfield. The EMPHIR field uses the same are of the structure as does the EMPYR, EMPMO, and EMPDA fields combined.

In columns 44-47 you specify the "From" position and in columns 48-51 you specify the "To" position of the subfield. Both the "From" and the "To" values must be right-justified, and leading zeroes may be omitted. The allowable entries are as follows:

From:
Columns 44-47
1-9999 Specify a 1- to 4-digit number. This entry is for the beginning of a field (from position)

To:
Columns 48-51
1-9999 Specify a 1- to 4-digit number This entry is for the end of a field (to position).

IDSSF Column 52 Decimal Positions

The subfields in the example in Figure 8-1 use this position for the number of decimal places. The only field with more than zero decimal places is EMPPAY with two places defined out of the length of the field. Those subfield entries with 0 decimals are by default numeric zoned decimal format and those with no entry for decimal places are character. format.

The allowable entries for column 52 are as follows:

Blank Character field
0-9 Number of decimal positions in numeric field.

In combination with the data format entry in position 43, this entry describes the full format of the field. This entry indicates whether the field

described on this line is a character field or a numeric field. If the field is numeric, an entry must be made (0 to 9). This entry represents the number of decimal positions to be carried for the field coming in. For a numeric field, obviously the number is limited by and cannot exceed the length of the field.

IDSSF Column 53-58 Field Name

In positions 53 through 58, enter the name of the subfield that is being defined. The name can be an array name, but cannot be an array element name. Twelve meaningful subfield names are provided for your inspection if Figure 8-1.

IDSSF Columns 59 – 74 Reserved

Columns 59 through 74 of the data structure subfield form of the input specification must be blank. This space is reserved for future use.

IDSSF Columns 75 – 80 Comments

Columns 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

I Named Constant Entries

The input specification in RPG/400 is definitely overworked. It is the major utility player for a language that evolved past the logical means of supporting it without change. The major changes to the RPGIV language have addressed all of these concerns. In RPGIV, the Named Constant is defined on the new "D" type specification which is covered in Chapter **** along with some additional detail and sample code.

But, before we get to RPGIV we need to take a look at how to define constants in RPG/400 and provide them with a name. That is what this section is about. The designator **INC** is used to begin each header so it is easy to spot the entries that belong with Input, Named Constant.

The example to keep in mind as we examine the structure of the named constant variant of the INPUT specification is shown in Figure 8-2.

Figure 8-2 Continued Named Constant

```
I...   'Press Enter to con-  C...    CONTNU
I...   'tinue operation.'
```

The name of this constant is CONTNU and it is continues so that it can fit in all of the words, "Press Enter to continue operation.' The text on the first line continues to the second. The first line contains a C in position 43 and the second line is blank in position 43 meaning it is continued.

INC Columns 7 – 20 Reserved

Columns 7 through 20 of the named constant form of the input specification must be blank. This space is reserved for future use.

INC Columns 21 – 42 Constant

Columns 21-42 are reserved in the named constant form of the input specification for the contents of the named constant field which may hold a real constant or an edit word. The constant may be continued on subsequent lines by coding a hyphen as the last character. For character named constants the hyphen replaces the ending quote. A continued numeric constant must result in a valid decimal number with at most 30 digits, a maximum of 9 being to the right of the decimal point. Named constants can be declared anywhere in the input specifications.

INC Columns 43 Type / Continuation

Just as a DS entry differentiates a data structure from an input spec, a C in column 43 tells the compiler that this is a named constant. If one or more lines above this entry contain a C, then this line may be a blank meaning it if only other blanks for comments separate this from the C entry then this is a continuation of a named constant.

The allowable entries are as follows:

C Type of name is constant
Blank Constant continuation line

INC Columns 44 – 52 Reserved

Columns 44 – 52 of the named constant form of the input specification must be blank. This space is reserved for future use.

INC Columns 53 – 58 Constant Name

Place the name of the constant in columns 53 – 58. Normal RPG field naming rules apply

INC Columns 59 – 74 Reserved

Columns 59 – 74 of the named constant form of the input specification must be blank. This space is reserved for future use.

INC Columns 75 – 80 Comments

Columns 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 9

The Specifics of RPG Coding – Calculations – by Example

Mathematics and Logic

The Calculation Form is where most of the action takes place in RPG programs. All programming languages provide a vehicle for arithmetic and logical operations and RPG programmers use the calc form to get these functions accomplished. Of course, when the RPG cycle is used as in the PAREG program, the calculation form does not get the input/output action since that is handled by the cycle itself.

The calculation specifications for the external PAREG program (Figure 5-1) and internal PAREG program (Figure 5-2) that we have been decoding over the last few chapters are repeated below for your convenience as we examine them in detail in this chapter. You will first notice that in column 6, there is the "C" official specification designator to differentiate the calc format of this RPG form from all others. The statements coded on this spec form are executed in the RPG cycle during either detail time calculations (Step 7) or total time calculations (Step 3). This is the place in RPG in which the bulk of the decision making as well as the computations take place. One can expect lots of action with a stop at the "C" spec during the fixed logic cycle..

As you examine the calculation specification shown in Figure 9-1, you will notice that each statement is divided into three parts that specify the following:

1. The conditions under which the calculations are to be done: The conditioning indicators specified in positions 7 through 17 determine when and under what conditions the calculations are to be done.

2. The type of calculations to be done: The entries specified in positions 18 through 53 determine the kind of calculations to be done, specify the data (such as fields or files) upon which the operation is to be done, and they specify the field that is to contain the results of the calculation.

3. The type of tests that are to be made on the results of the operation: Indicators may be specified in positions 54 through 59 of RPG/400 specs. These are used to test the results of the calculations and the indicators that are turned on or off can affect subsequent calculations or output operations. The resulting indicator positions (54 to 59 in RPG/400) have various uses, depending on the specific operation code. For a perspective on the uses of these positions for various operations, see the individual operation codes in Chapter ****,
"Operation Codes."

All detail operations that are unconditioned are performed each cycle. All unconditioned operations that are specified at a total level will be executed unconditionally when RPG passes through that particular total cycle. Any arithmetic operation that is performed, including the compare operations, which internally subtract Factor 2 from Factor1, sets the status of the indicators specified on the right side of the calculation spec. This section is known as the resulting indicator area.

Three indicators can be specified in this area and RPG will turn one of them on during a calculation operation depending on the result. The three conditions that can be specified in RPG/400 are (1) columns 54 & 55 [greater than zero – non-blank and positive], or (2) columns 56 & 57 [less than zero --negative or minus], or (3) columns 58 & 59 [equal to each other -- or equal to zero]. Only one of these conditions will occur after an arithmetic or logic operation and thus if indictors 01, 02, and 03 are selected respectively as resulting indicators, If the results are positive, 01 is turned on; if the results are negative, 02 is turned on, and if the results are

equal or zero, 03 is turned on. The positive, negative or equal status is determined when the field in Factor 1 is compared, added, subtracted, multiplied, or divided to/by the field in Factor 2 as B. When this happens as part of RPG's magic, the other two indicators specified that are not true (not set on) for this operation are actually set off. So, just one of the three conditions will be reflected by the status of the indicators.

The process works exactly the same in RPGIV. However, since the RPG IV calculation specification is shaped differently than the RPG/400 statement, the three resulting indicators in an RPGIV calculation are reflected in columns 71-72, 73-74, and 75-76 respectively. Figure 9-1 shows how indicators 01, 02, and 03 would be specified in statement 21 of the PAREG program if we had a reason to test the results of the multiplication operation. Indicator 01 would turn on if the result was positive. If negative or zero, 01 would be turned off. Indicator 02 would turn on if the result was negative. If positive or zero, 02 would be turned off. Indicator 03 would turn on if the result was zero. If positive or negative, 03 would be turned off.

Figure 9-1 Resulting Indicators Specified in Multiply

```
678911234567892123456789312345678941234567895123456789
C*
C* RPG CALCULATION SPECIFICATION FORMS
C*
C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72 010203
```

The RPG Calculation Specification

To give you a head start to decoding the PAREG calculations, lets' take a shot at decoding the filled in entries in Figure 9-2. Of course, we have no real idea of what this program is, where it came from, who wrote it, what it does, etc. But isn't that what you will experience when you get to look at your first program written by somebody in the "shop." So, let's take a shot at decoding this program one line at a time. Let's first see what it would look like in Figure 9-3 if we typed it up nice with a source editor such as SEU.

Figure 9-1 Old-Time Filled-In Calc Spec

Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not / And / Not / And / Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal
C			START	TAG								
C				EXFMT	PROMPT							
C		N21	ACTNUM	CHAIN	CUSMST					30		
C			ADD	IFNE	'A'							
C			*IN30	ANDEQ	'1'							
C				SETON						40		
C		N21 40		GOTO	START							
C				END								
C		N21		EXFMT	RESPONSE							
C		N21 30		WRITE	CUSMST							
C		N21N30		UPDAT	CUSMST							
C		N21		GOTO	START							
C				WRITE	HEADER							
C			PRINT	TAG								
C				READ	CUSMST							45
C		N45 10		WRITE	HEADER							
C		N45		WRITE	DETAIL							
C		N45		GOTO	PRINT							
C				CLOSE	*ALL							
C				SETON						LR		

Figure 9-2 Typed Version of Written Code

```
FMT C   CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEq
0026.00 C             START     TAG
0027.00 C                       EXFMTPROMPT
0028.00 C   N21       ACTNUM    CHAINCUSMST                 30
0029.00 C             ADD       IFNE 'A'
0030.00 C             *IN30     ANDEQ'1'
0031.00 C                       SETON                       40
0032.00 C   N21 40              GOTO START
0033.00 C                       END
0034.00 C   N21                 EXFMTRESPONSE
0035.00 C   N21 30              WRITECUSMST
0036.00 C   N21N30              UPDATCUSMST
0037.00 C   N21                 GOTO START
0038.00 C                       WRITEHEADER
0039.00 C             PRINT     TAG
```

```
0040.00 C                      READ CUSMST                45
0041.00 C  N45 10              WRITEHEADER
0042.00 C  N45                 WRITEDETAIL
0043.00 C  N45                 GOTO PRINT
0044.00 C                      CLOSE*ALL
0045.00 C                      SETON                   LR
```

Now, let's examine each of these specifications in detail and decode them as best we can. After you have done your best with the limited operations knowledge that you have at this point, come back and look at the results in Table 9-4.

Table 9-4 Decoding CUSTMST Program

St#	What does it do?
26	**Provides a spot in the program to which a GOTO can branch. Sine this is in the beginning, it is probably a return point for processing or for errors.**
27	**The EXFMT sends out a screen panel named in Factor 2 as PROMPT from a display file described in file descriptions. When this operation starts it sends out the PROMPT panel. Then it waits for the user to enter something into the prompt screen. Then, the operation reads the data into the program and fills up the input fields defined for that particular display format.**
28	**It looks like indicator 21 means that the user selected an option in the program (possibly an indicator on the PROMPT panel) to print a report and if indicator 21 is not on the CUSMST file will be updated. The ACTNUM field which came in from the prompt panel is used to chain to (random read) the customer master file to pull up that account record. More than likely indicator 21 turns on as a result of some action on the screen, such as a user hitting F21 and the indicator is passed to the program along with the screen panel. Set indicator 30 on If the record does not exist in the customer master file. If the account is found, load the fields with the CUSMSTcusmst account information**
29	**The "ADD" field which was read in from the PROMPT panel is tested for the negative of a value "A" (anything but A is true) It looks like they are testing to see if the user wants to add a record if it is not found in the customer**

master.

30	**Indicator 30, (meaning the CUSMST record was not found on the CHAIN operation) is tested to see if it was turned on. If this condition is true (record not found) and the prior condition is true (not A), then the if statement evaluates as true If either are false, the If statement is false..**
31	**If both are true, indicator 40 is turned on to record this status meaning that no record was found and the use does not want to create a record.**
32	**If both are true which sets indicator 40 on, and if indicator 21 is not on, the GOTO will be executed and the program will go back to start. If 40 is not on or 21 is on, the next statement (34) will be executed. So, if the record is not found and the user did not say to add the record, then go back to start. If the user said to add the record, proceed. with the program. If indicator 21 is on, the program passes on and goes to the PRINT routine after writing headings.**
33	**This ends the action for the IF group 30-33** **What does it mean to have arrived here in the program. Indicator 40 is not on and 21 is or is not on not on. If 21 is not on, this means that either the terminal operator typed an A to say it was OK to create a new record or there was no A but the record was found. – therefore the record from CUSTMAST is OK to update. That's what we know at this point. So we write a new record if no 40 and the A was keyed or we update the record read if no 40 and the record was founf (30 not on) If 21 is on, we are ehading to the print routine but first we will print headings.**
34	**If indicator 21 is not on, 40 is not on, so Send out the response display with the EXFMT operation to get the information to update or create a data record in the customer file, then wait for a response from the screen user to get the customer fields from the panel, then load the input fields and indicator fields from the RESPONSE panel to the CUSTMST record format for update or output.**
35	**If indicator 21 is not on, and indicator 30 is on (customer master does not exist) write out a customer master record**
36	**If indicator 21 is not on, and indicator 30 is not on (customer master was found and in memory, go ahead and update the**

customer master with the information that was received in the RESPONSE display panel.

37 **If 21 is not on, go to start (26) to get more data. If 21 is on, proceed to next statement in program (38) to start the print routine**
38 **Write the headings from an externally defined print file to a printer (spooled).**
39 **Provides a spot in the program to which a GOTO can branch. Since this is in the beginning of a routine loop called PRINT, this is where the CUSMST file gets read and printed..**
40 **Read the CUSMST file from the beginning to create a report. Turn on Indicator 45 if all the records have been read from the CUSMST file.**
41 **If 45 is not on (not the end of the CUSMST file yet) and 10 is on because of an overflow condition on the printer, print the headings again**
42 **If 45 is not on (not the end of the CUSMST file yet) print out a detail line to the externally described print file record format named detail.**
43 **If 45 is not on (not the end of the CUSMST file yet) go to PRINT at line 39 to read another record and go through the print cycle again.**
44 **When the report is printed, close all the files**
45 **Set on LR so the program can end**

Now, wasn't that fun. If you stay with that little menagerie of operations as in Figure 9-2 and 9-3, until you know it cold or are fairly clued in, you will already have a jump start on learning what RPG is all about. Now, let's examine each of the calculation statements in the PAREG program starting with the four functional lines of code in Figure 9-5.

Figure 9-5 RPG Calculations for PAREG

```
        67891123456789212345678931234567894123456789512
0018.00 C*
0019.00 C* RPG CALCULATION SPECIFICATION FORMS
0020.00 C*
0021.00 C    02 MR    EMPRAT    MULT EMPHRS    EMPPAY   72
0022.00 C    02 MR    EMPPAY    ADD  CTYPAY    CTYPAY   92
0023.00 CL1           CTYPAY    ADD  STAPAY    STAPAY   92
0024.00 CL2           STAPAY    ADD  TOTPAY    TOTPAY   92
```

Statement numbers 18, 19 and 20. are comments (* in column 7) so they have no bearing on the logic of the program whatsoever. Following the comments are the detail calculations. These are the calculations without an * in column 7 and without an "L1 or L2 " in columns 7 and 8. They are shown in statement 21 and 22 in Figure 7-2. As you have learned Each time a record is read from the primary or the secondary file(s) (Steps 2 and 6) the next step in the cycle (Step 7) is for RPG to execute these detail calculations.

The first detail calculation specified in statement 21 occurs when the 02 record (TIMCRD) is read and there is a match with the payroll master record for that employee.

Statement 23 and 24 show total calculations. In the RPG cycle at Step 3, whenever there is a change in the value of the control field that is marked by L1, those calculations with an L1 such as statement 23 above are performed. This processing occurs after the last record read from the prior group of records with the same control field value (EMPCTY -city) and before the first record of a group with a new control field value is processed. In the RPG cycle, we might say that all level calculations occur "in-between time." For the PAREG program, the level 1 break occurs when the City value in the group of pay records change from say, Scranton to Wilkes-Barre.

L2 calculations work the same as L1. In this case, a change in the EMPSTA field -- state, creates an L2 control break and L2 calculations occur In the PAREG program, the only L2 calculation line is shown as statement 24 above. For the PAREG program, the level 2 break occurs when the State value in the group of pay records change from say, Alaska to Pennsylvania. Of course whenever there is a higher level such as when the state changes forcing a Level 2 break, there is also a corresponding Level 1 break forced by the L2 change. This makes lots of sense logically

for if we were processing Anchorage Alaska and the next record were Anchorage, Pennsylvania, the two of these are definitely different cities.

Calculation Specification Statement Format

Now that we have examined the calculations specification in light of the CUSTMST program snippet in 9-2 and 9-3 and the PARGE calculations in Figure 9-5, let's look at the full calculation specification in much the same way as we examined the Header, File Description, Line Counter, and the many Input variants. As you know by now, the specification form begins with the requisite "C" that must appear in position 6 to identify this line as a calculation specifications. In addition to column, 6, the calc spec has many other options that we can specify. Let's review them now. For the calculation specification we use the header designation of C to easily differentiate this form from all others.

C Columns 7-8 (Control Level)

The possible entries for the control level are as follows:

Blank	A. Operation is performed at detail calculation time for each program if conditions are met B. Operation is performed within a subroutine
L0	The operation is done at total calculation time for Each program cycle – independent of control fields
L1-L9	The operation is done when the appropriate control Break occurs at total calculation time, or when the indicator is set on.
LR	The operation is done after the last record has been Processed or after the LR indicator has been set on.
SR	The calculation operation is part of an RPG/400 subroutine. A blank entry is also valid for subroutines
AN, OR	"And" and "Or" Indicators are used to group

calculations to have more than one line condition the calculation.

The RPGIV calc specification also uses columns 7 & 8 for the control level information

PAREG and Control Levels

There are only two calculation specifications in PAREG that occur at total time. They are listed in Figure 9-6.

Figure 9-6 Total Calculations – Occur After Control Break

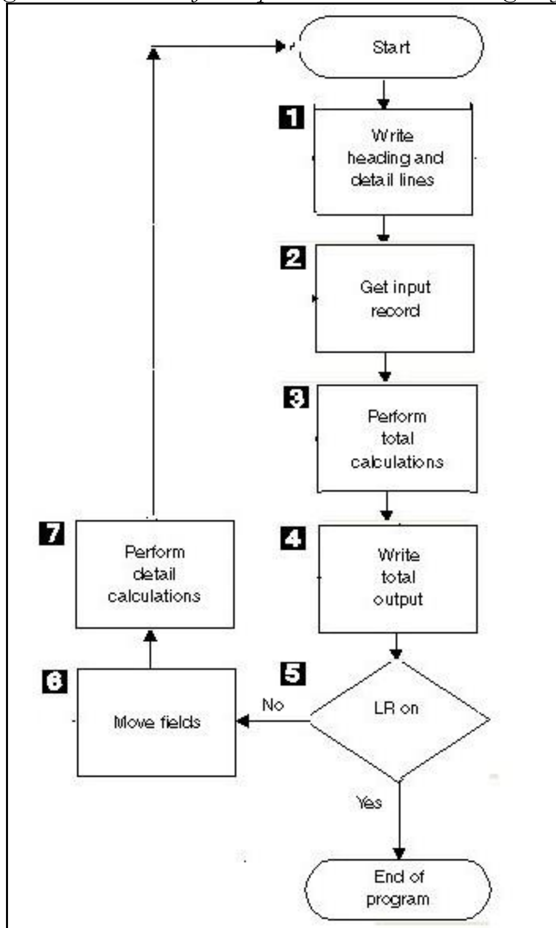
```
0023.00 CL1          CTYPAY    ADD  STAPAY     STAPAY  92
0024.00 CL2          STAPAY    ADD  TOTPAY     TOTPAY  92
```

Notice in the PAREG program calculations specifications shown completely in Figure 9-5 that the Control Level calculations are specified after the detail calculations. This is not by happenchance. Detail calculations (no L indication in column 7) come before L1 come before L2 come before L3 etc. come before LR calculations.

We must relate this to the RPG cycle since in order to have level calculations, you must be using the RPG cycle. Control level calculations occur in Step 3 of the RPG cycle that we described in Chapter 3. Though they are specified in the program after detail calculations, in fact, they occur in the cycle at step 3 whereas detail calculations occur in Step 7 right before the cycle goes back to the top for detail output..

Let’s spend just a little time to refresh your memory. Figure 9-6 is shown to help jog your RPG cycle learning from Chapter 3. The cycle starts with 1P output to provide a means to get headings on reports prior to records ever being read. PAREG dutifully produces its 1P output in accordance with the cycle. This first cycle of course, there is no detail output from data since no data has been read. So, RPG goes ahead starts the next input process in step 2 to get the next record for processing.

Figure 9-6 The 7 Major Steps to the RPG Fixed Logic Cycle



RPG however, does not make the information from the fields it just read available to the cycle until step 6, so it still has the stuff from the last record read by the time it hits step 3 in the cycle. To get the gist of total calculations, lets move away from the RPG first cycle. Let's say instead that RPG has just read in a record in which the state field EMPSTA in PAREG is different from the last state field that was read. Using the data from Figure 4-1 as an example, suppose Pennsylvania (PA) changed to Alaska (AK).

This means that the totals for Pennsylvania can be accumulated and printed. The state creates a level 2 break which always forces a Level 1 break. The L1 break is processed first. So, when the state changes, the city by definition is also different and it changes also. During this time in which the data from the last record is available though the next record is read, RPG learns of the control field change and starts the L1 calculations and then the L2 calculations as specified in Figure 9-6.

Thus, by the time the L2 calc in line 24 of Figure 9-6 is executed, the pay total for the last city processed for PA is added to the state total (STAPAY) at L1 total time. Then the L2 calc is executed which takes the STAPAY (state total for PA) and adds it to the total pay for all states (TOTPAY). After level calculations, PAREG moves to step 4 of the cycle which is total output which is covered in Chapter 10.

As a sneak peak into total output, consider that we have the city and state totals prepared to print at the end of Pennsylvania and before the record from Alaska that has been read actually has its fields available for processing. So, if we were to print a State by the state total, it would be PA, not AK because the data from the record has not yet been moved to the fields at this point of the cycle.

C -- Columns 9-17 (Indicators)

Logic type decisions are what separate computers from calculators. A logic decision in all cases begins with a test of one value against another and that test can provide three different results

1. Is Value 1 greater than Value 2 (+, HI,>)
2. Is Value 1 less than Value 2 (-, LO, <)
3. Is Value 1 equal to Value 2 (0, EQ, =)

Logic tests in typical programming are followed by branch operations. A branch operation after a logic test can alter the address of the next program instruction to be someplace in the program other than the enxt sequential instruction. If you have three different routines that you would like to fire up, based on the results of the value test (HI, LO, EQ), then there would be three different branches that could be taken by the

program. Each branch could take the program to a different routine based on the result of the test. That's program logic.

RPG calculations can be set to execute each detail cycle or they can be set to execute under certain "conditions." In most programming language, a form of an "IF" operation provides a means for the bulk of the logic decisions in a program.

RPG/400 and RPGIV have many variations of the IF statement available but there is also a mechanism that is unique to RPG called conditioning indicators. Three ANDed sets of these can be specified in RPG/400 in columns 9-17. In RPGIV, one set can be specified in columns 9-11.

Pseudo Code

In our PAREG program, we know that whenever we have just read a TIMCRD record and it matches the EMPMAST record that was just read, we would like to calculate the gross pay by multiplying the hours by the hourly pay rate. The pseudo-code logic to perform this function using an "IF" operation is shown in Figure 9-6.

Figure 9-6 Pseudo Code for PAREG

```
IF (Time Card Record is in process
and If Time Card Record Field EMPNO =
EMPNO in PAYMAST) then GROSS PAY
(SIZE 7,2) EQUALS PAYRATE TIMES HOURS
WORKED  ELSE READ NEXT RECORD.
```

That's pretty verbose pseudo code if I don't say so myself. Let's see if we can't get that pseudo code down to a more manageable size by using some things that we already know about this RPG program. We know that if there is a match between the M1 specified fields that an indicator MR is turned on for us. That means that the value of indicator MR is equal to 1. We also know that if the time card record is in process, then RPG will turn on its record identifying indicator, which we have told the compiler is Indicator 02 So if Indicator 02 is equal to 1, we are processing the correct record. We also have database fields for the rate and hours

from their respective files and we know that RPG would like any field we create on the fly to have no more than 6 characters in its name. The size for the result field of GROSS PAY must be given to the compiler since the field is being created on the fly. It is not a database field.

So, now our pseudo code can look like as teeny as the code shown in Figure 9-7.

Figure 9-7 Reduced Pseudo Code for PAREG

```
If (INMR = "1" and IN02 = '1") THEN
EMPPAY (SIZE 7,2) = EMPRAT  MULT
EMPHRS
```

This can read in English as follows: "if indicators MR and 02 are both on then create the field EMPPAY seven positions in length with two decimal places and store in it the results of the multiplication of the EMPPRAT and EMPHRS fields." The EMPRAT in the EMPMAST file is set up with two decimal places. There is also an implied else and that is if the condition is not true, go to the next statement.

The pseudo "IF" statement provides the means of specifying that we want the operation to take place only if two conditions are true. This means that the conditions are logically "ANDed" and linked since both need to be true for the operation to take place. Moving away from pseudo code to RPG, it is in columns 9 through 17 of the CALC spec that these conditions cane be specified in straight, unstructured RPG. Let's look more closely at the first calculation from statement 21 to see how this logic fits.

Figure 9-8 Detail Calculations from PAREG

```
        678911234567892123456789312345678941234567890
0021.00 C    02 MR    EMPRAT    MULT EMPHRS     EMPPAY
```

Yes, all of that pseudo code and all of that reduced pseudo code is accomplished in this simple statement. You can see that there is an 02 and an MR indicator specified in 9 through 17. Any indicator specified here is a conditioning indicator and all of the indicator sets that are specified for the multiplication operation to take place.

This example shows there is room for a third indicator in positions 16 and 17 so three indicators can be ANDed on the same statement to condition just one calculation. You may also have noticed that three positions are reserved for each indicator. MR, for example is in columns 12 to 14 right justified. So, you may be wondering what can be placed in position 12? The answer includes the option blank which means that we are conditioning with a positive of the indicator value. If, on the other hand, we placed an "N" for not in position 12, then we would be testing for the negative of the indicator or an off condition.

Moving away from PAREG for just a minute, let's show how this operation would look if we were to look for the Not condition (N) of three indicators, 02, MR, and 01.

Figure 9-9 Not Condition in Play for Calculations Spec

```
        678911234567892123456789312345678941234567890
00xx.00 C   N02NMRN01EMPRAT    MULT EMPHRS    EMPPAY
```

The RPG calculation spec example in Figure 9-9 tests for the negative of 02, MR, and 01 respectively. The negative must be true in the example, meaning that the indicator must be off for any of the sets to be true. All three sets are ANDed and therefore all need to be true for the condition to be true and for the multiplication operation to take place.

When three indicator tests are not enough for one calculation there is the option to place the AN operation in positions 8 and 9 as explained above. Also, if we would like to make these conditions OR's instead of AND's, the letters "OR" can be placed in positions 7 and 8 of a new line. The code in Figure 9-10 demonstrates this.

Figure 9-10 ANDing and ORing CALC specs

```
        67891123456789212345678931234567894123456789
00xx.00 C  N02NMRN01
00xx.00 COR 06 07 08
00xx.00 CAN 67       EMPRAT    MULT EMPHRS    EMPPAY
```

A maximum of seven AND/OR lines can be specified in one group. This reads that if 02 is not on and MR is not on and 01 is not on perform the operation, or if 06 is on and 07 is on and 08 is on and 67 is on, perform the operation. If either side of the or condition is true, the operation is performed.

The entry in positions 7 and 8 of the line immediately preceding an AND/OR line or a group of AND/OR lines determines when the calculation is to be processed (detail or total time). The entry in positions 7 and 8 on the first line of a group applies to all AND/OR lines in the group. Since the control level indicator (L1 through L9, L0, or LR) is entered for total calculations or an SR or blanks for subroutines, or a blank for detail calculations, the subsequent AN/OR line knows whether it is part of detail calculations, subroutine calculations, or total calculations and it behaves accordingly.

You can see how efficient traditional RPG is in its use of space for calculation statements. As we go through the rest of the calc statement columns, you will see that it is even more efficient than the examples that we have shown..

Taking Totals

Since we have already shown just one detail calc which calculates the gross pay, let's now examine the next three statements in PAREG since we have already discussed the notion of detail calculations and level calculations.

Figure 9-11 PAREG Total Accumulation Calculations

```
0022.00 C   02 MR   EMPPAY    ADD  CTYPAY     CTYPAY  92
0023.00 CL1         CTYPAY    ADD  STAPAY     STAPAY  92
0024.00 CL2         STAPAY    ADD  TOTPAY     TOTPAY  92
```

In statements 22 to 24 in Figure 9-11, the program is preparing totals for printing during the creation of the report. The City total is first and each time a gross pay value is calculated, it is added to the total in City Pay. When City Pay is printed on the report after a city changes, the CTYPAY field is cleared so it can accumulate again. Each time the city changes (L1), an L1 break occurs. During L1 calculations, you can see that the state total is accumulated (added to). The city total is added to the state total for later printing. When the state total is printed, so also is the city total and both are cleared and reset.. Whenever the state changes (L2) , an L2 break occurs and the state total (STAPAY) is added to TOTPAY to create a final total for the report.

Yes, it is probable that all of these defined totals and totals of all kinds (including the STAPAY and TOTPAY totals in Figure 9-11) could be created by detail time additions to the collection buckets. Instead we use an approach that uses less resources. Rather than first filling a city total collection bucket and then a state total collection bucket and dumping each bucket at each level of collection, we could have added EMPPAY to all totals at detail time. And, yes, RPG might be far easier to understand in this case if we chose to do that. However, totaling items at level time instead of detail time takes less machine instructions --- and that is a fact. If there are six records for example worth of totals in the CTYPAY bucket when it gets added in statement 23, then that would be five ADD calculations that we saved by doing it that way. So, the RPG cycle does offer IT RPG shops opportunities to conserve precious processor cycles.

Every chance we get, we try to help the learner understand that RPG is very understandable. It does so much that even pseudo-code appears complex when real RPG is exposed. Yet, we have advanced no more than column 17 of the calc spec. It's almost time to move on.

Before we move on, since the conditioning section of each calculation specification can be triggered by so many unusual options, it's time we explained how they might gain influence:

The least understood of all RPG notions is that of the "indicator." However, that is not where the conundrum is finished. In all other languages, especially COBOL, conditions, such as those specified in columns 9-17 of the calc spec, that are detected at a logic point in the program that need to be remembered later in the program are stored in things that are colloquially described as "switches." If you understand the notion of a COBOL switch, you know that the purpose and meaning of an RPG indicator is the same as that of a switch.

One of the major revisions to calculations in RPGIV is the elimination of second two indicator blocks in columns 12-17. RPGIV has room for just one indicator block from positions 9 to 11.

C – Columns 18-52 Factors and Operators

The inventors of RPG considered that in every calculation there are at least two factors, an operation, and a result, so they named the two areas in which the factors are placed "Factor 1 and "Factor 2" respectively and they aptly named the operation area as "Operation," and the area for the result of the calculations as the Result Field. The optional field length comes next followed quickly by the number of decimal positions. Within this vast area of 45 columns in RPG/400 and 59 columns in RPGIV, the meat of the calculation operations takes place. The collective format of the sub areas for both RPG/400 and RPGIV are shown in Table 9-12.

Table 9-12 Meat of the Calculation Spec – RPG/400 and RPGIV

Function	RPG/400	RPGIV
Factor 1	**18- 27**	**12-25**
Operation (plus op extender)	**28-32 (none)**	**26-35 (extender)**
Factor2	**33-42**	**36-49**
Extended Factor 2	**NA**	**36-80**
Result Field	**43-48**	**50-63**
Length	**49-51**	**64-68**
Decimal places	**52**	**69-70**
Operation Extender	**53**	**NA (26-35)**

C -- Columns 18 – 27 Factor 1

In Factor 1 you specify the name of a field or you can provide actual data (literals) or RPG/400 special words such as (*NAMVAR DEFN) on which an operation is to be done. The entry must begin in position 18. The entries that are valid for factor 1 depend on the specific operation specified in positions 28 through 32.

C -- Columns 28 – 32 Operation

Columns 28 through 32 specify the type of operation to be done using other elements on the calc form - Factor 1, Factor 2, and the Result Field entries. The operation code must begin in position 28. The program processes the operations in the order specified on the calculation specifications form.The RPG operations are examined in detail in Chapter ****.

C -- Columns 33 – 42 Factor 2

You specify the name in Factor 2 of a field or you give the actual data (literals) on which you want a particular calculation is to be done. For the file operation codes, factor 2 names a file or record format to be used in the operation. The entry must begin in position 33. Just as for Factor 1, the entries that are valid for Factor 2 depend on the specific operation code used on the CALC spec. in n positions 28 through 32.

C -- Columns 43 – 48 Result Field

You specify a result field to catch the result of operations. After an arithmetic operation for example, the result field contains the result of the calculation operation specified in positions 28 through 32

C -- Columns 49 – 51 Length

In columns 49 through 51 specify the length of the result field. This entry is optional, but can be used to define a field that is not defined elsewhere in the program or in an external database. The below entries for the length are allowed if the result field contains a field name.

1-30 Numeric field length.
1-256 Character field length.
Blank The result field is defined elsewhere.

The length entry specifies the number of positions to be reserved for the result field. The entry must be right-adjusted. The unpacked length (number of digits) must be specified for numeric fields.

If the result field is defined elsewhere in the program, no entry is required for the length. However, if the length is specified, and if the result field is defined elsewhere, the length must be the same as the previously defined length.

If half-adjustment is specified in position 53 of the calculation specifications, the entries for field length (positions 49 through 51) and decimal positions (position 52) refer to the length of the result field after half-adjustment.

C -- Columns 49 – 51 Decimal Positions

In column 52 indicate the number of positions to the right of the decimal in a numeric result field that is defined in columns 49 – 51. The allowable entries are:

Blank The result field is character data or has been defined elsewhere in the program.
0-9 Number of decimal positions in a numeric result field.

If the defined result field is numeric and it contains no decimal positions, enter a '0' (zero). For character data, this position must be blank. This position can also be left blank if the result field is numeric but was described by input or calculation specifications or in an external description. In this case, field length (positions 49 through 51) must also

be left blank. Obviously, the number of decimal positions specified cannot exceed the length of the field.

C- Column 53 (Operation Extender)

Various operations use this extender operation for different purposes. We have not examined the use of input output operation codes, such as READ as of yet, but when we do, the Operation Extender will take on even more meaning.

The possible entries for the extender field are as follows:

Blank	**No operation extension supplied.**
H	**Half adjust.**
N	**Record is read but not locked (Update files).**
P	**Pad the result field with blanks.**

An H indicates whether the contents of the result field are to be half adjusted (rounded). Half-adjusting is done by adding 5 (-5 if the field is negative) one position to the right of the last specified decimal position in the result field. The half adjust entry is allowed only with arithmetic operations.

The "P" operation will also make sense after we study the various operations that can be used in RPG. For example, the "P" entry indicates that, for CAT, SUBST, MOVEA, MOVEL, or XLATE operations, the result field is padded on the right after executing the instruction if the result field is longer than the result of the operation. Padding is done from the left for MOVE. This will make more sense in Chapter **** as we explore operations in greater detail.

RPGIV does not have a column corresponding to the operation extender. Instead in the expanded operation area, each op code that has an extender adds the extender or extenders by adding a set of parentheses to the operation and the extenders are placed in between the parentheses right next to the operation code. RPGIV operations are covered in Chapter *****.

C – Columns 54-59 (Resulting Indicators)

These six positions used to hold up to three indicators can be used, for example, to test the value of a result field after the completion of an operation, or to indicate an end-of-file, error, or provide an indication that a record was not found. Depending on the operation, the three areas that are often associated with a +, =, or – or grater, equal, or less than arithmetic result conditions can be used for other purposes. Each operation that is studied may have its own specific use for these three indicator areas.

The resulting indicator positions have different uses, depending on the operation code specified. See the individual operation codes in Chapter **** for a description of the associated resulting indicators.

Remember the following points when specifying resulting indicators: When the calculation operation is finished, before the resulting indicator conditions are set, any resulting indicators that are on are set off. Then the new resulting indicators are set.

Our program PAREG is so simple, there are no resulting indicators in play. Figure 7-1 shows an RPG/400 program with the resulting indicators specified. RPGIV resulting indicators are specified in positions 71 to 76.

C- Columns 60-80 (Comments)

Positions 60 through 80 of each RPG/400 calculation specification line can be used for comments to document the purpose of that calculation. That's 15 more positions than the File Description and Input specifications and it comes in handy often to identify the purpose of the resulting indicators that are turned on during calculations. RPGIV provides the same number of comment lines in positions 81 to 100.

Another Look at PAREG Example CALCS

Let's take another look in Figure 9-13 at the four calculation operations from PAREG to see how the factors and operations that we just examined in detail look in practice.

Figure 9-13 Closer Look at PAREG CALCS

```
  Factor 1  OP   Factor2   ResultLGTD
  8921234567893123456789412345678951 2
1 EMPRAT    MULT EMPHRS    EMPPAY  72
2 EMPPAY    ADD  CTYPAY    CTYPAY  92
3 CTYPAY    ADD  STAPAY    STAPAY  92
4 STAPAY    ADD  TOTPAY    TOTPAY  92
```

Depending on the type of operation, calculation oriented or input/output oriented, the types of values placed in Factor 1 and Factor 2 will vary. In the multiply operation (MULT) in the line labeled 1 above, the field or value placed in Factor 1 (EMPRAT) is multiplied by the field or value placed in Factor 2 (EMPHRS) to produce a value that gets loaded into the result field (EMPPAY). The three ADD operation lines labeled 2 through 4 above are fairly readable and self explanatory. The value in the field specified in Factor 1 in all cases is added to the value in the field specified in Factor 2 to produce the result that is stored in the Result field.

Notice the length of 7 for EMPPAY and the length of 9 for all the other values. RPG/400 has no single area in which field declarations are made and there is no working storage section as in COBOL in which to define independent variables. This may sound like a disadvantage of the language until you see that you can actually define new fields on the fly in calculations. When you want to take a total, for example, you invent a field name such as EMPPAY that does not exist in any database used in the program.

Since EMPPAY is not defined within a database or an input specification, it can be defined in calculations with a length and a decimal designation. If there is a number placed in a the Decimals column from 0 to 9, the field is numeric. If no decimal is specified then the field is alphabetic. The largest numeric field is 32 bytes and the largest alphabetic field that can be defined on the fly is 256 bytes. In the PAREG program section above, each of the fields were defined as numeric with 2 decimal places.

Factor 1 and / or factor two, depending on the operation used can contain a field, a numeric value or an alphabetic constant. For numeric operations, the factors both must be numeric as they are in the sample program.

There is a more simple way, a short cut form of CALCS, to specify the three calculation lines labeled 2 through 4 above. This is shown in Figure 9-14.

Figure 9-14 Short Form of Calc Operations

```
  Factor 1  OP   Factor2   ResultLGTD
  89212345678931234567894123456789512
2           ADD  EMPPAY    CTYPAY   92
3           ADD  CTYPAY    STAPAY   92
4           ADD  STAPAY    TOTPAY   92
```

In this shorter form of the ADD statement, for example, the compiler assumes that Factor 1 is the same named field as the result field.

The various operations and combinations of formats with Factor 1 and Factor 2 to support those operations are described in Chapter *********

Chapter 10

The Specifics of RPG Coding – Output – by Example

Showing the Results

For all the good describing files, reading input, and computing results does for programming, it provides no value to regular human beings unless the results are communicated in the form of a printed report or a display. In RPG, the output specification is used to provide a means of coding this communication.

O-- Output Specification Form

The Output Specification Form is where most of the coding takes place in the PAREG program, which we have been decoding for the last set of chapters. Since PAREG is a report-writing style fixed cycle program, it is understandable that the bulk of the programming is done with the output form. All of the output specifications for the PAREG program originally shown in Figure 5-1 that we are about to decode are repeated below for your convenience. The output for the PAREG program is all program described and since there is so little externally described printed output in RPG, this topic is not covered in this book. Therefore, the output for Figure 5-1 and Figure 5-2 is repeated below since it is the same. Notice that in column 6, there is the "O" designator to differentiate the format of this RPG form from all others.

Figure 8-1 RPG/400 Output Specs for PAREG

```
        67891123456789212345678931234567894123456789512345678961234567 8
0026.00 O* RPG OUTPUT SPECIFICATION FORMS
0027.00 O*
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0030.00 O                                  32 'THE DOWALLOBY COMPANY'
0031.00 O                                  55 'GROSS PAY REGISTER BY '
0032.00 O                                  60 'STATE'
0033.00 O                        UDATE Y   77
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0036.00 O                                   4 'ST'
0037.00 O                                  13 'CITY'
0038.00 O                                  27 'EMP#'
0039.00 O                                  45 'EMPLOYEE NAME'
0040.00 O                                  57 'RATE'
0041.00 O                                  67 'HOURS'
0042.00 O                                  77 'CHECK'
0043.00 O        D 1      02NMR
0044.00 O                                  46 'NO MATCHING MASTER'
0045.00 O                        EMPNO     27
0046.00 O                        EMPHRS1   67
0047.00 O        D 1      02 MR
0048.00 O                        EMPSTA     4
0049.00 O                        EMPCTY    29
0050.00 O                        EMPNO     27
0051.00 O                        EMPNAM    52
0052.00 O                        EMPRAT1   57
0053.00 O                        EMPHRS1   67
0054.00 O                        EMPPAY1   77
0055.00 O        T 22     L1
0056.00 O                                  51 'TOTAL CITY  PAY FOR'
0057.00 O                        EMPCTY    72
0058.00 O                        CTYPAY1B  77
0059.00 O        T 02     L2
0060.00 O                                  51 'TOTAL STATE PAY FOR'
0061.00 O                        EMPSTA    54
0062.00 O                        STAPAY1B  77
0063.00 O        T 2      LR
0064.00 O                        TOTPAY1   77
0065.00 O                                  50 'FINAL TOTAL PAY'
```

The first two statements (26, 27) in calculations are comments as they each have an asterisk in column 7.

Record Identification and Control Entries

Just like the Input Specification which have two major formats for both externally described and internally described files, so also does the Output form. The first format of the Input form is called the Record Identification and Control Entries. Whereas in input, we place indicators

in this area to test which record format was read, in out put we use indicators to tell the RPG compiler which record it should write.

Just as RPG permits record IDs to be tested on input using the left half of the Input form, so also does it condition and control output using the left half of the RPG output form. For the PAREG program, the left half of the RPG/400 form ends in position 31 while for those programs that use a more modern form of exception (calculation-driven) output, the area from 32 to 37 is used for what is referred to as an exception name. The notion of an exception name will be examined in Chapter ****** so for now, we end the Record Intensification and Control Entries for the PAREG program at column 31 for RPG/400. For RPGIV, the area ends in column 51

The output section of PAREG is shown in its entirety in Figure 8-1 above. For our analysis on how output is controlled let's take all of the record ID code from Figure 8-1 and place it in a separate figure (Figure 8-2) so we can get a closer look at it. To do this, we will just eliminate the right half of the output form in which control is not included. Take a look at Figure 8-1 now and see if you can spot all of the record conditioning and control statements that should be in Figure 8-2.

Hopefully you spotted them all. There are nine of them and they are presented below in Figure 8-2 for your immediate and close-up examination. The original line numbers have been included to make it easier for you to compare back to the full output form in Figure 8-1.

Figure 8-2 Record Conditioning and Control Output for PAREG

```
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0043.00 O        D 1      02NMR
0047.00 O        D 1      02 MR
0055.00 O        T 22     L1
0059.00 O        T 02     L2
0063.00 O        T 2      LR
```

OPDRI -- Specification Columns 7-14

Immediately following the O, since we have designated the printer file to be a program described output file, we get to do lots more describing column by column. The first thing you notice is that in statement 28 above, the file name (7-14) is QPRINT. This is the same name as defined in File Descriptions. Because this is a program described file for the printer, we use the file name as described in File Descriptions instead of the record format name as we did for input.

Though RPG programmers sometimes define their own print files to the System i5 to provide special formatting for certain reports such as invoices and statements, many continue to use any of the IBM default printer files such as QPRINT. (See Appendix ***** to see how to create your own printer file). The key thing in output is that when you begin to code the Record Identification and Control format of the output specification, if you have just one print file to which to refer, you must use the same name to reference the print file you have defined in the F specification. In this case the name is QPRINT.

Printing to Multiple Printers

Of course, RPG/400 programs can have multiple printers defined for output. For this, additional F specs would be used with different file names. To assure that the correct output goes to the correct printer, when there are multiple printers, the programmer uses the File Name area from 7 to 14 of the O spec to designate the file that is related to the printer to which output is to be directed.

In RPGIV, the File name is expanded to 10 positions from 7 to 16.

OPDRI Columns 14-16 (Logical Relationship)

Statements 29 and 35 in the output specifications of program PAREG make use of this logical relationship area. Looking above just a bit, you may recall that the File Name was within positions 7 and 14 of the O spec. Yet, the logical relationship is taking up one of those columns. The only reason that is OK is that the File Name does not get repeated on a logical relationship continuance line.

You must decode the Logical Relationship column in statements 29 and 35 and recognize that this continues the conditional testing as to whether the output record will be written. If either set of conditions exist (indicators 1P or OF as specified on lines 28 and 29) for example, that means that the output line will be printed.

We will be examining the specific conditions (1P or OF or other indicators) under which lines print below, but for this lesson it is important to gain that the logical relationship column of a second and possibly even a third line of Record ID and Control Entries broadens the conditions under which a line will print or a record will be written. If instead of the "OR" relationship which we show, the actual relationship were AND, the logical relationship column again can serve to expand the conditions under which an output record will be written. For the AND condition, you would place the letters AND on the second and subsequent Record ID and Control Entries. A modified sample of lines 28 and 29, using an "AND" linkage is shown below:

```
0028.00 OQPRINT  H  206   1P
0029.00 O       AND 206   OF
```

Now that you know how to "AND" out put conditions together, it is helpful to know that RPG provides a natural way to provide "ANDing." Without any words you can see the "ANDed" conditions below:

```
0028.00 OQPRINT  H  206   1P OF
```

This line is conditioned by the 1P indicator and the OF indicator, meaning it should print in the first output cycle before input and it should print whenever the OF indicator is turned on as printing fills up the prior page. We will be examining those columns very shortly… and by the way, don't run away yet, because the coding we did above for AND, though syntactically correct, is illogical.

For RPGIV, the logical relationship is specified in columns 16-19.

OPDRI — Column 15 (Type)

Though seemingly innocuous, the infamous Output Type column has created as much frustration for procedural programmers as the notion of L1 and L2. The idea of conditioning indicators for output lines is very logical. In the big picture, however, it takes a good knowledge of the RPG cycle in order to know just when those conditioning indicators might be on or off. Since we have made it through the CALC spec, it may help to know that the notion of total time output is really not much different than the notion of level calculations (columns 7 and 8) that are used for calculations to occur in between control breaks.

In fact a look at the cycle would reveal that as soon as Control Level calculations are performed, the RPG cycle moves on a mythical journey to something called total output time so that it can print what it has accumulated.. For those watching at home, total output time is another of these in-between time cycle notions that occur in between control break changes.

To be specific, the RPG cycle saga continues right here from calculations since this "in between time" is really the part of the cycle from step 2 in which a record gets read until step 6 in which RPG makes the data from that record available. During this time totals for the prior group can be taken in Cycle step 3 and then in Cycle step 4, the totals that were accumulated can be printed with the heading information from the prior record, not the record just read. See Figure 4-1 for verification. Thus, the

total for Pennsylvania that prints right before the detail record for Alaska gets processed, can actually print PA next to the total since PA has not yet been replaced by AK – even though the AK record is in. AK does not replace PA in memory until step 6 of the RPG cycle.

The coding for column 15 is self revealing and is as follows:

H Heading records in a printed report (occur only when the first Report page is to be printed or when the printer has passed the last line on the prior page – a condition called "overflow" or printer overflow. This is relevant only with printers.

Heading records usually contain constant identifying information for reports such as titles, column headings, page numbers, and date. What typically differentiates these records from pure detail records is that they require no variable input. The data is typically coded as constants or reserved words. RPG can print headings even before it has gone its first input cycle.

Heading records are printed during the detail print cycle. There is no structural difference in RPG between H and D specifications other than the intention of the programmer.

D Detail records usually contain data that comes directly from the input record or that is the result of calculationsprocessed at detail time in the RPG cycle. For example, the field in our program named EMPPAY is created at detail calculation time within the RPG cycle.

T Total records usually contain data that is the end result of specific calculations on several detail records. In the PAREG program, the City, State and Final totals fir this mold

E Exception records are not needed for the PAREG program. These are lines of output that will print when triggered by a specific calculation operation for output known as EXCPT. EXCPT is described in detail in Chapter *********

They type specification in RPGIV is located in column 17.

OPDRI – Column 16 (Fetch Overflow / Release)

PAREG does not use the Fetch Overflow / Release facility. There are a number of mythical notions that many who have toiled learning the RPG cycle have wrestled with. Fetch Overflow is one of them. In a nutshell, because RPG records can be written at detail, total or exception output time, sometimes the wrong cycle is in play to print the overflow headings. Fetch overflow (F) option is a way around this problem. It has enough of its own issues that if you learn that you need Fetch overflow in one of your programs, you will already be in graduate level RPG. We do not need Fetch overflow in the PAREG program.

We do not need the RELEASE facility or the "R" code in column 16 in this PAREG program either. The "R" for release comes about in programs that control their own access to databases (non cycle programs) and after they have made a request for input and a record is inside the RPG program staged for update, circumstances in the program may alter the requirement to actually update the database record. So, for those using the RPG cycle, the R code in column 16 will release the exclusive update lock on the record and make that particular record available to other programs.

Fetch Overflow and Release options are provided in RPG IV column 18.

OPDRI – Columns 17-22 (Space and Skip)

Figure 8-2 is repeated below so that we can examine the Space and Skip entries used in PAREG

Figure 8-2 Record Conditioning and Control Output for PAREG

```
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0043.00 O        D 1      02NMR
0047.00 O        D 1      02 MR
0055.00 O        T 22     L1
0059.00 O        T 02     L2
0063.00 O        T 2      LR
```

Line 28 & 29 skips to line 6 of the form and prints the 1P 1st line of headings or overflow 1st line headings which in PAREG is the report title. It then spaces 2 lines after printing the report title. Lines 34 and 35 print The report column headings and then space three lines to begin printing the detail lines. The detail lines (43 and 47) print when data is read after each line the printer spaces one to the next line and awaits printing. In line 55, the City total spaces 2 lines and then prints the City total. It hen spaces two lines. In line 59, the printer spaces 0 lines (city already moved it down two lines) and then it prints the State total followed spacing of two lines. In line 63, for the final total, the printer spaces another two lines for big separation between the final total and the last state total and then the line prints the final total information.

Powerful Report Writing

Only a programmer who has never tried to perfect the look of a spiffy aged trial balance report or a sales report with five dimensions of totals or a properly formatted GL financial statement would even consider pooh poohing the raw power in RPG that intrinsically enables report formatting. Yet, they are out there and some would choose to double or triple the number of statements in a program rather than use RPG's (that's Report Program Generator folks) innate reporting facilities. If I were to correct myself on this statement, I would change the word "use" to learn. Hey, you are struggling through the cycle and soon you will understand that the cycle is remarkable once you have made that "4GL" reporting investment in your career.

Spacing & Skipping

Let's envision a physical printer. In other words, a physical printer would be one that actually is forced to print on paper and not to PDF. Spooling has separated most of us in the tech community from the care and feeding of printers. Yet, they have their needs. Let's not look at HP's or Xerox's big lasers now or we won't get the right picture and we will miss the learning opportunity. Let's envision a line printer of any name. It can print one line at a time. Coincidentally, the RPG cycle prints one detail line at a time.

Now, let's go back even ten years from this. You can't get the notion of spacing and skipping without this. The big behemoth printers were capable of printing upwards o 4,000 lines per minute mechanically by impact. Such printers still exist today and their owners do not want to get rid of them though maintenance is now quite prohibitive.

These printers were equipped with a paper tape which controlled a skipping carriage. When a program sent out a skip command it was to one of 12 channels in the paper tape. So, a high speed skip to channel 6 might bypass 30 lines or more on a preprinted form and go ahead and print the needed total at the bottom of the form on the correct line every time. These little carriage tapes were Mickey-Mouse to computer operations so printer companies such as IBM made the little tape electronic and then gave the programs the control to send the printer to an exact line on a page – no matter how many lines had to be skipped. Almost instantaneously a printer could go from line 5 to line 35 and print the next line of the form. It worked just as well as the carriage control printers but now, instead of skipping to a channel, the programs were able to skip to a specific line number. It was a great advancement in technology and the RPG language matched the ability of the system by providing skip to line facilities instead of carriage control slots.

Yet, sometime, no matter where you were on a form or a report printout, you would need to skip one more line to print a total or such. Thus, the notion of line spacing persisted even after the skipping technology advancement. Whether the following is true or not does not matter sine it is a constraint regardless. RPG was devised so that the compiler would support no more than three lines of spacing before or after a print line.

IBMers told me over the years that this was a printer issue, not an RPG issue. I think it was neither. I think now that it was IBM believing that a programmer ought to be using skipping if they have to leave three blank lines on a report. So, IBM would not permit more than 3 lines to be spaced before and 3 lines to be spaced after each of those Record Identification and Control Entries. For printer control, "them" are still the rules in RPG today.

The column specifications in RPG and RPGIV today that provide spacing and skipping before and after are shown in Table 8-3.

Table 8-3 Printer Spacing and Skipping

Printer Function	RPG/400	RPGIV
Spaces before	**17**	**40-42**
Spaces after	**18**	**43-45**
Skip to line before	**19, 20**	**46-48**
Skip to line after	**21, 22**	**49-51**

If a space/skip entry is left blank, the particular function with the blank entry (such as space before or space after) does not occur. If entries are made in position 17 (space before) or in positions 19 through 22 (skip before and skip after) and no entry is made in position 18 (space after), no space occurs after printing.

OPDRI – Columns 23-31 (Output Indicators)

The same notion of conditioning indicators applies to output as it does to calculations. In calculations, for example, we learned that there were three distinct slots on one calculation specification in which the programmer could specify three indicators to condition operations and all of those indicators were automatically involved in an "AND" condition. They all had to be on for the line to be executed.

Positions 23 to 31 of output represent the same notion. Three indicators are permitted – that's two positions each or six positions of the nine. Just

like CALCS, each one of the indicators can be negated with the "not" modifier by simply placing an N in position 23, 26, or 29 as long as an indicator was specified in the two right adjacent columns.

To show you what this looks like let's take a peek at statements 43 and 47 of PAREG in Figure 8-4.

Figure 8-4 Detail Record Conditioning

```
0043.00 O         D 1      02NMR
0047.00 O         D 1      02 MR
```

These two Record Identification and Control Entries each use two of the natural "ANDed" slots for indicators to condition the output. Both use indicator 02 and indicator MR, meaning a time card record was read and there is a match. Well, at least that is what statement 47 does. Statement 43 tests to see if the time card just read does not have a matching master (PAYMAST). Of course that means that the 02NMR is clearly an error condition. It is fair to ask, as we decode this section, what are we asking the program to do when this condition occurs?

To know the answer to this, we must also look at the field codes (which we have not yet explained). However, we can rough up an explanation with what is obvious. The print line and the error conditioning (02 NMR) are shown immediately below:

Figure 8-5 Error Conditioned Output

```
0043.00 O…   D 1… 02NMR
0044.00 O…                    46  'NO MATCHING MASTER'
0045.00 O…              EMPNO     27
0046.00 O…              EMPHRS1   67
```

Right after the detail (D) indicator there is a blank and then in position 17 of Line 43, there is a "1." This tells the printer to space one before printing an error message if indicator 02 (a time card) is on and indicator MR is not on (NMR). If both are true then this is a non match with PAYMAST. So, after executing a one space before, this code tells the RPG compiler to go ahead and print the employee number and the

number of hours next to an error message of "NO MATCHING MASTER," that ends in printer position 46 of the report. Thus, the RPG cycle MR technique as coded in this program with 02 and NMR permits the program to identify a condition in which we have a time card with a missing master. We could do the same type of check if we have a PAYMAST record with a missing time card. This condition would be made known by an indicator 01 on condition along with the NMR. However, we have not coded that in this example.

The output field conditioning indicators for RPGIV have the same meaning and they live in columns 21 to 29 of the RPGIV calc spec respectively.

Field Description and Control Entries

Using the Record Identification and Control Entries as we have above, the objective is to specify the conditions under which a print line is produced by the program. Via the conditioning indicators that we have used in this program, 1P, OF, 01, 02, and MR, we have conditioned lines to print in a sequence that can produce the output report that we defined first in Figure 4-1.

Before we discuss the Field description part of the Field Description and Control Entries form, let us first look at the control entries. Just as we conditioned print lines at the record level, the Control entries portion of this form permits us to condition specific fields to print when and if a line is printed. Theoretically a line can be enabled to print at the record level and because no fields are conditioned to print, a blank line may be produced

OPDFD- Columns 7 through 22 Reserved

No entries are permitted in the RPG/400 Field Description Form from column 7 to column 22.

OPDFD- Columns 23–31 Output Indicators

The conditioning output indicators occupy the same relative space in this Field Form as they do in the Record Form – 23 to 31 and they work the same way. If all of the indicator statuses that are tested on a field line create a true condition, then the field will be printed. Of course, there are no such controls needed in the PAREG program since no fields need to be conditioned for printing. Each line that is conditioned in the PAREG program prints all fields and literals that are defined for the print line in the end positions noted on this form.

RPGIV's output indicators are provided for in columns 21 to 29.

The variable output fields printed on the report defined in Figure 5-1 are shown in Figure 8-6 twice with their original line numbers. There are lots of lines missing in this example. As you may recall from the full PAREG program in Figure 5-1, the end positions on the print line to print these fields are often the same because they in fact print on different lines – Heading, detail, Level 1, Level 2, and LR.

Figure 8-6 Output Fields for PAREG

```
        67891123456789212345678931234567894123
0033.00 O                        UDATE Y   77
0046.00 O                        EMPHRS1   67
0052.00 O                        EMPRAT1   57
0053.00 O                        EMPHRS1   67
0054.00 O                        EMPPAY1   77
0058.00 O                        CTYPAY1B  77
0062.00 O                        STAPAY1B  77
0064.00 O                        TOTPAY1   77

0033.00 O                45 46N47UDATE Y   77
0046.00 O               NMRN1PNOFEMPHRS1   67
0052.00 O                   31   EMPRAT1   57
0053.00 O                  n29   EMPHRS1   67
0054.00 O                01      EMPPAY1   77
0058.00 O                      36CTYPAY1B  77
0062.00 O                     N56STAPAY1B  77
0064.00 O               N83      TOTPAY1   77
```

There is one difference as you can see. Some of the fields in the repeated group have numbers next to them to the left. These numbers are in columns 23 to 31 and are field conditioning indicators. For the PAREG program, they are absolutely meaningless but they sure give you an idea of how to code field indicators. The UDATE field for example only prints in position 77 if indicator 45 and 46 are on and 47 is off. EMPHRS only prints in 67 if MR, 1P, and OF are all off. TOTPAY prints in 77 only if 83 is off.. As you can surmise, field indicators can create holes in output lines for information that should not be printed. You can also specify two different fields to print in the same position on the same line and before printing assure that just one of the printing conditions is true.

OPDFD-- Columns 32-37 Field Name

When coding output for PAREG, we placed the names of the fields that we wanted to print in positions 32 through 37. In Figure 8-6, you can see not only the field names and the end positions but also the editing codes which will soon be discussed. Because other elements besides fields are also specified in columns 32-37, let's a get a full appreciation for all the entries and their meanings:

- ✓ **Field name**
- ✓ **Blanks** if a constant is specified in positions 45 through 70
- ✓ **Table name**, array name, or array element
- ✓ **Named constant**
- ✓ **RPG/400** reserved words such as PAGE, PAGE1 through PAGE7, \PLACE, UDATE, \DATE, UDAY, \DAY, UMONTH, \MONTH, UYEAR, \YEAR, \IN, \INxx, or \IN,xx
- ✓ **Data structure name** or data structure subfield name.

Field Names, Blanks, Tables and Arrays

Tables and Arrays are described in Chapter *****. To be used for output, the field names used must be defined in the program, either explicitly or implicitly from being within an externally described file.

You should not enter a field name if a constant or edit word is used in positions 45 through 70. The end positions represent an either / or scenario. If a field name is entered in positions 32 through 37, positions 7 through 22 must be also be blank.

The RPGIV field name is coded in positions 30 – 33 and the field name can be indented (leading blanks).

OPDFD Column 38 Edit Codes

The PAREG program takes advantage of another very powerful facility in RPG known as Edit Codes. These codes get specified on a field line in column 38 of the output specification. A snippet from the PAREG program showing all of the lines that have edit codes is shown in Figure 10-7.

Figure 10-7 Output Fields for PAREG

```
        67891123456789212345678931234567894123
0033.00 O                         UDATE Y   77
0046.00 O                         EMPHRS1   67
0052.00 O                         EMPRAT1   57
0053.00 O                         EMPHRS1   67
0054.00 O                         EMPPAY1   77
0058.00 O                         CTYPAY1B  77
0062.00 O                         STAPAY1B  77
0064.00 O                         TOTPAY1   77
```

As you can see by walking down column 38, we were not very picky in our use of edit codes. In line 33, for example, we use the reserve word UDATE to print the system data along with a special Y edit code that makes sure the slashes are placed properly. The rest of the fields above from PAREG use the "1' edit code.

Table 10-7 shows some of the fields from the report in Figure 4-1 and how the edit code has made the printed data look much better

Table 10-7 Output Editing of PAREG Fields

Field Represents	No Edit	With Edit
UDATE	022106	2/21/06
RATE	780	7.80
HOURS	3500	35.00
CHECK	27300	273.00
TOTAL CITY PAY FOR WILKES-BARRE	58900	589.00
TOTAL CITY PAY FOR SCRANTON	114475	1,144.75
FINAL TOTAL PAY	289360	2,893.60

We picked these two (Y and 1) edit codes for PAREG because that's all we needed. But, they came from a very wide and diverse barrel. Let's take a look at all of the edit codes in the barrel in Figure 10-8 as well as what they actually mean. Edit codes do so much work with so little work involved that it is best to best explain it all with combinations of minus signs and CR signs and slashes and dashes. So we borrowed the table from IBM's AS/400 Reference Manual and it is shown in Figure 10-8 just to show you how much can really be stashed in this one column.

In RPGIV, the Edit Code place is column 44.

OPDFD —Column 39 Blank After

PAREG uses the notion of blank after for two output fields as follows:

```
0058.00 O...                          CTYPAY1B  77
0062.00 O...                          STAPAY1B  77
```

The total for city pay is collected by adding the individual's gross pay to the city pay total for each time card record read that also has a match (02 MR). When the city total is printed, the next logical step would be to clear out the total accumulator so that the prior city's total does not get mixed in with the new city's total. So also for the state total when it is printed. In typical programming languages, a separate operation or small routine

would be invoked to reset the city accumulator. In RPG it can be done with one column called blank. This very handy tool shows the power of RPG for report writing. And it certainly saves programmer coding. Immediately after RPG prints CTYPAY and STAPAY fields, it clears them (blanks them out) to prepare the fields for collecting the next city or state's total pay.

Figure 10-8 Complete RPG Edit Code Table and Meanings

				Entry in Column 21 of Control Specification			
Edit Code	Commas	Decimal Point	Sign for Negative Balance	D or Blank	I	J	Zero Suppress
1	Yes	Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	Yes
2	Yes	Yes	No Sign	Blanks	Blanks	Blanks	Yes
3		Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	Yes
4		Yes	No Sign	Blanks	Blanks	Blanks	Yes
5-9[1]							
A	Yes	Yes	CR	.00 or 0	,00 or 0	0,00 or 0	Yes
B	Yes	Yes	CR	Blanks	Blanks	Blanks	Yes
C		Yes	CR	.00 or 0	,00 or 0	0,00 or 0	Yes
D		Yes	CR	Blanks	Blanks	Blanks	Yes
J	Yes	Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
K	Yes	Yes	- (minus)	Blanks	Blanks	Blanks	Yes
L		Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
M		Yes	- (minus)	Blanks	Blanks	Blanks	Yes
N	Yes	Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
O	Yes	Yes	- (floating minus)	Blanks	Blanks	Blanks	Yes
P		Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
Q		Yes	- (floating minus)	Blanks	Blanks	Blanks	Yes
X[2]							Yes
Y[3]							Yes
Z[4]							Yes

[1] These are the user-defined edit codes.

The entries for column 39 are as follows:

Blank The field is not cleared (reset).
B The field is reset to blank or zero after the output operation

In RPGIV, the Blank after code is placed in column 45.

OPDFD - Columns 40-43 End Position

Figure 10-1 shows all the output and Figure 10-9 sets up a learning sample from PAREG to show the end positions in header (literal) output from the beginning of the output section in Figure 10-1. In Figure 10-1, both literals and variables for the PAREG program are coded along with their end positions, which do not overlap in the PAREG program..

Fields in output can be specified in any order because the sequence in which they appear on the output records is determined by the end position entry in columns 40 through 43. If fields overlap, the last field specified is the only field completely written. In other words, RPG moves all the fields you specify into the output record prior to printing it. As it moves them in, it builds the print line in the sequence that the fields are specified taking care to end each field as it is placed in the print buffer in the position defined in 40-43 as the end position. The last field specified may very well overlap another field. In this case, RPG dutifully places the full contents of that field in the buffer and overlays whatever value may have been there from a field specified in a lower numbered statement.

The PAREG program has no fields that overlap so this is not an issue in our sample program. When edit codes are used, the programmer must leave room to the left on the print line to assure that the fully edited field can fit without creating an overlay issue with the low order positions of the field to the left.

The entries that can be specified in the END position area from 40 – 43 are as follows:

1-n	Numeric value for print line end position
K1-K8	Length of format name for WORKSTN file RPG provides a facility to program describe format names in the RPG II style.
+nnn	Number of spaces to leave between last field specified Sometimes programmers do not want to precisely calculate the exact end position for a printed field or constant. RPG provides the “+” option to leave a certain amount of space in between fields saving the

programmer the work of figuring out specifically where everything ends The sign must be in position 40.

-nnn Negative of above

nnn Same position as last

Let's take a look at the first section of output in the PAREG program shown in Figure 10-9 for a real example:

Figure 10-9 End Positions

```
        6789112345678921234567893123456789412345678951234567896123456789
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0030.00 O                                  32 'THE DOWALLOBY COMPANY'
0031.00 O                                  55 'GROSS PAY REGISTER BY '
0032.00 O                                  60 'STATE'
0033.00 O                        UDATE Y   77
```

Positions 40 through 43 define the end position of a field or constant on the output. Lines 30 to 32 specifies constants and their end positions on the print line while line 33 specifies the reserved word UDATE which tells the RPG compiler to access the system date and print it ending in position 77.

Valid entries for end positions are blanks, +nnn, -nnn, and nnnn. All entries in these positions must end in position 43. Enter the position of the rightmost character of the field or constant. The end position must not exceed the record length for the file.

The plus (+) and minus (-) additions are a tool to help programmers design simple forms without having to use detailed printer spacing charts. The +nnn or -nnn entry specifies the placement of the field or constant relative to the end position of the previous field. The sign must be in position 40. The number (nnn) must be right-adjusted, but leading zeros are not required. To calculate the end position, use these formulas:

EP = PEP +nnn + FL
EP = PEP -nnn + FL

EP is the calculated end position. PEP is the previous end position. For the first field specification in the record, PEP is equal to zero. FL is the

length of the field after editing, or the length of the constant specified in this specification. The use of +nnn is equivalent to placing nnn positions between the fields. A -nnn causes an overlap of the fields by nnn positions.

For example, if the previous end position (PEP) is 6, the number of positions to be placed between the fields (nnn) is 5, and the field length (FL) is 10, the end position (EP) equals 21.

In RPGIV, the end position is specified in columns 47-51.

OPDFD - Column 44 (Data Format)

The PAREG program does not need this column to get its job done. This area is not needed for printed reports. The entries are for disk files and are as follows:

Blank	The field is written as zoned decimal numeric or character or a constant.
P	The field is written in packed decimal format.
B	The field is written in binary format.
L	The numeric output field is written with a preceding (left) plus or minus sign.
R	The numeric output field is written with a following (right) plus or minus sign.

This position must be blank if editing is specified. In RPGIV, the data format code is specified in columns 52.

OPDFD - Columns 45-70 Constant or Edit Word

Let's look at Figure 10-10 for some of the constant (literal) data from PAREG and two edit word examples in statement 99.03 and 100 that are not in PAREG. The extra code shows a seven position field being edited with an edit word. In 99.03, the zeroes are replaced by a floating dollar

sign that prints to the left of the high order non-zero digit. In 100, the same field is edited with a fixed place dollar sign.

Figure 10-10 Constant & Edit Word Examples from PAREG etc.

```
        678911234567892123456789312345678941234567895123456789
0034.00 OQPRINT   H   3      1P
0035.00 O        OR   3      OF
0036.00 O                                          4 'ST'
0037.00 O                                         13 'CITY'
0038.00 O                                         27 'EMP#'
0039.00 O                                         45 'EMPLOYEE NAME'
0040.00 O                                         57 'RATE'
0041.00 O                                         67 'HOURS'
0042.00 O                                         77 'CHECK'
0099.00 O*
0099.01 O*  Floating dollar sign followed by fixed dollar sign
0099.02 O+
0099.03 O                          NUM7           90 '      $0.  '
0100.00 O                          NUM7           90 '$      0.  '
```

As we decode the PAREG output specifications, we quickly see that the only use that the PAREG program makes of columns Positions 45 through 70 is for program literals which are also called constants since they do not change. Statements 36 to 42 of the program as shown above represents the headings for the columns of the report and this "line" or record is formatted by its end positions as specified in 40 to 43. In position 45 in the referenced PAREG statements, there is a required quote and immediately following the literal is a second single quote to signify the end of the literal. When the line prints each of the described literals will print ending in the positions specified next to them in positions 40 to 43.

Coding literals is one of the easiest things in RPG and it makes the language very powerful in being able to provide descriptive report titles and column headings as noted above.

Figure 10-11 Literals Used for Totals

```
        678911234567892123456789312345678941234567895123456789
0055.00 O        T 22     L1
0056.00 O                                   51 'TOTAL CITY  PAY FOR'
0057.00 O                         EMPCTY    72
0058.00 O                         CTYPAY1B  77
0059.00 O        T 02     L2
0060.00 O                                   51 'TOTAL STATE PAY FOR'
0061.00 O                         EMPSTA    54
0062.00 O                         STAPAY1B  77
0063.00 O        T 2      LR
0064.00 O                         TOTPAY1   77
0065.00 O                                   50 'FINAL TOTAL PAY'
```

Literals are not always used for headings, however. As you can see in the output snippet from PAREG in Figure 10-11, literals are marked at the appropriate spot of output to provide a caption for the intermediate (city and state) and the final totals. The literal prints to the left of the fields in this case as shown in Figure 10-12 below:

Figure 10-12 Literals Used to Mark Totals

```
TOTAL CITY  PAY FOR Newark            225.75

TOTAL STATE PAY FOR NJ                225.75

FINAL TOTAL PAY                     2,893.60
```

Columns 45 to 70 can be used for other functions, though our sample program PAREG does not deploy any of those functions. For example it can be used to specify an RPG II style screen format name or an edit word as shown in Figure 10-10 w. See Chapter **** for a discussion of screen formats.

RPG/400 Edit Words

Though PAREG does not used Edit words, there is no better place to describe this facility than right here. If you have editing requirements that cannot be met by using the edit codes described above, you can use an edit word or named constant. An edit word allows you to directly specify:

- ✓ Blank spaces
- ✓ Commas and decimal points, and their position
- ✓ Suppression of unwanted zeros
- ✓ Leading asterisks
- ✓ currency symbol, and its position
- ✓ Constant characters
- ✓ Negative sign, or CR, as a negative indicator.

Describing Edit words can make up lengthy book chapter in itself. Yet, a whole chapter on edit words simply is not worth your time. So, we will explain the notion and then show you some examples. As many things in RPG, it is good to know they are there in case you ever need them for

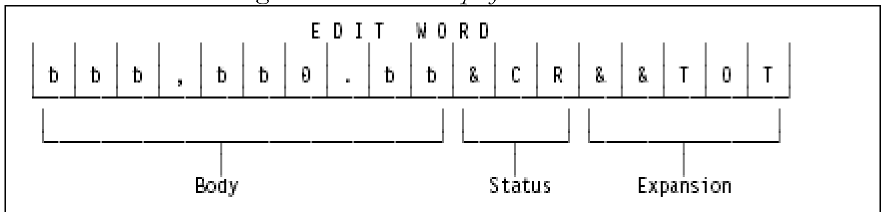
coding or decoding but when you move from learning to having reference needs, IBM's free RPG manuals will be your best tool.

As you can see in the 26 spaces space provided on the output spec, RPG edit words can be up to 24 characters long and must be enclosed by apostrophes, unless of course it is a named constant which is covered in Chapter *********.

What are the parts of an edit word?

An edit word consists of three parts: the body, the status, and the expansion. The following shows the three parts of an edit word:

Figure 10-13 Make Up of an Edit Word



Body - Status - Expansion

The body is the space for the digits transferred from the source data field to the output record. The body begins at the leftmost position of the edit word. The number of blanks (plus one zero or an asterisk) in the edit word body must be equal to or greater than the number of digits of the source data field to be edited. The body ends with the rightmost character that can be replaced by a digit.

The status defines a space to allow for a negative indicator, either the two letters CR or a minus sign (-). The negative indicator specified is output only if the source data is negative. All characters in the edit word between the last replaceable character (blank, zero suppression character) and the negative indicator are also output with the negative indicator only if the source data is negative; if the source data is
positive, these status positions are replaced by blanks. Edit words without the CR or - indicators have no status positions.

The status must be entered after the last blank in the edit word. If more than one CR follows the last blank, only the first CR is treated as a status; the remaining CRs are treated as constants. For the minus sign to be considered as a status, it must be the last character in the edit word.

The expansion is a series of ampersands and constant characters entered after the status. Ampersands are replaced by blank spaces in the output; constants are output as is. If status is not specified, the expansion follows the body.

Since a picture is worth a thousand words, the following examples, taken from IBM's reference manual should help in bringing home the notion of edit codes and how useful their use can be in your report creation.

Figure 10-14 Zero Suppression

Edit Word	Source Data	Appears in Output Record as:
'bbb0bbbbbb'	00000004	bbbb000004
'bbb0bbbbbb'	012345	bbbb012345
'bbb0bbbbbb'	012345678	bb12345678

Figure 10-15 More Zero Suppressions

Edit Word	Source Data	Appears in Output Record as:
'0bbb'	0156	b156
'0bbbb'	0156	b0156

Figure 10-16 Zero Suppression with Two Decimals

Edit Word	Source Data	Appears in Output Record as:
'bbbbbb0.bb'	000000001	bbbbbbb.01
'bbbbbb0.bb'	000000000	bbbbbbb.00
'bbb,b0b.bb'	00000012	bbbbbb0.12
'bbb,b0b.bb'	00000123	bbbbbb1.23
'b0b,bbb.bb'	00000123	bb0,001.23

Figure 10-17 Zero Suppression, Two Decimals, Asterisk Front Fill

Edit Word	Source Data	Appears in Output Record as:
'*bbbbbb.bb'	000000123	*bbbbb1.23
'bbbbb*b.bb'	000000000	******0.00

Figure 10-18 Zero Suppression, Two Decimals, Asterisk Front & Back Fill

Edit Word	Source Data	Appears in Output Record as:
'bbbbb*b.bb**	000056342	****563.42**

Figure 10-19 Zero Suppression, Two Decimals, Floating Dollar Sign

Edit Word	Source Data	Appears in Output Record as:
'bb,bbb,b$0.bb'	000000012	bbbbbbbbb$.12
'bb,bbb,b$0.bb'	000123456	bbbb$1,234.56

Figure 10-20 Miscellaneous Edit Words / Results

```
Input          Edit Word                     Edited Result

0042           "0_HRS.__MINS.&0""CLOCK"      _0HRS.42MIN._0"CLOCK

000000         "_,___.0_"                    _____.00

000000         "_,___.0_"                    | ______0

000000         "_,_0_DOLLARS__CENTS&CR"      ____0DOLLARS00CENTS___

+000002        "__0LBS.&__OZ.TARE&-"         ____0DOLLARS00CENTS___

013579         "&_,*_0,___"                  ***130,579

100199         "__/__/__"                    10/01/99

00123456       "_,__$,0__.__"                ___1$,234.56

-0000000000    "_________*&CR"               **********___

0000135678     "__,___,___DOLLARS__CENTS"    _____1,356DOLLARS78CENTS

-0034567890    "___,___,_&0.__CR**"          ___$345,678.90CR**
```

Besides the fancy edit word tricks shown in Figure 10-20, there are plenty more than those that we have the opportunity to show in this book, But, now you know how powerful the edit word facility is in RPG/400 and you have seen a number of interesting uses for this tool. If you can't find an edit code to do the formatting that you are looking for, it helps to know that there is not much you can't do with an edit word. Again, if it is taking you time to get your edit word working, and you still have issues, all of the rules are well explained in IBM's RPG/400 Reference Manual for the latest release.

The Constant Edit Word area in RPGIV is in columns 53-80.

OPDFD -- Columns 71-74 Reserved

For a field definition, positions 71 through 74 must be blank.

OPDFD -- Columns 75-80 Comments

For output, there is a paucity of space provided for comments as there is overall in RPG. Positions 75 through 80 can be used for comments, or left blank. Most programmers, who make meaningful and formatted comments in RPG, use multiple statements with the "*" in column 7 for their verbiage.

Comments in RPGIV are in columns 81 – 100.

Chapter 11

Decoding and Debugging RPG Programs

The PAREG Program Decoded

Now that we have been fully introduced to RPG/400 specifications and many of the options on each of the various coding forms, it's time to examine the PAREG program one more time to assure that we can decode it (read it with understanding) in anticipation of a potential maintenance change.

That is the other half of the development cycle in which you write and implement and then fix or enhance and implement. The enhancement that we will be making after we do the once through on the PAREG program is the introduction of the DEBUG operation to the program. In my personal experience and as I witnessed others learning RPG for the first time, a key factor in understanding the RPG cycle is to watch what is happening through the eyes of a debug listing. This is not "Source Level Debugging" but it is an innovative operation code called DEBUG which opens up the RPG cycle for viewing while a report is being printed. For the RPG craftsman needing that special look in a report or a form, the DEBUG operation is priceless.

To get the decoding process moving, let's repeat the objects that are involved in our study. Figure 11-1 is the report; Figure 11-2 and 11-3 are the data and Figure 11-4 is the compile listing of the program.

Figure 11-1 Program Output from Running Sample Learning Program

```
          THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE          2/21/06

ST      CITY          EMP#      EMPLOYEE NAME        RATE      HOURS      CHECK

PA      Wilkes-Barre   001      Bizz Nizwonger       7.80      35.00     273.00
PA      Wilkes-Barre   002      Warbler Jacoby       7.90      40.00     316.00

                                TOTAL CITY  PAY FOR Wilkes-Barre         589.00

PA      Scranton       003      Bing Crossley        8.55      65.00     555.75

                                TOTAL CITY  PAY FOR Scranton             555.75

                                TOTAL STATE PAY FOR PA                 1,144.75

AK      Fairbanks      004      Uptake N. Hibiter    7.80      25.00     195.00
AK      Fairbanks      005      Fenworth Gront       9.30      33.00     306.90
                       006 NO MATCHING MASTER                  40.00
AK      Fairbanks      007      Bi Nomial            8.80      39.00     343.20

                                TOTAL CITY  PAY FOR Fairbanks            845.10

AK      Juneau         008      Milly Dewith         6.50      40.00     260.00
AK      Juneau         009      Sarah Bayou         10.45      40.00     418.00

                                TOTAL CITY  PAY FOR Juneau               678.00

                                TOTAL STATE PAY FOR AK                 1,523.10

NJ      Newark         010      Dirt McPug           6.45      35.00     225.75

                                TOTAL CITY  PAY FOR Newark               225.75

                                TOTAL STATE PAY FOR NJ                   225.75

                                FINAL TOTAL PAY                        2,893.60
```

Figure 11-2 Query Listing of EMPMAST File Data

```
        EMP EMPNAM                EMP EMPCTY       EMP   EMP
        #                         RAT              STA   ZIP
000001    1 Bizz Nizwonger       7.80 Wilkes-Barre PA  18702
000002    2 Warbler Jacoby       7.90 Wilkes-Barre PA  18702
000003    3 Bing Crossley        8.55 Scranton     PA  18702
000004    4 Uptake N. Hibiter 7.80 Fairbanks    AK  99701
000005    5 Fenworth Gront       9.30 Fairbanks    AK  99701
000006    7 Bi Nomial            8.80 Fairbanks    AK  99701
000007    8 Milly Dewith         6.50 Juneau       AK  99801
000008    9 Sarah Bayou         10.45 Juneau       AK  99801
000009   10 Dirt McPug           6.45 Newark       NJ  07101
****** ********  End of report  ********
```

Figure 11-3 Query Listing of TIMCRD File Data

```
        EMPNO   EMPHRS
000001      1    35.00
000002      2    40.00
000003      3    65.00
000004      4    25.00
000005      5    33.00
000006      6    40.00
000007      7    39.00
000008      8    40.00
000009      9    40.00
000010     10    35.00
****** ********  End of report  ********
```

Figure 11-4 Expanded Fields Compile Listing PAREG

```
                          S o u r c e   L i s t i n g
    100  H* RPG HEADER SPECIFICATION FORMS
    200  H
    300  F*
    400  F* RPG FILE DESCRIPTION SPECIFICATION FORMS
    500  F*
    600  FPAYMAST IPEAE                    DISK
          RECORD FORMAT(S):  LIBRARY RPGBOOK FILE PAYMAST.
                   EXTERNAL FORMAT PAYR RPG NAME PAYR
    700  FEMPTIM  ISEAE                    DISK
          RECORD FORMAT(S):  LIBRARY RPGBOOK FILE EMPTIM.
                   EXTERNAL FORMAT TIMR RPG NAME TIMR
    800  FQPRINT  O   F      77     OF     PRINTER
    900  I*
   1000  I* RPG INPUT SPECIFICATION FORMS
   1100  I*
   1200  IPAYR        01
   1300  I                                           EMPNO   M1
   1400  I                EMPCTY                     EMPCTYL1
   1500  I                EMPSTA                     EMPSTAL2
   1500   INPUT  FIELDS FOR RECORD PAYR FILE PAYMAST FORMAT PAYR.
A000001                                      1   30EMPNO   M1
A000002                                      4  23 EMPNAM
A000003                                     24  282EMPRAT
A000004                 EMPCTY              29  48 EMPCTYL1
A000005                 EMPSTA              49  50 EMPSTAL2
```

```
A000006                                               51  550EMPZIP
   1600  ITIMR        02
   1700  I              EMPNO                               EMPNO   M1
   1800  C*
   1900  C* RPG CALCULATION SPECIFICATION FORMS
   2000  C*
   1700   INPUT  FIELDS FOR RECORD TIMR FILE EMPTIM FORMAT TIMR.
B000001                 EMPNO                          1   30EMPNO   M1
B000002                                                4   72EMPHRS
   2100  C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72
   2200  C   02 MR   EMPPAY    ADD  CTYPAY    CTYPAY  92
   2300  CL1         CTYPAY    ADD  STAPAY    STAPAY  92
   2400  CL2         STAPAY    ADD  TOTPAY    TOTPAY  92
   2500  O*
   2600  O* RPG OUTPUT SPECIFICATION FORMS
   2700  O*
   2800  OQPRINT  H  206   1P
   2900  O       OR  206   OF
   3000  O                                   32 'THE DOWALLOBY COMPANY
   3100  O                                   55 'GROSS PAY REGISTERBY'
   3200  O                                   60 'STATE'
   3300  O                         UDATE Y   77
   3400  OQPRINT  H  3     1P
   3500  O       OR  3     OF
   3600  O                                    4 'ST'
   3700  O                                   13 'CITY'
   3800  O                                   27 'EMP#'
   3900  O                                   45 'EMPLOYEE NAME'
   4000  O                                   57 'RATE'
   4100  O                                   67 'HOURS'
   4200  O                                   77 'CHECK'
   4300  O         D 1     02NMR
   4400  O                                   46 'NO MATCHING MASTER'
   4500  O                         EMPNO     27
   4600  O                         EMPHRS1   67
   4700  O         D 1     02 MR
   4800  O                         EMPSTA     4
   4900  O                         EMPCTY    29
   5000  O                         EMPNO     27
   5100  O                         EMPNAM    52
   5200  O                         EMPRAT1   57
   5300  O                         EMPHRS1   67
   5400  O                         EMPPAY1   77
   5500  O         T 22    L1
   5600  O                                   51 'TOTAL CITY  PAY FOR'
   5700  O                         EMPCTY    72
   5800  O                         CTYPAY1B  77
   5900  O         T 02    L2
   6000  O                                   51 'TOTAL STATE PAY FOR'
   6100  O                         EMPSTA    54
   6200  O                         STAPAY1B  77
   6300  O         T 2     LR
   6400  O                         TOTPAY1   77
   6500  O                                   50 'FINAL TOTAL PAY'
         * * * * *   E N D   O F   S O U R C E   * * * * *
```

The Decoding Process

You start the full decoding process from top to bottom. There are no entries in the Headers specification so it is moot to decode for this run.

Then as you examine the File Descriptions in the PAREG program the world of the logic in this RPG program begins to unfold before you.

There is a primary database file for master records, a secondary database file for time card records and a printer file for a report. A quick look at the output specs and you can verify that this program's mission is to print a report. From file description, you also know that the data will be in ascending sequence and a drop down look at Input tells you that there is just a sequenced field designated as M1 and its name is EMPNO. While in the Input area, as you look down to statement 17, you can see that there is a second M1 specified for the time card file. This tells you that the reading of the primary and secondary files will be controlled by the matching records logic of the RPG fixed cycle. You have learned a lot about the program already.

Looking a little closer at the INPUT specs, you will see that there is an L1 indicator specified on City and an L2 indicator on State. That typically means that the program will be performing some intermediate (in-between detail time) calculations and intermediate output probably for totaling functions when a state and/or city control break occurs. If you take a quick look at calculations, at statements 23 and 24 respectively, you can see that this hypothesis (speculation) is true as there is a total calculation for city and one for state.

Now, looking down to output, you can see that the L1 and L2 totals are defined on total cycle "T" lines at statements 55 to 65. You can see an L1 printout for a city break, an L2 printout for a state break, and a final total when the last record (LR) is processed.

You have covered a big part of the logic already in trying to get a handle on what this program does (decoding).

Going back to the input record area at statements 12 and 16, you can see that a payroll master record is identified with indicator 01 when it is read and a time card record is identified by indicator 02 when it is read. Below each of these records, you can see the input specifications that you typed in the program so as to provide the matching and control level selections and since this is a compile listing, you can also see the other fields that automatically come into the program from the externally described databases. Looking into the fields printed in the output lines, you can see

that the names are the same as the input fields. RPG does not make a programmer move input to output records for printing. Just by specifying the name on output, RPG will provide the move from the input files to the print line for you.

We have looked at just about everything so far other than what happens at detail time in the cycle. The two detail calculations shown in statements 21 and 22 are executed only when a time card record (indicator 02) is read. The two detail output lines at statements 43 and 47, are printed only when a time card record (indicator 02) is on.

Though the payroll master (PAYMAST) as processed in the primary file has its own indicator (01), the indicator is not used at all in this program. In fact, if I showed you the bottom of the compile listing, you would see that indicator 01 is unused and the compiler is complaining about it. It conditions nothing. That means that each time a master is read, nothing happens in the detail cycle. And we know that RPG intersperses the reading of the masters and the time cards through its matching records logic based on the ENPNO match fields (M1).

Of course you would also see that it is not just the 02 indicator that causes calculations to occur and lines to be printed. It has a partner in crime. In this case, the partner is the MR indicator which comes on when there is a match between the primary and secondary file. When indicator 01 and MR are on however, this means that the program has read a master record and it matches the time card about to be processed. We do nothing in this program with this information. The fact is that we need the time card information in order to have a complete unit of information for an employee. So when the condition of 02 on and MR on is true, we know that the last master read was the master for the time card being processed so the master fields and the time card fields represent the fields for the same employee. Thus, a look at lines 21, 22, and 47, shows the calculation of pay taking and the accumulation of the city total and then the printing of the line with the master, time card and the just calculated gross pay field.

There is one more condition for which this program tests and it causes output to be produced if the condition is true. The program tests to see if there is a time card record (indicator 02 is on) has been read without a corresponding matching master (indicator MR is off). In other words, if

02 and NMR are both true, then an error message gets printed by the output record at statement 43.

And that's that for simple decoding of the PAREG program. Once you have decoded a program, you then can maintain it. The next logical step for you to learn this stuff cold would be to walk through the RPG cycle with the data and see what happens. I'll get you started on that right now.

RPG Cycle and PAREG Decoding

We learned that the very first ting that happens in an RPG program that uses the cycle is that output is produced. We call this type of output "heading output," though in fact the "H" and "D" for heading and detail occur at almost the same exact spot in the detailed RPG cycle. In other words, though there is a separate heading cycle and detail cycle, in many ways they appear to be the same and not many programs need to know that they are not the same. So, now walking though this program from the start, what would happen the first cycle?

Looking at the output specifications we see some good documentation for headings in that we used the "H" designator rather than the D in lines 28 and 34. So, during the first cycle, RPG is willing to print whatever we specify in an unconditioned state or if any of the conditions are met in any of our detail or heading output lines.

So, while RPG is hoping to print something at the beginning of the program, we have both of these lines conditioned with an indicator called 1P and another called OF. These are both special indicators. RPG makes the 1P indicator available every time it passes through this very first cycle. The second time around, it will not be on. RPG turns on the OF indicator if you specify it in the "F" spec for the print file whenever the last print line of a defined print page is passed. Since our report is just one page with the data provided, OF will never come on. But, we are staged for more employees. If we have more employees and the number of print lines increases, the OF indicator will cause the print headings to print on every subsequent page in the report.

1P stands for 1st Page indicator and it is most often ORed with the OF indicator to assure that headings are printed on each page. So, in our

sample printout in Figure 11-1, the headings were created by the 1P indicator being on the first cycle of the program. Here's a question for you. If we chose to not put a 1p indicator there at all but left it blank, would any heading printing have occurred on the 1st page?

In essence we are saying to RPG that the line is to print unconditionally during the detail cycle in which heading output is produced. So, yes, we would have headings? Now, what happens on the second cycle if we have removed 1P as a conditioning indicator? Now, you've got to start looking at the data in Figures 11-2 and 11-3, because when we hit output in the second cycle, we have read our first 01 record (PAYMAST). Well, we know that we do nothing in the cycles in which a 01 record is read and RPG very nicely takes the data from the database and moves it into the PAYMAST fields, replacing the blanks and zeroes that were there from the beginning of the program. What about heading output when an 01 record is read?

If our heading lines are unconditioned, then they will print every cycle. The 1p and the OF not only tells RPG to print when 1P and OF are on, it also tells RPG to "not print" when they are not on. So, by using these indicators, we are spared from having headings on top of each detail line. In fact, if we got a heading fro 01 and then got another the next cycle for 01, we would have two sets of headings for each one detail line.

Let's go back to having the 1P there and we have just read an 01 record. When RPG takes us to detail calculations, do we have anything to calculate or print? No! Nothing is unconditioned and nothing is conditioned for 01. So, we pass through the cycle and read the next record. It comes from the secondary because matching records logic says intersperse the reading from both file base don the match field (EMPNO). The 02 record is read and MR is on. We perform the two CALCS conditioned on 02 and MR and now RPG takes us to the detail output cycle. We print the 02 MR line at statement 47 and that's it for detail output. Now, RPG takes us to total output. Do we have a control break?

Yes, The state for employee 001 is PA and the city is Wilkes-Barre. At the beginning of the program L1 and L2 fields compare against blank fields since this is the first employee for the city. Therefore, L2 and L1 calculations and L2 and L1 output should occur when we go to the total

output cycle. But it doesn't. RPG "knows that this is the first control break of the first cycle so it holds off on doing total calculations and total output until the next set of data comes in.

When the two records, 01 and 02 are read for employee 2, the same thing happens as for employee 1. However, RPG looks up when it is finished with the employee 02 detail stuff and it says hey, there is a change here for the city but not the state for employee 003. So, before employee 03 is processed, at in-between time in total calculations and total output, RPG first calculates based on L1 conditioning and then it prints the L1 totals before it goes back to read another record.

When the matching records for employee 3 are processed, RPG again looks ahead and it sees that with the next record for employee 004, there is both a state and a city change coming so it turns on indicators L1 and L2 and the L1 and L2 calculations are executed and the L1 and L2 totals for both city and state are printed. Now, if you move to the bottom of the report in Figure 9-2, you see three totals. When there are no more records to be read, RPG turns on LR which automatically turns on L2 and L1 and so the total calculations are performed and the total output is produced for city and state and also for the grand total.

Now, you might want to pick up with record 4 and walk it through the elements of the cycle that are in play and assure yourself that you now understand the one thing in RPG that most find most difficult to master – the RPG cycle. Congratulations.

Debugging for Learning and Decoding

RPG has a built in facility to help you figure out what is going on during the cycle or actually anytime you want. DEBUG has some disadvantages that skeptics may say make the tool worth less but I would argue that its simplicity and its ability to teach RPG make it a very powerful tool and one that deserves mention early in a book on learning about RPG.

The major disadvantage of the DEBUG is that you must alter your code to use it. Therefore, you must recompile your code. Then, when you have figured out the problem, you have to change the program to recompile it. Though this is true it is not as bad as all that.

The **first action** that you must take is to place a 1 in column 15 of the RPG Header Specification. This tells the compiler to honor the DEBUG calculations that are imbedded in the code. The **second thing** you must do is find a printer file or database file with a record length of at least 80 to accept the output of the debug. If your program has no such file then for the Debug to provide its output, you need to add one. The disk option helps in debugging reports in which you do not want the output format changed and it also helps in the event that you are trying to trap an elusive problem and you never know when it will strike. Notice that I had to change the QPRINT file specification by giving it a record length of 80 instead of 77 since DEBUG needs at least 80 to do its job. The **third thing** that you must do is to add DEBUG statements in the code where you would like RPG to provide program status information.

The modified PAREG code for the DEBUG problem is shown in Figure 11-5.

Figure 11-5 PAREG Program modified to support DEBUB Operation

```
0001.00      F* RPG HEADER SPECIFICATION FORMS
0002.00      H        1
0003.00      F*
0004.00      F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00      F*
0006.00      FPAYMAST IPEAE                   DISK
0007.00      FEMPTIM  ISEAE                   DISK
0008.00      FQPRINT  O   F      80      OF      PRINTER
0019.00      C* RPG CALCULATION SPECIFICATION FORMS
0020.00      C*
0021.00      C   01       'MR01'    DEBUGQPRINT    EMPNO
0022.00      C   02 MR    EMPRAT    MULT EMPHRS    EMPPAY  72
0023.00      C   02 MR    EMPPAY    ADD  CTYPAY    CTYPAY  92
0024.00      CL1          CTYPAY    ADD  STAPAY    STAPAY  92
0025.00      CL2          STAPAY    ADD  TOTPAY    TOTPAY  92
0026.00      CL2          'L1L2'    DEBUGQPRINT    TOTPAY
```

Statements 2 and 8 above are changed from the original PAREG program and statements 21 and 26 are added. In statement 21, you can see that I have asked RPG to give me a snapshot called MR01 whenever indicator 01 is on and with the snapshot, provide the value of the field called EMPNO in the snapshot output. Thus, only at 01 time will the debug be operation since as you can see, the DEBUG can be conditioned to

execute only when you want it to execute. We get no output from this DEBUG statement when an 02 record is read.

I added statement 26 to the program to show the status of the indicators during the total calculations cycle. This information can eb very revealing when you control breaks are not working correctly.

Figure 11-6 shows the way the report now looks with the DEBUGS on in the program. Notice that the first page headings worked fine as they did before since no DEBUG operations have occurred at all at this time (1P) in the program. Now, take a look down further in the report for more headings. For illustrative purposes, I have labeled this area as *** Second Page. This is not printed by the program.

Right after the 1P headings, during detail calculations after the first 01 record has been read by the input cycle, the first debug statement fires. You may recall that this is conditioned by indicator 01. Notice also that there are no detail lines printed before the DEBUG calculation. That is because the printout fired from detail calculations before RPG has even hit its second detail output cycle. During the first detail output cycle, of course, headings were printed but there were nor records read and thus, no non heading output was produced.

Figure 11-6 PAREG Report with RPG/400 DEBUG

```
           THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE         3/11/06
  ST     CITY           EMP#     EMPLOYEE NAME        RATE     HOURS      CHECK
DEBUG = 2100      MR01      INDICATORS ON = MR IR L1 L2 01
FIELD VALUE =   1
  PA     Wilkes-Barre   001     Bizz Nizwonger       7.80     35.00    273.00
DEBUG = 2100      MR01      INDICATORS ON = MR IR 01
FIELD VALUE =   2
  PA     Wilkes-Barre   002     Warbler Jacoby       7.90     40.00    316.00
                                TOTAL CITY  PAY FOR Wilkes-Barre       589.00
DEBUG = 2100      MR01      INDICATORS ON = MR IR L1 01
FIELD VALUE =   3
  PA     Scranton       003     Bing Crossley        8.55     65.00    555.75
DEBUG = 2600      L1L2      INDICATORS ON = MR IR L1 L2 01
FIELD VALUE =    114475
                                TOTAL CITY  PAY FOR Scranton           555.75
                                TOTAL STATE PAY FOR PA               1,144.75
DEBUG = 2100      MR01      INDICATORS ON = MR IR L1 L2 01
FIELD VALUE =   4
  AK     Fairbanks      004     Uptake N. Hibiter    7.80     25.00    195.00
DEBUG = 2100      MR01      INDICATORS ON = MR IR 01
FIELD VALUE =   5
  AK     Fairbanks      005     Fenworth Gront       9.30     33.00    306.90
                        006 NO MATCHING MASTER                40.00
DEBUG = 2100      MR01      INDICATORS ON = MR IR 01
FIELD VALUE =   7
  AK     Fairbanks      007     Bi Nomial            8.80     39.00    343.20
                                TOTAL CITY  PAY FOR Fairbanks          845.10
```

```
DEBUG = 2100      MR01      INDICATORS ON = MR IR L1 01
FIELD VALUE =   8
  AK     Juneau         008     Milly Dewith          6.50     40.00    260.00
DEBUG = 2100      MR01      INDICATORS ON = MR IR 01
FIELD VALUE =   9
  AK     Juneau         009     Sarah Bayou          10.45     40.00    418.00
DEBUG = 2600      L1L2      INDICATORS ON = OF 1F 2F MR IR L1 L2 01
FIELD VALUE =    266785
                                TOTAL CITY  PAY FOR Juneau               678.00
                                TOTAL STATE PAY FOR AK                 1,523.10
```

***** Second Page**

```
           THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE          3/11/06
  ST     CITY          EMP#     EMPLOYEE NAME        RATE     HOURS     CHECK
DEBUG = 2100      MR01      INDICATORS ON = OF MR IR L1 L2 01
FIELD VALUE =  10
  NJ     Newark         010     Dirt McPug            6.45     35.00    225.75
DEBUG = 2600      L1L2      INDICATORS ON = MR L1 L2 L3 L4 L5 L6 L7 L8 L9 LR
FIELD VALUE =    289360
                                TOTAL CITY  PAY FOR Newark               225.75
                                TOTAL STATE PAY FOR NJ                   225.75
                                   FINAL TOTAL PAY                     2,893.60
```

The field value for EMPNO in the first DEBUG is 1 or an edited “001” as the field data is available during the processing of the 01 record in calculations. Now look across the line to something called INDICATORS ON. There is a world of information there. You can see that five indicators, **MR IR L1 L2 01**, are on. So, w e can see that though the 02 record has never been read for this employee, RPG has turned on MR to designate that this 01 record matches the 02 record that will be read. The IBM RPG/400 Reference manual offers me no clue as to what indicator IR is and I have found no reference on the Internet so I guess it is something that RPG thinks it needs to be OK. Perhaps it means internal routine??

Why are L1 and L2 on while we are processing the first record that has been read? Can you figure that out? We are in the detail calculations phase right now. Why is L1 and L2 on when we have yet to have one in-between time. It has to do with the first cycle test. RPG does turn these indicators on and they stay on until right before RPG tests again to see if there is a break in the next record to be read. So, the L1 and L2 indicators stays on through detail calculations and detail output. What good is this and What can be done with it if it is good?

Well, it is good because it signifies to RPG that it is processing the first record of a new group. Whereas total level time is in-between, and we have yet to actually read the record that caused the break, detail L1 / L2 is when the record is actually being processed. It helps to know that this is

the first record of a group because sometimes report designers like to group indicate.

Group indication is a formatting trick that provides just the first line of detail print with the control field value, say state for L2 and city for L1. The next line would not print the state since it is the same state. The same notion goes for City as you can see in the sample report shown in Figure 11-6

Figure 11-6 Group Indication of PAREG Report

```
          THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE          3/11/06
ST     CITY            EMP#     EMPLOYEE NAME         RATE     HOURS     CHECK

PA     Wilkes-Barre     001     Bizz Nizwonger        7.80     35.00    273.00
                        002     Warbler Jacoby        7.90     40.00    316.00

                                TOTAL CITY  PAY FOR Wilkes-Barre        589.00

       Scranton         003     Bing Crossley         8.55     65.00    555.75

                                TOTAL CITY  PAY FOR Scranton            555.75

                                TOTAL STATE PAY FOR PA                1,144.75
```

To make this work, you would use field indicators. On State, for example, on the detail record report line, you would place L2, as a field indicator and for city, you would place L1. Then the report would print just as in Figure 9-6. What makes it work is that L1 and L2 and the other L indicators stay on through detail output then they go off. When they are off, the state and city does not print. They print only on the first record of a new group – or detail L1 / detail L2 time.

Figure 11-7 Snippet Of Debug Listing

```
  PA     Scranton        003     Bing Crossley         8.55     65.00     555.75
DEBUG = 2600      L1L2      INDICATORS ON = MR IR L1 L2 01
FIELD VALUE =    114475
                                TOTAL CITY  PAY FOR Scranton             555.75
                                TOTAL STATE PAY FOR PA                 1,144.75
DEBUG = 2100      MR01      INDICATORS ON = MR IR L1 L2 01
FIELD VALUE =   4
  AK     Fairbanks       004     Uptake N. Hibiter     7.80     25.00     195.00
```

Figure 11-7 is a smaller snippet of the big debug listing shown first in Figure 11-5. It shows the data that caused the first L2 control break in the program between states PA and AK. The last employee for Pennsylvania

is shown on the top and the first employee for Alaska is shown on the bottom. Notice that the DEBUG marked L1L2 on the second line. It fired from an L2 calculation in the program at statement 26 as shown below:

```
0026.00       CL2             'L1L2'    DEBUGQPRINT
```

So, we know that in order for this DEBUG to fire, L2 must be on. The L1L2 in Factor 1 merely creates a marker in the reports so you can see where your DEBUGS are firing.

Clearly from Figure 11-7 we can see control field changes in both the city and the state fields and the DEBUG in line 2 shows us, that among a bunch of other particulars, L1 and L2 are on as you would expect. Notice also that at total calculation time, when this Debug fired, indicator 01 is also on. Yet, the last record processed for employee # 003 was an 02 (time card) record. So, why is 01 on?

It is the record identifying indicator of the next record to be processed. The fields for the 02 record still contain employee 003's data, however, RPG has turned off indicator 02 already at this point in the total calculations cycle and it has turned on the identifying indicator 01 of the next record to be read. This tells the programmer a bunch of interesting things.

First, it tells the programmer that record identifying indicators cannot be used at in-between time for prior groups. So, and 02 on line 26 would not fly. The other thing it says is that if need be, a programmer can know ahead of time, which record type was about to be read in. Sometimes this knowledge could help a programmer take a different logic path than otherwise. It also shows us that record identifying indicators get turned off during "in- between time" and they get turned on during in-between time. When the 01 record data is actually read, the 01 record DI will continue to be on until in-between time when the next record is looked at.

For a detail lesson in RPG, take another go at running the data through this program and you will have a pretty good idea of how the RPG cycle works and when it turns on stuff and when it turns stuff off. After

awhile, it will all seem logical because of course, it is. See the PAREG with an even more detailed RPG cycle in Figure 11-8.

Figure 11-8 PAREG and the RPG Cycle

Points in RPG Cycle	**PAREG Activity- RPG Cycle**
0. Start	PAREG program is loaded and started.

1. Heading & Detail Output -- What is there to output?	`0028.00 OQPRINT  H  206   1P`
	`0029.00 O       OR  206   OF`
	`0034.00 OQPRINT  H  3     1P`
	`0035.00 O       OR  3     OF`

2.1 Turn off record id & Level Indicators. None on the first cycle	Time card or pay master records (01, 02). Level indicators between control fields -- L1, L2
2.2 Read a record from file just processed. At start, read a record from each file in sequence specified	Read primary and secondary at start to see which gets processed first.
2.3 Check for end of file	If all files are at end, set on the last record indicator and indicators L0 through L9
2.4 Are matching fields specified	Select highest priority record from appropriate file based on match field primary, secondary and match field value.
2.5 Turn on record ID indicator. You pick the #, the cycle turns it on7.	First cycle through the primary (01) record (Master) is selected, Subsequently the primaries and secondaries alternate based on matched field values.
3.0 Have any control fields changed, if so, turn on Level indicators – also test for first cycle... If first cycle, skip total calcs & total output by going to Step 6.0 If not first cycle, go to step 3.1	First cycle always creates a false control break because the control fields compare against a blank or zero value. When a city changes at this point, L1 turns on and when a state changes, L2 and L1 are turned on.

3.1 Perform total calcs (caused by control field change). You pick the control fields, RPG does the tests. When the fields change, the level indicator is turned on.	City and state total calculations are taken during this time in the cycle
	`23.CL1 CTYPAY  ADD STAPAY STAPAY 92`
	`24.CL2 STAPAY  ADD TOTPAY TOTPAY`

4.0 Perform total output(caused by control field change) You pick the field, RPG does the tests. Output conditioned with a T and a level indicator is performed.	City and state total output (subtotals) are performed at this time during the cycle
	`55.T 22  L1`
	`56.          51 'TOTAL CITY PAY FOR'`
	`57.  EMPCTY  72`
	`58.  CTYPAY1B77`
	`59.T 02  L2`
	`60.          51 'TOTAL STATE PAYFOR'`
	`61   EMPSTA  54`

	`62   STAPAY1B77`
	`63.T 2   LR`
	`64.  TOTPAY1 77`
	`65.           50 'FINAL TOTAL PAY'`

5.0 Check to see if LR is on. If LR is on, go to end of cycle and end job.	LR will not turn on in this program until the last secondary record (time card is read from the secondary file.
5.1 If LR not on, check to see if overflow has occurred and overflow indicator (s) are on.	RPG offers another time in the cycle not described in the 7 point cycle in which overflow printing occurs. It is close to detail cycle calculations and output but the data from the last record read has not yet become available.

5.2 If a print line has overflowed, perform overflow output cycle. Print those lines conditioned by an overflow indicator such as OF.	PAREG has overflow defined and in the debugged program, it occurs.
	`0028.00 OQPRINT  H  206   1P`
	`0029.00 O       OR  206   OF`
	`0034.00 OQPRINT  H  3     1P`
	`0035.00 O       OR  3     OF`

6.0 Move data from input area to fields	Fields are now available from the master or time card record – whichever was read last.

6.1 Is selected record a matching record? If so, turn on MR. Is there a match of the PAYR record from the master with the TIMR record from the time card file?	PAREG uses matched fields (M1) to identify matching records.
	`0012.00 IPAYR   01`
	`0013.00 I                   EMPNO   M1`
	`0014.00 I                   EMPCTYL1`
	`0015.00 I                   EMPSTAL2`
	`0016.00 ITIMR   02`
	`0017.00 I                   EMPNO   M1`

6.2 If there is a match, turn on the MR indicator. If there is no match, turn off the MR indicator.	RPG processes just one record at a time. If the master is in process and there is a match with a to-be-read secondary, then the record ID (01) for the master is also on. If the time card is in process, 02 is on.
7.0 Perform Detail Calculations	`21.C 02 MR EMPRAT MULT EMPHRS EMPPAY 72` `22.C 02 MR EMPPAY ADD  CTYPAY CTYPAY`
Repeat Cycle	PAREG goes another cycle

When you have the two machinations of the RPG fixed logic cycle digested and you want even greater detail on the cycle, it's time to get out the IBM reference manual. For your convenience we have provided a link to the RPG cycle in the IBM Boulder online books library:

http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/c092508347.htm#FIGDETLOBJ.

Chapter 12

Introduction to RPGIV

You've Already Seen RPGIV?

So far in this book as we decomposed and decoded the PAREG program using the RPG/400 specification templates, we also presented the formatting of the RPG/400 specifications used in PAREG for RPGIV. To an extend then, this chapter on RPGIV is somewhat redundant – at least as far as we have gotten in our study of the RPG language.

The Fact is, however, RPGIV is not just RPG/400 with a better coding sheet design. It's lots more than that and in this chapter we begin a series of chapters in which we differentiate the two languages better than was necessary in earlier chapters.

A Better RPG

In 1988, with the introduction of its highly successful AS/400 series of machines, IBM introduced a somewhat new name for an old language. The language was RPG III and the new moniker was RPG/400. There was little if any difference between RPG III on System/38 and the new version of RPG for the AS/400 known as RPG/400.

To be correct, RPG/400 is actually a compiler package that contains RPG II for System/36 compatibility, RPG III for System/38 compatibility and the RPG III for the AS/400 which also took on the name RPG/400.

As of OS/400 Version 3 release 1, IBM changed the name of this compiler package to "AS/400 ILE RPG/400". With this the company

introduced the RPG IV language which is a version of RPG that targets the Integrated Language Environment (ILE).

ILE is the latest "native" runtime environment for programs on the System i5. Way back in OS/400 Version 2 Release 3, IBM introduced this new program model that in many ways essentially changed the way the operating system works with languages. This new programming model now provides support for a mixed set of high-level languages. But, in V2R3, C/400 was the only ILE language and it became the basis for the other ILE languages soon to follow.

Prior to ILE, RPG for example had its own runtime environment. To be fair, CL had its own; COBOL had its own, and C also had its own. Thus, there were often issues trying to get two different languages to work well together in the same job. With ILE, all programming languages now run in ILE. Thus, the same "environment" is now used for COBOL, C, C++, RPG and CL.

To take advantage of the functionality built into the ILE, however, IBM had a lot of work to do. New compilers needed to be created. With its new RPG IV compiler introduced first in 1994, IBM decided to make full use of the ILE. This had the double advantage of providing a new version of RPG with an ILE targeted compiler.
Though ILE wasn't really ready for RPG IV in 1994 with V3R1 of CISC architecture OS/400, RPGIV made its debut and it's initial syntax addressed many of the ills that programmers had begged IBM for years to correct – especially the multiple faces of RPG/400 input.

In 1994 in its CISC processors (that's all IBM had back then), Big Blue delivered a new language with a new syntax. Yet, this new language with its new syntax had a nice familiar flavor so that the RPG/400 programming community were able to almost immediately understand it as a new language. The new ILE notions were somewhat more problematic for the RPG community.

The new RPGIV defined a new look for all of RPG's specifications. The specs were revitalized and made much better. Since IBM had not really made any substantial changes to RPG for ten or more years at the time, there was lots of long overdue facility made available with this "new language."

Besides the changes to all the forms, IBM also added a new data specification – the "D spec." The "D" spec helped make the formerly stodgy language much easier to learn and much more similar to other languages. The "D" spec in many ways brought to RPG the capabilities that had been in COBOL's Data Division from the outset.

Programmers from other systems and other languages on the AS/400 were then able to learn RPG coding without having to learn the RPG cycle and many of the other RPG nuances that you have just learned in the prior 11 chapters. And, though this was mostly good, a disadvantage was that new RPG programmers often were not introduced to the old RPG. This created two RPG camps that today still have some disagreements. There are those that know, understand and love the RPG cycle, who use it as a 4GL, and there are those who think that the RPG cycle is worth about as much as one of those electromechanical monsters from which the cycle originated would be worth in today's world.

Popular thought has RPGIV first being developed for the RISC models but this is not true. IBM created RPGIV in 1994 and released it with V3R1 for its CISC processors. It originally ran under a programming model that got its name only after ILE was introduced. If the Integrated Language Environment was the new name for how IBM would bring all of the code produced by all of its compilers together, then, what was it that had existed prior to this new model? It was the same dilemma as RISC vs. CISC. CISC got its name when RISC came out.

With OS/400 V3R1, They christened the old "no name" model as "the original program model," or simply OPM. Thus OPM is just a name that has been given to the original runtime environment of RPG and CL under OS/400.

Since about 1995 with the availability of the RISC boxes, ILE has become the native mode and now the OPM a.k.a. the original native environment is emulated. Reading between the lines you can see that once ILE was the way, everything else became part of ILE. It was that good of an idea. ILE isn't really an environment at all, today it is in fact, the native OS/400.

So, where does that leave OPM? It still exists but it is a somewhat Rube Goldberg emulated environment running under ILE. Because of this

emulation factor, and the machinations that IBM had to perform to perfect it, the new wave of RPG IV purists who understand the underpinnings of this issue in detail, do not think so well of RPG/400 in the ILE world. Yet, despite their thoughts and the concerns of skeptics over the years, RPG/400 performs very well within its new ILE home. In many ways, RPG/400's fine performance is a big reason why there has been no compelling need for the full adoption of RPG IV in the one-language RPG shop. Twelve years after its introduction, full adoption of RPGIV is under 50%

Yet, as you will note RPGIV is a far superior language than is RPG/400. Just the specification changes alone with the addition of the "D" spec were major improvements and enough to make the language a worthy replacement for RPG/400 code. You'll see for yourself shortly.

One of the tools that IBM built to help in the transition from RPG/400 to RPGIV is a program source conversion command. This tool converts RPG/400 to RPGIV in the most natural way. In other words, the flavor of the original program is preserved and the programmer can recognize the RPGIV version of the code quite readily without having to do substantial decoding.

The format of the CVTRPGSRC command is shown in Figure 12-1.:

If you have never used the CVTRPGSRC command, it will definitely burp the first time because there is no log file (QRNCVTLG). The message you get will tell you exactly how to create the log file. The key things about this command and how the new RPGIV code is stored are as follows:

1. You specify the QRPGSRC file for the RPG/400 code.
2. You specify the library for the RPG/400 code – RPGBOOK.
3. You specify the source member –PAREG.
4. You specify the QRPGLESRC file for the RPGIV code
5. You specify the library in which QRPGLESRC is located.
   The record length is 112 characters to accommodate the expanded source record (100 characters in RPGIV).
6. You specify "yes" for source template to get formats intersperses inside of your program to more easily see the new RPG specification formatting.

Figure 12-1 IBM's CVTRPGSRC Command

```
                    Convert RPG Source (CVTRPGSRC)

Type choices, press Enter.

From file . . . . . . . . . . . . > QRPGSRC
  Library . . . . . . . . . . . . >   RPGBOOK
From member . . . . . . . . . . . > PAREG
To file . . . . . . . . . . . . .   QRPGLESRC
  Library . . . . . . . . . . . . >   RPGBOOK
To member . . . . . . . . . . . .   *FROMMBR

                             Additional Parameters

Expand copy member . . . . . . .    *NO
Print conversion report . . . .     *YES
Include second level text . . .     *NO
Insert specification template .     *yes
Log file . . . . . . . . . . . .    QRNCVTLG
  Library . . . . . . . . . . .       *LIBL
Log file member . . . . . . . .     *FIRST
                                                  Bottom
```

The converted code for PAREG is shown in Figure 12-2. Before we examine the coding changes, let's take a look at some of the nuances that are shown in Figure 12-2. The first item of interest is the formatting lines themselves. Because I specified "yes" for a source template, lines, 2. 7, 14, 15, and 26 were automatically generated and included in the converted source as comments. These statements takes us through calculations and then there are no more format records.

You cans see that each of added source records for formatting was also numbered by the converter. Yet, for some reason, this command did not place the format lines for output into the file. For our learning purpose, I had requested formats to make it easier to read the source. When I examined the code in the source file, I added the two SEU FMT lines for output and I left the FMT in the source to differentiate these lines from those auto-generated.

The other two nuances regarding the format lines are (1) in some cases, the format was inserted before the first spec form of a given type and in others it was after the first. and (2) for INPUT, the record format and the field format were added together, one line after the other, and were not separated by record and field form type.

To make the code easier for you to read, I highlighted the format statements as you can see in Figure 12-2. To make It easier for you to compare the programs, The RPG/400 version is shown in Figure 10-2.

The first difference you would notice is that the RPGIV program has 72 statements whereas the RPG/400 program has just 65. However, five of the RPGIV statements are used for formatting and none are used for formatting in the RPG/400 version. Therefore there are two extra lines in the RPGIV code. Where are they?

If you add up all the spec types, there is a one for once conversion in this program except for one specification type – calculations. With the structured operations that were placed in RPG III years ago, many RPG programmers had begun to use IF statements and DO statements to control conditioning. So, when IBM redesigned the calculation specification, rather than three "Anded" areas on one calculation spec, RPGIV was designed with space fro just one conditioning indicator. The RPG/400 PAREG program uses both indicators 02 and MR to condition the detail calculations. Since RPGIV offers just one spot, to get MR and 02 as conditioning indicators, the converter needed two calc specs that use the AN linkage to extend the condition. See lines 25 to 29 in Figure 12-2. So now, everything else in this program is one to one.

Figure 12-2 RPGIV Version of PAREG

```
       6789112345678921234567893123456789412345678951234567896123456789712
001.00 H* RPG HEADER SPECIFICATION FORMS
002.00 H*eywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
003.00 H
005.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
006.00 F*
007.00
F*ilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++
008.00 FPAYMAST   IPE AE             DISK
009.00 FEMPTIM    ISE AE             DISK
010.00 FQPRINT    O    F   77        PRINTER OFLIND(*INOF)
011.00 I*
012.00 I* RPG INPUT SPECIFICATION FORMS
013.00 I*
014.00
I*ilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..........................
015.00
I*..............Ext_field+Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr
016.00 IPAYR           01
017.00 I               EMPNO                       EMPNO           M1
018.00 I                                           EMPCTY        L1
019.00 I               EMPSTA                      EMPSTA        L2
020.00 ITIMR           02
021.00 I               EMPNO                       EMPNO           M1
022.00 C*
023.00 C* RPG CALCULATION SPECIFICATION FORMS
024.00 C*
025.00 C   02
026.00
C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLo
027.00 CAN MREMPRAT        MULT      EMPHRS         EMPPAY             7 2
028.00 C   02
029.00 CAN MREMPPAY        ADD       CTYPAY         CTYPAY             9 2
030.00 CL1   CTYPAY        ADD       STAPAY         STAPAY             9 2
031.00 CL2   STAPAY        ADD       TOTPAY         TOTPAY             9 2
032.00 O*
033.00 O* RPG OUTPUT SPECIFICATION FORMS
034.00 O*
FMT O
O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.....................
035.00 OQPRINT    H    1P                  2 06
036.00 O         OR    OF                  2 06
FMT P
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTfor
037.00 O                                            32 'THE DOWALLOBY
COMPANY'
038.00 O                                            55 'GROSS PAY REGISTER
BY'
039.00 O                                            60 'STATE'
040.00 O                       UDATE          Y     77
041.00 OQPRINT    H    1P                   3
042.00 O         OR    OF                   3
043.00 O                                             4 'ST'
044.00 O                                            13 'CITY'
045.00 O                                            27 'EMP#'
046.00 O                                            45 'EMPLOYEE NAME'
047.00 O                                            57 'RATE'
048.00 O                                            67 'HOURS'
049.00 O                                            77 'CHECK'
050.00 O          D    02NMR             1
051.00 O                                            46 'NO MATCHING MASTER'
052.00 O                       EMPNO                27
053.00 O                       EMPHRS         1     67
054.00 O          D    02 MR             1
055.00 O                       EMPSTA                4
056.00 O                       EMPCTY               29
057.00 O                       EMPNO                27
058.00 O                       EMPNAM               52
059.00 O                       EMPRAT         1     57
060.00 O                       EMPHRS         1     67
061.00 O                       EMPPAY         1     77
062.00 O          T    L1                2  2
063.00 O                                            51 'TOTAL CITY  PAY FOR'
064.00 O                       EMPCTY               72
065.00 O                       CTYPAY         1B    77
066.00 O          T    L2                0  2
067.00 O                                            51 'TOTAL STATE PAY FOR'
068.00 O                       EMPSTA               54
069.00 O                       STAPAY         1B    77
070.00 O          T    LR                2
071.00 O                       TOTPAY         1     77
072.00 O                                        50 'FINAL TOTAL PAY'
```

Figure 12-3 *RPG/400 Version of PAREG*

```
        *************** Beginning of data **********************
        678911234567892123456789312345678941234567895123456789612345678
0001.00 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
0002.00 H
0003.00 F*
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FEMPMAST IPEAE                    DISK
0007.00 FTIMCRD  ISEAE                    DISK
0008.00 FQPRINT  O   F      77     OF     PRINTER
0009.00 I*
0010.00 I* RPG INPUT SPECIFICATION FORMS
0011.00 I*
0012.00 IPAYR        01
0013.00 I              EMPNO                          EMPNO   M1
0014.00 I                                             EMPCTYL1
0015.00 I              EMPSTA                         EMPSTAL2
0016.00 ITIMR        02
0017.00 I              EMPNO                          EMPNO   M1
0018.00 C*
0019.00 C* RPG CALCULATION SPECIFICATION FORMS
0020.00 C*
0021.00 C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72
0022.00 C   02 MR   EMPPAY    ADD  CTYPAY    CTYPAY  92
0023.00 CL1         CTYPAY    ADD  STAPAY    STAPAY  92
0024.00 CL2         STAPAY    ADD  TOTPAY    TOTPAY  92
0025.00 O*
0026.00 O* RPG OUTPUT SPECIFICATION FORMS
0027.00 O*
0028.00 OQPRINT  H  206   1P
0029.00 O       OR  206   OF
0030.00 O                               32 'THE DOWALLOBY COMPANY'
0031.00 O                               55 'GROSS PAY REGISTER BY '
0032.00 O                               60 'STATE'
0033.00 O                     UDATE Y   77
0034.00 OQPRINT  H  3     1P
0035.00 O       OR  3     OF
0036.00 O                                4 'ST'
0037.00 O                               13 'CITY'
0038.00 O                               27 'EMP#'
0039.00 O                               45 'EMPLOYEE NAME'
0040.00 O                               57 'RATE'
0041.00 O                               67 'HOURS'
0042.00 O                               77 'CHECK'
0043.00 O        D 1      02NMR
0044.00 O                               46 'NO MATCHING MASTER'
0045.00 O                     EMPNO     27
0046.00 O                     EMPHRS1   67
0047.00 O        D 1      02 MR
0048.00 O                     EMPSTA     4
0049.00 O                     EMPCTY    29
0050.00 O                     EMPNO     27
0051.00 O                     EMPNAM    52
0052.00 O                     EMPRAT1   57
0053.00 O                     EMPHRS1   67
0054.00 O                     EMPPAY1   77
0055.00 O        T 22     L1
0056.00 O                               51 'TOTAL CITY  PAY FOR'
0057.00 O                     EMPCTY    72
0058.00 O                     CTYPAY1B  77
0059.00 O        T 02     L2
0060.00 O                               51 'TOTAL STATE PAY FOR'
0061.00 O                     EMPSTA    54
0062.00 O                     STAPAY1B  77
0063.00 O        T 2      LR
0064.00 O                     TOTPAY1   77
0065.00 O                               50 'FINAL TOTAL PAY'
        ***************** End of data ****************************
```

Decoding the PAREG RPGIV Program

Before we give a general column definition for the newer RPGIV specs, let's see if the resulting program looks like RPG or something else. Since you are quite familiar with the RPG/400 version, take a good look at the RPGIV version before you read any more and see if you can decode it as we did the RPG/400 version in Chapter 11.

The RPGIV Header Specification

Let's start with the PAREG RPG/400 Header specification in Figure 12-4 and note its changes in RPGIV as shown in Figure 12-5. Then, let's move on to the rest of the specs from there. Just as there is no H information at all provided for the RPG/400 version, there is none provided for RPG IV as you can see. The H entry has no entries at all – no keywords. However, the format line at statement 2 provides a clue that something is different.

Figure 12-4 Header RPGIV Spec – No Debug

```
       678911234567892123456789312345678941234567895123456789612345678
0001.00 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
0002.00 H
```

Figure 12-5 Header RPGIV Spec – No Debug

```
       67891123456789212345678931234567894123456789512345678961234567897123 45
001.00 H* RPG HEADER SPECIFICATION FORMS
002.00 H*eywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
003.00 H
```

H*eywords in line 2 of Figure 12-4 is a comment using a HEADER spec. This comment was inserted by CVTRPGSRC when it converted the RPG/400 source to RPGIV. If we replaced the asterisk with a "k" it would read "Keywords." That is the difference in a nutshell. The H spec now is keyword only. There are no columnar designations at all for the RPGIV Header.

In Figure 12-6, to demonstrate the DEBUG facility, we added a "1" in column 15 to the RPG/400 H spec to tell it to turn on debug for the program. Since there is no longer a spot for a 1 in column 15 of the RPGIV H specs, how do you handle the notion of debug as in RPG/400?

There is a keyword. And the keyword is DEBUG as shown in Figure 12-7. In fact, if you explored the H spec for RPG/400, you would find that those values that were once in H spec columns now are all represented by RPGIV keywords. It makes it easier to remember and hard to get in the wrong column.

Figure 12-6 Header RPGIV Spec Debug

```
        6789112345678921234567893123456789412345678951234567896123456789
0001.00 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
0002.00 H         1
```

Figure 12-7 Header RPGIV Spec with DEBUG Keyword

```
       678911234567892123456789312345678941234567895123456789612345678971234 5
001.00 H* RPG HEADER SPECIFICATION FORMS
002.00 H*eywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
003.00 H DEBUG
```

RPGIV File Description Specification

Let's move on down to the File Description section. The RPGIV code is shown in Figure 12-8 and the RPG/400 code is shown in Figure 12-9.

Figure 12-8 RPG IV File Description Spec

```
      67891123456789212345678931234567894123456789512345 67
07.00 F*ilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++
08.00 FPAYMAST   IPE AE             DISK
09.00 FEMPTIM    ISE AE             DISK
10.00 FQPRINT    O    F   77        PRINTER OFLIND(*INOF)
```

Figure 12-9 RPG/400 File Description Spec

```
      67891123456789212345678931234567894123456789512345 67
FMT…  Filename IPEAF........L..I........Device+......KExit
06.00 FEMPMAST IPEAE                    DISK
07.00 FTIMCRD  ISEAE                    DISK
08.00 FQPRINT  O   F      77      OF    PRINTER
```

Doing a quick compare, you can readily see that the RPGIV code is a little bit tighter and not as spread out. Moving from left to right, the first difference is that there are 10 spaces for the File name compared to 8.

Since AS/400 objects have a natural max name length of 10 characters, this makes RPG capable of handling a file with a name length of 10.

Continuing the trek from left to right, you can see that there is a space between the glob of code for the primary and secondary files – IPEAE and ISEAE respectively. The new layout adds a column between the end of file designator and the ascending sequence columns. It is blank in the PAREG program. Its meaning is File addition. In other words, the RPG/400 column 66 entry has been moved to column 20 of the new RPGIV File description specification.

Moving again from left to right, you will notice that the OF indicator from RPG/400 is removed completely and that the device name is moved further to the left, now starting in column 36 instead of column 40. The OF indicator is put back via a keyword starting in column 44.

OFLIND(*INOF)

The OFLIND keyword is one of many RPGIV File Description keywords. In this case, the keyword says to use the *INOF a.k.a. the reserved word OF as the overflow indicator. Its purpose is exactly the same as the OF entry in RPG/400. Ant that does it for the F spec RPGIV entries needed for the converted PAREG program.

RPGIV Input Spec Changes

One of my personal observations regarding the CVTRPGSRC command is that IBM did not invest a lot of resources in assuring that all was clean. Just as an example, the record format name PAYR in statement 12 of the RPG/400 input specs when converted shows a format line in statement 14 with *ilename and this is not correct even if you replace the asterisk with the "F." It presents the internally described record format instead of the external format. For a facility that has been out for over ten years, this problem should already be fixed. In Figure 12-10, to make up for the wrong formatted provided by the converter, I placed correctly shaped format records on top of record line 16 and field line 21.

Figure 12-10 Converted RPGIV Input Specs

```
   6789112345678921234567893123456789412345678951234567896123456
14.00 I*ilename++SqNORiPos1+NCCPos2+NCCPos3+NCC...................
15.00 I*.............Ext_field+Fmt+SPFrom+To+++DcField+++++++++L1M1
FMTIX IRcdname+++....Ri.............
16.00 IPAYR          01
17.00 I                                          EMPNO           M1
18.00 I              EMPCTY                      EMPCTY        L1
19.00 I              EMPSTA                      EMPSTA        L2
20.00 ITIMR          02
FMTJX I..............Ext-field+..................Field+++++++++L1M1
21.00 I              EMPNO                       EMPNO           M1
```

Figure 12-11 RPG/400 Input Specs

```
      678911234567892123456789312345678941234567895123456789612345
FM IX IRcdname+....In.........
FM JX.I..............Ext-field+......................Field+L1M1
12.00 IPAYR        01
13.00 I              EMPNO                           EMPNO   M1
14.00 I                                              EMPCTYL1
15.00 I              EMPSTA                          EMPSTAL2
16.00 ITIMR        02
17.00 I              EMPNO                           EMPNO   M1
```

Record ID in RPGIV

Let's walk down the specifications for Input first looking at RPG/400 right above and then RPGIV. Lines 12 and 16 in Figure 10-8 represent the record format. This converts to lines 16 and 20 in Figure 10-7. If you were glancing without concentrating at the RPG/400 and RPGIV record format statements, you might conclude that they were exactly the same. That's how close they are. The both RPGs are extremely easy to read because of this. The two difference to the record format are that the format name area is expanded to 10 positions and that instead of positions 19 and 20, the Record ID Indicator in RPGIV is found in positions 22 and 23.

Input Field Spec in RPGIV

Four fields are defined in the RPG/400 program at statements 13, 14, 15, and 17. These are converted to statements 17, 18, 19, and 21 respectively. Besides all of the "from and to" positions changing, the major change to

RPGIV is that the field name area is now 14 positions long. The length of the field name itself has been increased to 10 positions but the area in RPG in which to specify the field has grown by an additional 4 positions so that indented The field is specified from positions 49 to 62 of the record. And thus the programmer now has the latitude of indentation of input to make it more readable.

In case you want to rename a field, the external name is still in positions 21 – 30. The area in which group levels and match fields are coded has moved from positions 59 – 62 to 63 – 66 respectively. And, that's about it for significant change to INPUT.

RPGIV Calculation Spec Changes

Figure 12-12 Converted RPGIV Calc Specs

```
      6789112345678921234567893123456789412345678951234567896123456789 0
FMT C CL0N01Factor1+++++++Opcode&ExtFactor2+++++++Result++++++++Len++D+
23.00 C* RPG CALCULATION SPECIFICATION FORMS
24.00 C*
25.00 C   02
26.00 C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+
27.00 CAN MREMPRAT        MULT      EMPHRS        EMPPAY               7 2
28.00 C   02
29.00 CAN MREMPPAY        ADD       CTYPAY        CTYPAY               9 2
30.00 CL1   CTYPAY        ADD       STAPAY        STAPAY               9 2
31.00 CL2   STAPAY        ADD       TOTPAY        TOTPAY               9 2
```

Figure 12-13 RPG/400 Calc Specs

```
         6789112345678921234567893123456789412345678951234567896123
        CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+
0018.00 C*
0019.00 C* RPG CALCULATION SPECIFICATION FORMS
0020.00 C*
0021.00 C   02 MR   EMPRAT    MULT EMPHRS    EMPPAY  72
0022.00 C   02 MR   EMPPAY    ADD  CTYPAY    CTYPAY  92
0023.00 CL1         CTYPAY    ADD  STAPAY    STAPAY  92
0024.00 CL2         STAPAY    ADD  TOTPAY    TOTPAY  92
```

If it were not for the formatting lines and the two AN lines, the RPGIV calc specs in Figure 10-9 would very similar to the RPG/400 version in Figure 10-10. The biggest change so far in this program is that the number of "ANDed" conditioning indicators has been reduced from three to one per each RPG calculation statement. Thus, for those heavily indicator driven calculations, as many as three RPGIV statements may be

needed for each RPG/400 operation conditioned by three "AN" indicators. In this case, we need one extra statement for each detail calculation as shown in lines 25 and 28 of Figure 12-12.

Besides the AN indicator changes, the space for just about everything has become larger in the new RPG Calculations Specification. Factor 1 and Factor 2 and the Result Field now begin in columns 12 and 36 and 50 respectively and each entry has been widened to 14 positions. This enables indented calculations with ten character field names as well as larger literals / constants.

The operation code area is also larger providing from positions 26 to 35 supporting operation codes with extenders up to ten characters. The operation extender was formerly specified in column 53 and was traditionally labeled as the half-adjust column. For numeric operations, an H in column 53 told the compiler to round up. In RPG IV, operations such as an ADD with half adjust as you will see when we cover operations in Chapter **** is specified as such:

```
ADD(H)
```

The parentheses are required. There are now many operations with RPGIV that can use the extender portion. Still, compared with the five position operations from RPG/400 that once were happy to exist in columns 28 to 32, there is plenty of room for the operation, the extenders as well as a lengthening of the base operation to make it more readable in English. Though I would like to tell you that IBM redid the multiply (MULT) in RPGIV but it did not. So, for me to give you an example, let me say that the operation for updating a record on disk in RPG/400 is UPDAT. That's about all you can get in a 5 character space. However, in RPGIV, IBM added the E at the end to make the new UPDAT command in RPGIV read much better as:

```
UPDATE
```

RPGIV Field Length and Decimals

The new field length columns (64-68) and the number of decimals in (69-70) have grown by one each from the RPG/400 calc spec and in V5R1 the largest numeric field has grown from 15 to 31 characters and the largest alphabetic field has gone from 256 to 65535. To support this, the length is now five columns and the length of the # of decimal positions in RPGIV is now 2, now permitting more than 9 decimal positions in a field.

RPGIV Output Spec Changes

To make it very convenient to compare the differences and mostly the similarities between RPGIV and RPG/400 output in Figure 12-14 and Figure 12-15, both programs are squeezed onto the same one page. But, to contrast individual features of RPGIV with RPG/400, as we did for all other spec sheets, we ill narrow in and blow-up lines for a more concentrated examination. For example, to get a better idea of the new output record format, take a look at Figures 12-14 and 12-15.

RPGIV Output Record Format and Control

Just as with every other RPGIV spec sheet, the size reserved for name lengths has increased for the output spec. From left to right on the output record format, the filename has grown to 10 positions starting at 7. The detail or total (D or H) indication has been relocated to column 17 instead of 15 because of the longer file name.

The printer device control area has been completely moved from its original position of importance when RPG was a report writing language. The RPG IV designers, who have a mission to minimize the report writing facility of RPG and concentrate on its strong database and workstation facilities, seem to have hidden the printer control portion of output, but they have added a PRTCTL keyword to File Descriptions to help out. Without printer control as was the next area of consequence in RPG/400, RPGIV goes right on to conditioning indicators. So, the new RPGIV output record form now finds its nine position conditioning indicator area starting in column 21. Yes, with a few more changes to the

spec, IBM was able to line up the indicators just as they are in RPG/400. The next item is the Exception name. The Exception name facility is described in Chapter *******. Just as in RPG/400, it begins in position 30 right after the three sets of indicators.

Continuing the examination of what's next for output, we eventually bump into the area in which the RPGIV designers chose to put the printer controls. Since printers of today have way more capabilities than RPG could ever have imagined in the 1950's, the designers extended the power of spacing and skipping to limits few had imagined or requested. So, with RPGIV, you now get to specify expanded print controls way down in the O spec in positions 40 through 51.

Figure 12-14 RPGIV New PAREG Output Record Format

```
       6789112345678921234567893123456789412345678951 2
FMT O  OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.
035.00 OQPRINT    H    1P                          2 06
036.00 O         OR    OF                          2 06
```

Figure 12-15 RPG/400 PAREG Output Record Format

```
       678911234567892123456789312345678941234567895 12
FMT O  OName++++DFBASbSaN01N02N03Excnam...............
028.00 OQPRINT  H  206   1P
029.00 O       OR  206   OF
```

The definitions of spacing and skipping have not changed with RPGIV, but there are apparently no physical printer reasons for a programmer to use either skipping or spacing. Both are greatly enhanced and based on your printer with spooling, it may not even matter from a performance perspective. Spacing still refers to advancing one line at a time, and skipping refers to jumping from one print line to another lines. But RPG has gotten lots smarter inside as to how it executes these operations.

Because choosing spacing and skipping before and after can get messy and there are since there are lots more spacing options than ever before, let's examine some facts in this area that can help bring some real order out of it all. If spacing and skipping are specified for the same line, it would help to know what happens first?

Figure 12-16 Converted RPGIV Output Specs

```
       67891123456789212345678931234567894123456789512345678961234567
FMT O  O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.......................
035.00 OQPRINT    H    1P                     2 06
036.00 O         OR    OF                     2 06
FMT P  O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTfor
037.00 O                                           32 'THE DOWALLOBY COMPANY'
038.00 O                                           55 'GROSS PAY REGISTER BY'
039.00 O                                           60 'STATE'
040.00 O                       UDATE         Y     77
041.00 OQPRINT    H    1P                     3
042.00 O         OR    OF                     3
043.00 O                                            4 'ST'
044.00 O                                           13 'CITY'
045.00 O                                           27 'EMP#'
046.00 O                                           45 'EMPLOYEE NAME'
047.00 O                                           57 'RATE'
048.00 O                                           67 'HOURS'
049.00 O                                           77 'CHECK'
050.00 O          D    02NMR               1
051.00 O                                           46 'NO MATCHING MASTER'
052.00 O                       EMPNO               27
053.00 O                       EMPHRS        1     67
054.00 O          D    02 MR               1
055.00 O                       EMPSTA               4
056.00 O                       EMPCTY              29
057.00 O                       EMPNO               27
058.00 O                       EMPNAM              52
059.00 O                       EMPRAT        1     57
060.00 O                       EMPHRS        1     67
061.00 O                       EMPPAY        1     77
062.00 O          T    L1                  2  2
063.00 O                                           51 'TOTAL CITY  PAY FOR'
064.00 O                       EMPCTY              72
065.00 O                       CTYPAY        1B    77
066.00 O          T    L2                  0  2
067.00 O                                           51 'TOTAL STATE PAY FOR'
068.00 O                       EMPSTA              54
069.00 O                       STAPAY        1B    77
070.00 O          T    LR                  2
071.00 O                       TOTPAY        1     77
072.00 O                                           50 'FINAL TOTAL PAY'
```

Figure 12-17 RPG/400 Output Specs

```
028.00 OQPRINT  H  206   1P
029.00 O       OR  206   OF
030.00 O                                   32 'THE DOWALLOBY COMPANY'
031.00 O                                   55 'GROSS PAY REGISTER BY '
032.00 O                                   60 'STATE'
033.00 O                         UDATE Y   77
034.00 OQPRINT  H  3     1P
035.00 O       OR  3     OF
036.00 O                                    4 'ST'
037.00 O                                   13 'CITY'
038.00 O                                   27 'EMP#'
039.00 O                                   45 'EMPLOYEE NAME'
040.00 O                                   57 'RATE'
041.00 O                                   67 'HOURS'
042.00 O                                   77 'CHECK'
043.00 O        D 1      02NMR
044.00 O                                   46 'NO MATCHING MASTER'
045.00 O                         EMPNO     27
046.00 O                         EMPHRS1   67
047.00 O        D 1      02 MR
048.00 O                         EMPSTA     4
049.00 O                         EMPCTY    29
050.00 O                         EMPNO     27
051.00 O                         EMPNAM    52
052.00 O                         EMPRAT1   57
053.00 O                         EMPHRS1   67
054.00 O                         EMPPAY1   77
055.00 O        T 22     L1
056.00 O                                   51 'TOTAL CITY  PAY FOR'
057.00 O                         EMPCTY    72
058.00 O                         CTYPAY1B  77
059.00 O        T 02     L2
060.00 O                                   51 'TOTAL STATE PAY FOR'
061.00 O                         EMPSTA    54
062.00 O                         STAPAY1B  77
063.00 O        T 2      LR
064.00 O                         TOTPAY1   77
065.00 O                                   50 'FINAL TOTAL PAY'
```

The spacing and skipping operations are processed in the following sequence:

```
Skip before
Space before
Print a line
Skip after
Space after.
```

If the PRTCTL (new RPGIV printer control option) keyword is not specified on the file description specifications, an entry must be made in one of the following positions when the device is PRINTER:

```
40-42 (space before)
43-45 (space after)
46-48 (skip before)
49-51 (skip after)
```

If a space/skip entry is left blank, the function on the record line with the blank entry (such as space before or space after) does not occur.

If entries are made in positions 40-42 (space before) or in positions 46-51 (skip before and skip after) and no entry is made in positions 43 - 45 (space after), no space occurs after printing.

When PRTCTL is specified, it is used only on records with blanks specified in positions 40 through 51. If a skip before or a skip after a line on a new page is specified, but the printer is already on that line, the skip does not occur.

Considering that there were just two positions for skipping and one for spacing in RPG/400, it will be interesting to see if print programs for

printers of the future can ever take advantage of the vastly expanded printer capabilities in RPGIV.

RPG IV Field and Control Specification

As a sample of a field and control specification from the converted PAREG program, let's pick some print lines that have both variable detail time data (fields) and constant detail time data (literals.). Looking first at the RPG/400 code, statements 43 – 46 show exactly the type of print line that demonstrates the major change with RPGIV. This code snippet in RPG/400 form is shown in Figure 12-19 below. The corresponding RPGIV code is shown in Figure 12-18.

Figure 12-18 Converted RPGIV Output Field Specs

```
       6789112345678921234567893123456789412345678951234567896 12345670
FMT P  O..............N01N02N03Field++++++++++YB.End++PConstant/editword/DTfo
050.00 O          D    02NMR               1
051.00 O                                          46 'NO MATCHING MASTER'
052.00 O                       EMPNO              27
053.00 O                       EMPHRS        1    67
```

Figure 12-19 RPG/400 Output Field Specs

```
       678911234567892123456789312345678941234567895123456789612345
FMT P  O................N01N02N03Field+YBEnd+PConstant/editword
043.00 O        D 1      02NMR
044.00 O                                   46 'NO MATCHING MASTER'
045.00 O                         EMPNO     27
046.00 O                         EMPHRS1   67
```

Comparing statement 44 in RPG/400 to Statement 51 in RPGIV, if there were no template, the trained RPG programmer would not be able to tell them apart. With RPGIV, one noticeable change is that the end position length is now 5 positions ending in column 51 v. 4 positions in RPG/400 ending in column 43. Additionally, the constant area begins in position 53 v. 45 in RPG/400.

Looking at the field names in RPG/400 and RPGIV respectively, the six position space for EMPNO from statement 45 begins in position 32

whereas the 14 positions of space permit up to 10-character field names that begins anywhere from column 30 to 33 and end anywhere from column 39 to 43. IBM provides all this space for field names so that the programmer can indent subfields, thereby making the program more readable. The end position for the length of the field is column 51. With 5 positions available, the largest record position that can be defined in this space for a field is 65535. Of course, we wouldn't be expecting that field to be printed on a real print line any time soon.

The changes in the other lines of the PAREG program follow the same principals that we have outlined for all of the code that has been transitioned to RPGIV.

Summary

You now have seen how to code RPG Fixed logic cycle programs in both RPG/400 and RPGIV. From what you have learned in this chapter, you should already be able to conclude that with the expanded capabilities and minimal additional learning required for moving to RPGIV, there is no real technical reason, even for what some might call "legacy code" to stay in the RPG/400 environment for new program development..

The RPGIV Definition "D" Specification

The RPGIV Definition specifications is the most natural innovation to hit the RPG language from its conception. Just a walk down the many faces of RPG/400 input and you can readily see that IBM was running out of places for programmers to code the most innovative new notions to the language. The "D" spec was long overdue when it hit the streets in 1994. Though there are no "D" specifications required in the PAREG RPGIV version, this is the natural place to cover this most revealing RPGIV topic.

Many of us want to call it the Data Definition specification but that is not correct. IBM does not limit itself to data, though lots of data can be coded and coded logically using the "new" "D" spec. For example, the D spec can be used to define:

- ✓ Standalone fields

- ✓ Named constants
- ✓ Data structures and their subfields
- ✓ Prototypes
- ✓ Procedure interface
- ✓ Prototyped parameters

For more information on data structures and constants see Chapter ****. For more information on the more advanced RPGIV notions such as Prototypes, Procedure Interfaces, and Prototyped parameters, see Chapter ****.

Arrays and tables that formerly used the RPG/400 Extension "E" specification, which is covered in detail in Chapter **** can be defined as either a data-structure subfield or a standalone field. Definition specifications can appear in two places within a module or program: in the main source section and in a subprocedure. These notions are described in Chapter *****. Within the main source section, you define all global definitions.

A built-in function (BIF) can be used in the keyword field as a parameter to a keyword. It is allowed on the definition specification only if the values of all arguments are known at compile time. See BIFs in Chapter ****.

D Spec Keyword Continuation Line

Because RPGIV is so keyword oriented, it makes sense before we cover the columns of the "D" spe to show the "D" continuation spec. As you begin to code RPGIV, you will be looking for more space fro more "D" keywords and the continuationspec provides a nice and easy vehicle for you to accomplish this.

If additional space is required for keywords, the keywords field can be continued on subsequent lines as follows:

Position 6	Continuation line must contain a "D"
Positions 7 to 43	Continuation line must be blank
Positions 44 – 80	Continue the prior D spec keywords here

Posiitons 81 – 100 Comments

Figure 12-20 D Continuation Line Specification

```
*.. 1 .....4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
D............Keywords+++++++++++++++++++++++++++Comments++++++++++++
```

"D" Spec Continued Name Line

There are a number of surprises in RPGIV even if you already think you have an idea of what it is all about. One of the surprises is the "D" Spec Continued Name Line. Here's where this comes into play. A name that is up to 15 characters long can be specified in the Name entry of the definition specification without requiring continuation. Any name (even one with 15 characters or fewer) can be continued on multiple lines by coding an ellipsis (...) at the end of the partial name. A name definition consists of the parts identified in Table 12-21. The format for the spec line is shown in Figure 12-22

Table 12-21 Three Parts to Name Definition

Parts 1 to 3	Description
(1) Zero or more continued name lines	**Continued name lines are identified as having an ellipsis as the last non-blank character in the entry. The name must begin within positions 7 to 21 and may end anywhere up to position 77 (with an ellipsis ending in position 80). There cannot be blanks between the start of the name and the ellipsis character. If any of these conditions is not true, the line is parsed as a main definition line.**
(2) One main definition line,	**Contains name, definition attributes, and keywords. If a continued name line is coded, the Name entry of the main definition line may be left blank.**
(3) Zero or more keyword continuation	**Self explanatory**

lines

Figure *12-22 D Continuation Line Specification*

```
*.. 1 ...+... 2 ..... 7 ...+... 8 ...+... 9 ...+... 10
DContinuedName++++…++++++++++++++Comment s++++++++++++
```

D – Columns 7-21 Name

Use columns 7-21 of the D spec to supply the name for whatever you are defining. Besides the name, you may leave this position blank if the purpose is to define filler fields in data-structure subfield definitions, or an unnamed data structure in data-structure definitions.

The name can begin in any position in the space provided. Thus, indenting can be used to indicate the shape of data in data structures. For continued name lines, a name is specified in positions 7 through 80 of the continued name lines and positions 7 through 21 of the main definition line. As with the traditional definition of names, case of the characters is not significant. For an externally described subfield, a name specified here replaces the external-subfield name specified on the EXTFLD keyword.

For a prototype parameter definition, the name entry is optional. If a name is specified, the name is ignored. (A prototype parameter is a definition specification with blanks in positions 24-25 that follows a PR specification or another prototype parameter definition. See Chapter ****)

D – Column 22 External Definition

This column is used to identify a data structure or data-structure subfield as externally described. If a data structure or subfield is not being defined on this specification, then this field must be left blank.

The allowable entries for data structure are as follows

E Identifies a data structure as externally described: subfield

definitions are defined externally. If the EXTNAME keyword is not specified, positions 7-21 must contain the name of the externally described file containing the data structure definition.

Blank Program described: subfield definitions for this data structure follow this specification.

The allowable entries for data structure subfields are as follows

E Identifies a data-structure subfield as externally described. The specification of an externally described subfield is necessary only when keywords such as EXTFLD and INZ are used.

Blank Program described: the data-structure subfield is defined on this specification line.

D – Column 23 Type of Data Structure

When you are defining a data structure, code the type of data structure in column 23. This entry is used to identify the type of data structure being defined. If you are not defining a data structure then this space must be left blank.

The allowable entries are as follows:

Blank The data structure being defined is not a program status or data-area data structure; or a data structure is not being defined on this specification

S Program status data structure. Only one data structure may be designated as the program status data structure.

U Data-area data structure. RPG IV retrieves the data area at initialization and rewrites it at end of program. If the DTAARA keyword is specified, the parameter to the DTAARA keyword is used as the name of the external data area. If the DTAARA keyword is not specified, the name in positions 7-21 is used as the name of the external data area. If a name is not specified either by the DTAARA keyword, or by positions 7-21, *LDA (the local data area) is used as the name of the external data area.

D - Columns 24-25 Definition Type

Specify the type of definition that this D statement line represents. The allowable entries are as follows:

Blank The specification defines either a data structure subfield or a parameter within a prototype or procedure interface definition.

C The specification defines a constant. Position 25 must be blank.

DS The specification defines a data structure.

PR The specification defines a prototype and the return value, if any.

PI The specification defines a procedure interface, and the return value if any.

S The specification defines a standalone field, array or table. Position 25 must be blank.

Definitions of data structures, prototypes, and procedure interfaces end with the first definition specification with non-blanks in positions 24-25, or with the first specification that is not a definition specification. For a list of valid keywords, grouped according to type of definition, please refer to Figure ****. Data structures and constants are covered in Chapter ****. Procedures are covered in Chapter ****.

D Columns 26-32 (From Position)

Positions 26-32 may contain an entry only if the location of a subfield within a data structure is being defined. The allowable entries are as follows:

Blank A blank FROM position indicates that the value in the TO/LENGTH field specifies the length of the subfield, or that a subfield is not being defined on this

specification line.

nnnnnnn Absolute starting position of the subfield within a data structure. The value specified must be from 1 to 65535 for a named data structure (and from 1 to 9999999 for an unnamed data structure), and right-justified in these positions.

<u>Reserved Words:</u> Reserved words may also be specified. Reserved words for the program status data structure or for a file information data structure are allowed (left-justified) in the FROM-TO/LENGTH fields (positions 26-39). These special reserved words define the location of the subfields in the data structures. Reserved words for the program status data structure are *STATUS, *PROC, *PARM, and *ROUTINE. Reserved words for the file information data structure (INFDS) are *FILE, *RECORD, *OPCODE, *STATUS, and *ROUTINE. This information is given for completeness only. These keywords are not covered in any detail in this book.

Columns 33-39 (To Position / Length)

The allowable entries and their explanations follows:

Blank If columns 33-39 are blank: a named constant is being defined on this specification line, or the standalone field, parameter, or subfield is being defined LIKE another field, or the standalone field, parameter, or subfield is of a type where a length is implied, or the subfield's attributes are defined elsewhere, or a data structure is being defined. The length of the data structure is the maximum value of the subfield To-Positions.

nnnnnnn Columns 33-39 may contain a (right-justified) numeric value, from 1 to 65535 for a named data structure (and from 1 to 9999999 for an unnamed data structure)If the From field (position 26-32) contains a numeric value, then a numeric value in this field specifies the absolute end position of the subfield within a data structure. If the From field is blank, a numeric value in this field

specifies : the length of the entire data structure, or the length of the standalone field, or the length of the parameter, or the length of the subfield. Within the data structure, this subfield is positioned such that its starting position is greater than the maximum to-position of all previously defined subfields in the data structure. Padding is inserted if the subfield is defined with type basing pointer or procedure pointer to ensure that the subfield is aligned properly.

+|-nnnnn This entry is valid for standalone fields or subfields defined using the LIKE keyword. The length of the standalone field or subfield being defined on this specification line is determined by adding or subtracting the value entered in these positions to the length of the field specified as the parameter to the LIKE keyword.

Reserved Words : Reserved words may also be specified in these columns. If columns 26-32 are used to enter special reserved words, this field becomes an extension of the previous one, creating one large field (positions 26-39). This allows for reserved words, with names longer than 7 characters in length, to extend into this field. See Columns 26-32 (From Position), 'Reserved Words' above.

D - Column 40 (Internal Data Type)

This entry allows you to specify how a standalone field, parameter, or data-structure subfield is stored internally. This entry pertains strictly to the internal representation of the data item being defined, regardless of how the data item is stored externally on disk if it is stored on disk or tape or other media. To define variable-length character, graphic, and UCS-2 formats, you must specify the keyword VARYING to have it take effect; otherwise, the format will be fixed length. The allowable entries and explanations are as follows:

Blank When the LIKE keyword is not specified: If the decimal positions entry is blank, then the item is defined as character If the decimal positions entry is not blank, then the item is defined as packed numeric if it is a standalone field or parameter; or as

zoned numeric if it is a subfield.
Note: The entry must be blank when the LIKE keyword is specified.

A Character (Fixed or Variable-length format)

B Numeric (Binary format)

C UCS-2 (Fixed or Variable-length format)

D Date

F Numeric (Float format)

G Graphic (Fixed or Variable-length format)

I Numeric (Integer format)

N Character (Indicator format) |

O Object

P Numeric (Packed decimal format)

S Numeric (Zoned format)

T Time

U Numeric (Unsigned format)

Z Timestamp

***** Basing pointer or procedure pointer.

The simple internal data types that we cover in this book are described in Chapter ****.

D -- Columns 41-42 (Decimal Positions)

Use Columns 41-42 to indicate the number of decimal positions in a numeric subfield or standalone field. If the field is non-float numeric, there must always be an entry in these positions. If there are no decimal positions enter a zero (0) in position 42. For example, an integer or unsigned field (type I or U in position 40) requires a zero for this entry.

The allowable entries and explanations are as follows:

Blank The value is not numeric (unless it is a float field) or has been defined with the LIKE keyword.

0-30 Decimal positions: the number of positions to the right of the decimal in a numeric field. This entry can only be supplied in combination with the TO/Length field. If the TO/Length field is blank, the value of this entry is defined somewhere else in the program (for example, through an externally described data base file).

D -- Column 43 Reserved

This column is reserved for future use.

D -- Columns 44-80 (Keywords)

Columns 44 to 80 are provided for definition specification keywords. Keywords are used to describe and define data and its attributes. This area is used to specify any keywords necessary to fully define the field. If you don't have enough room for all the keywords you need, use the continuation form which was described at the beginning of this section.

D -- Columns 81-100 Comments

These columns can be used for optional comments.

D Specification Keywords.

From column 7 to 42 the D specification provides a standard means of defining data and other items to the RPGIV compiler. However, there are lots more options that the D specification handles besides what can be specified within the 7 – 42 column boundaries. As with the other RPGIV specifications, these other items are enabled through the use of keywords in positions 44 – 80 of the D spec. Table 12-23 provides a look at these powerful keywords as well as an explanation as to what they are all about.

Table 12-23 D Specification Keywords & Parameters

Keyword	Description
ALIGN	**The ALIGN keyword is used to align float, integer, and unsigned subfields. When ALIGN is specified, 2-byte subfields are aligned on a 2-byte boundary, 4-byte subfields are aligned on a 4-byte boundary and 8-byte subfields are aligned on an 8-byte boundary. Alignment may be desired to improve performance when accessing float, integer, or unsigned subfields.**
ALT	**(array_name) The ALT keyword is used to indicate that the compile-time or pre-runtime array or table is in alternating format.**
ALTSEQ	**(*NONE) When the ALTSEQ(*NONE) keyword is specified, the alternate collating sequence will not be used for comparisons involving this field, even when the ALTSEQ keyword is specified on the control specification.**
ASCEND	**The ASCEND keyword is used to describe the sequence of the data in arrays, tables, or prototyped parameters**
BASED	**(basing_pointer_name) When the BASED keyword is specified for a data structure or standalone field, a basing pointer is created using the name specified as the keyword parameter. This basing pointer holds the address (storage location) of the based data structure or standalone field being defined. In other words, the name specified in positions 7-21 is used to refer to the data stored at the location contained in the basing pointer.**
CCSID	**(number | *DFT) This keyword sets the CCSID for graphic and UCS-2 definitions. The number must be an integer between 0 and 65535. It must be a valid graphic or UCS-2 CCSID value.**
CLASS	**(*JAVA:class-name) This keyword indicates the class**

	for an object definition. Class-name must be a constant character value
CONST	{(constant)} The CONST keyword is used to specify the value of a named constant or to indicate that a parameter passed by reference is read-only.
CTDATA	The CTDATA keyword indicates that the array or table is loaded using compile-time data. The data is specified at the end of the program following the ** or **CTDATA(array/table name) specification.
DATFMT	(format{separator}) The DATFMT keyword specifies the internal date format, and optionally the separator character for a Date; standalone field; data-structure subfield; prototyped parameter; or return value on a prototype or procedure-interface definition
DESCEND	The DESCEND keyword is used to describe the sequence of the data in arrays, tables, or prototyped parameters
DIM	(numeric_constant) The DIM (Dimension) keyword defines the number of elements in an array; a table; a prototyped parameter; or a return value on a prototype or procedure-interface definition. The numeric constant must have zero (0) decimal positions. It can be a literal, a named constant or a built-in function.
DTAARA	{(data_area_name)} The DTAARA keyword is used to associate a standalone field, data structure, data-structure subfield or data-area data structure with an external data area. You can create three kinds of data areas: (1) *CHAR Character, (2) *DEC Numeric, and (3) *LGL Logical. You can also create a DDM data area (type *DDM) that points to a data area on a remote system.
EXPORT	{(external_name)} The EXPORT keyword allows a globally defined data structure or standalone field defined within a module to be used by another module in the program. The storage for the data item is allocated in the module containing the EXPORT definition. The external_name parameter, if specified, must be a character literal or constant. The EXPORT keyword on the definition specification is used to export data items and cannot be used to export procedure names. To export a procedure name, use the EXPORT keyword on the procedure specification.
EXTFLD	(field_name) The EXTFLD keyword is used to rename a subfield in an externally described data structure.

	Enter the external name of the subfield as the parameter to the EXTFLD keyword, and specify the name to be used in the program in the Name field (positions 7-21).
EXTFMT	**(code) The EXTFMT keyword is used to specify the external data type for compile-time and prerun-time numeric arrays and tables. The external data type is the format of the data in the records in the file. This entry has no effect on the format used for internal processing (internal data type) of the array or table in the program. The values specified for EXTFMT will apply to the files identified in both the TOFILE and FROMFILE keywords, even if the specified names are different.** **The possible values for the parameter are:** **B The data for the array or table is in binary format.** **C The data for the array or table is in UCS-2 format.** **I The data for the array or table is in integer format.** **L The data for a numeric array or table element has a preceding (left) plus or minus sign.** **R The data for a numeric array or table element has a following (right) plus or minus sign.** **P The data for the array or table is in packed decimal format.** **S The data for the array / table is in zoned decimal format.** **U The data for the array or table is in unsigned format.** **F The data for the array or table is in float numeric format.**
EXTNAME	**(file_name{:format_name}) Use the EXTNAME keyword to specify the name of the file which contains the field descriptions used as the subfield description for the data structure being defined. The file_name parameter is required. Optionally a format name may be specified to direct the compiler to a specific format within a file. If format_name parameter is not specified the first record format is used. If the data structure definition contains an E in position 22, and the EXTNAME keyword is not specified, the name specified in positions 7-21 is used**
EXTPGM	**(name) Use the EXTPGM keyword to indicate the external name of the program whose prototype is being defined.**
EXTPROC	**({*CL|*CWIDEN|*CNOWIDEN|{*JAVA:class-name:}}name) There are a number of formats as noted that can be used with this keyword. The EXTPROC**

	keyword indicates the external name of the procedure whose prototype is being defined.
FROMFILE	(file_name) Use the FROMFILE keyword to specify the input data file for the prerun-time array or table.. The FROMFILE keyword must be specified for every prerun-time array or table defined/used in the program.
IMPORT	{(external_name)} Use the IMPORT keyword to specify that storage for the data item being defined is allocated in another module, but may be accessed in this module.
INZ	{(initial value)} The INZ keyword initializes the standalone field, data structure, |data-structure subfield, or object to the default value for its data type or, |optionally, to the constant specified in parentheses.
LIKE	(name) Use the LIKE keyword to define an item like an existing |one. When you specify the LIKE keyword, the item you are defining takes on the length and the data format of the item specified as the parameter. You may define standalone fields, prototypes, parameters, and data-structure subfields using this keyword.
LIKEDS	(data_structure_name) Use the LIKEDS keyword to define a data structure, prototyped return value, or prototyped parameter like another data structure. The subfields of the new item will be identical to the subfields of the other data structure.
NOOPT	The NOOPT keyword indicates that no optimization is to be performed on the standalone field, parameter or data structure for which this keyword is specified. Specifying NOOPT ensures that the content of the data item is the latest assigned value. This may be necessary for those fields whose values are used in exception handling.
OCCURS	(numeric_constant) Use the OCCURS keyword to specify (in the num constant parm) the number of occurrences of a multiple-occurrence data structure.
OPDESC	The OPDESC keyword specifies that operational descriptors are to be passed with the parameters that are defined within a prototype.
OPTIONS	(*NOPASS *OMIT *VARSIZE *STRING *RIGHTADJ) The OPTIONS keyword is used to specify one or more parameter passing options: (1) Whether a parameter must be passed (2) Whether the special value *OMIT can be passed for the parameter passed by reference. (3) Whether a parameter that is

	passed by reference can be shorter in length than is specified in the prototype. (4) Whether the called program or procedure is expecting a pointer to a null-terminated string, allowing you to specify a character expression as the passed parameter. The single parameter passed with this keyword can have different values.
OPTIONS	**(*NOPASS) Use this option so that the called program or procedure will simply function as if the parameter list did not include that parameter.**
OPTIONS	**(*OMIT) Use this to allow the value *OMIT for that parameter. *OMIT is only allowed for CONST parameters and parameters which are passed by reference**
OPTIONS	**(*VARSIZE) Use this for parameters passed by reference that have a character, graphic, or UCS-2 data type, or that represent an array of any type.**
OPTIONS	**(*STRING) Use this for a basing pointer parameter passed by value or by constant-reference. You may either pass a pointer or a character expression.**
OPTIONS	**(*RIGHTADJ) Use this for a CONST or VALUE parameter in a function prototype. In this case, the character, graphic, or UCS-2 parameter value is right adjusted.**
OVERLAY	**(name{:pos | *NEXT) Use the OVERLAY keyword to overlay the storage of one subfield with that of another subfield, or with that of the data structure itself. This keyword is allowed only for data structure subfields. The Name-entry subfield overlays the storage specified by the name parameter at the position specified by the pos parameter. If pos is not specified, it defaults to 1.**
PACKEVEN	**Use the PACKEVEN keyword to indicate that the packed field or array has an even number of digits. The keyword is only valid for packed program-described data-structure subfields defined using FROM/TO positions.**
PERRCD	**(numeric_constant) Use the PERRCD keyword to specify the number of elements provided per record for a compile-time or a prerun-time array or table.**
PREFIX	**(prefix{:nbr_of_char_replaced}) Use the PREFIX keyword to change the field names of all of the fields from a particular external data structure so there is no conflict with fields already coming in to the program. Specify a character string or character literal, which is to**

	be prefixed to the subfield names of the externally described data structure being defined. In addition, you can optionally specify a numeric value to indicate the number of characters, if any, at the beginning of the existing name to be replaced.
PROCPTR	**The PROCPTR keyword defines an item as a procedure pointer. The internal Data-Type field (position 40) must contain a *.**
QUALIFIED	**Use the QUALIFIED keyword to specify that the subfields of a data structure will be accessed by specifying the data structure name followed by a period and the subfield name.**
STATIC	**Use the STATIC keyword to specify that a local variable is stored in static storage and thereby hold its value across calls to the procedure in which it is defined or to specify that a Java method is defined as a static method.**
TIMFMT	**(format{separator}) Use the TIMFMT keyword to specify an internal time format, and optionally the time separator, for type Time: standalone field; data-structure subfield; prototyped parameter; or return value on a prototype or procedure-interface definition.**
TOFILE	**(file_name) Use the TOFILE keyword to specify a target file to which a prerun-time or compile-time array or table is to be written.**
VALUE	**Use the VALUE keyword to indicate that the parameter is passed by value rather than by reference. Parameters can be passed by value when the procedure they are associated with are called using a procedure call.**
VARYING	**Use the VARYING keyword to indicate that a character, graphic, or UCS-2 field, defined on the definition specifications, should have a variable-length format. If this keyword is not specified for character, graphic, or UCS-2 fields, they are defined as fixed length.**

Chapter 13

RPG Operations

How RPG Gets Things Done!

In RPG/400 and RPGIV, if you are not fixed on the fixed cycle, you will quickly see that almost all of the action occurs in the calculations (calc) specifications. Moreover, the heart of the CALC spec as you learned earlier in this book is the operation area in which you place the specific op-code to tell RPG which operation to conduct on your behalf. In Chapter 9 we examined the CALC spec and how it is formatted for fill-in the blank operations. In this chapter, we take a real close look at those operations in their RPG/400 format. Better than 90% of these operations are also usable with RPGIV. We'll get to RPGIV calculations in Chapter 15***.

There are many different operations available and they are implemented via the codes that a programmer can specify in the op-code column (28 in RPG/400 and 26 in RPGIV) of the CALC form. RPG operations can be classified in a number of ways. The various op-code charts and tables shown in this chapter starting with Figure 7-1 provide a complete picture of all of the operations available in RPG and RPGIV. The first chart, shown in Figure 7-1 is a summary of all operations by type and it shows the classification and a number of specific the op code examples that are available in each classification.

This chapter breaks tutorial-like stride temporarily to present the complete set of RPG/400 operations that you can deploy in calculation specifications. It covers major language elements in RPG/400 to familiarize the student with the full capabilities of the RPG/400 language. It would be impossible to teach all of this with examples and text in an introductory book. By showing all of the RPGIV operations and by default over 90% of the RPGIV operations in this chapter, it sets the stage for further work with examples starting in Chapter 13.

In most cases, the operations for RPG/400 are the same in RPGIV, other than perhaps an expanded columns in RPGIV. So, to repeat somewhat, the intent of this chapter is to demonstrate the programming capabilities of the language by examining the operations that are found in both

modern forms of the RPG language – RPG/400 and RPGIV. By doing this, you can see and learn operations in one form (RPG/400 in this chapter) and unless otherwise noted, the same operation is available in just about the same form in RPGIV. In other words, when the reader learns an operation in RPG/400, the corresponding RPGIV operation is understandable by default.

You are about to see that in this chapter, the author has taken every RPG operation and has condensed its explanation into something that is germane and understandable without losing its meaning in simplicity. In fact, the RPG charts in this chapter are so comprehensive that they can be used as a reference for how to code operations without having to understand all of the intricacies and options that are provided in IBM's four RPG manuals.

When you become an RPG programmer instead of an RPG learner, because of these operation charts and other unique function charts, this book will continue to provide value to you to quickly recap your learning. However, when you are moving deeper into the language, you will be pleased that IBM's entire library for the AS/400 is included on-line. To get to the IBM RPG manuals, first RPG/400, then RPGIV, take the following links:

RPG/400 IBM Links

www.iseries.ibm.com
>>Support
>>iSeries Information Center
>>(your language – such as English) V5R1
>>Left frame > Programming
>>RPG
>>Right Frame – Scroll to bottom
>>Look for

RPG/400

1. **RPG/400 Reference - PDF**
   Also available in HTML version

2. **RPG/400 User's Guide - PDF**

Also available in HTML version

ILE RPG (RPGIV) IBM Links

www.iseries.ibm.com
>>Support
>>iSeries Information Center
>>V5R4
>>Printable PDFs and Manuals
>>Scroll down almost to the bottom of the list:
>> You will find these valuable manuals:

1. WebSphere Development Studio:
ILE RPG Programmer's Guide
Programming Manual

2. WebSphere Development Studio:
ILE RPG Reference

With this book and IBM's manuals, you have all the tools you need to become and to be an effective RPG programmer. For those enjoying the tutorial nature of this book, who would prefer not to absorb al of the language operations before continuing, I would recommend reading through the RPG BASIC Operations Set, browsing through the rest of the operations through Table 13-4. Then, you might want to check out the handy operations summary chart in Table 13-12 and walk quickly through RPGIV specific operations in Chapter 14 before continuing in tutorial mode in Chapter 15. You will find that the PAREG program that we have been working with so far in this book goes through another metamorphosis to be able to provide all of its function without the use of the RPG fixed logic cycle – no level time and no MR match fields. There's lots to do in Chapter 15. See you there

RPG Operations

To introduce you to the plethora of operations in RPG, let's first take a look at Table 13-1. This table provides a quick summary of the types of

operations that you will find in RPG along with a number of representative operations as shown by their op-codes. The detail for these types of operations is provided in the subsequent tables.

Table 13-1 Types of Operations & Op-Codes

Types Of Operations	Representative Op- Codes
Basic Operations	**ADD– add two numbers** **BEGSR– begin a subroutine** **MULT-- multiply two numbers**
Compare & Branch Operations / Subroutines	**GOTO -- Go to** **EXSR – Execute a subroutine** **CABXX -- Compare and branch if EQ,GT… .**
Data Manipulation Operations	**CLEAR– houseclean a structure** **BITON– Turn bits on** **RESET– Reset a structure**
Database & Device File Operations	**READ—read a record** **REDPE– read a prior equal record** **SETLL-- set file index position EXFMT– write then read a screen**
Data Structure, Table, Array, String Operations	**LOKUP—find item in table/array** **MOVEA– move data to-from array**
Program Control Operations	**DUMP—take a dump** **ACQ-- acquire a device in program**
Structured Operations	**DOWXX—do while EQ, GT etc.** **SELEC-- Select** **ITER—Repeat Do loop**
RPGIV Free Form Syntax Calculation spec	**Eval—evaluate expression** **EvalR-evaluate expression, rt adjust**
RPGIV Free Form Operations	**/FREE compiler directive signifies free form RPGIV is coming** **/END-FREE self explanatory**

Basic Operations - Including Arithmetic

It can be readily argued that COBOL is the most English-like language ever devised. With its lengthy field names, data attributes are able to be naturally explained well within the confines of the variable name itself. As you examine the list of operations in 13-1, you can see that the five character operation names available in RPG/400, though reasonably easy to understand are no match for COBOL.

RPG operations are logical and they are somewhat English-like. Once read and understood, they are difficult to forget. The op-codes are very unlike the computer science languages operations available in languages such as C. Moreover, it can be argued that despite their conciseness, the RPG operations resemble COBOL, the *C*ommon ***B*usiness** *O*riented *L*anguage more than any other language.

Clearly RPG is much more concise and thus much more precise than COBOL, PL/1 and any other business language. In fact, compared to mostly all languages, RPG is the most concise, rendering many less statements per equivalent program than all of the other languages du jour. There is no room for verbosity in RPG operations. The language iis built to solve business problems. In earlier chapters we demonstrated the utility of RPG for basic input and output and report writing facilities. Now, let us move from the overview in Table 13-1 to a look at a number of the most elemental operations available in the RPG language.

For your review, the columns of the CALC spec are as follows:

Column 6	**C for calc spec**
Columns 7- 17	**Level and detail conditioning indicators**
Column 18- 27	**Factor 1**
Column 28- 32	**Operation Code**
Column 33- 42	**Factor 2**
Column 43- 48	**Result Field**
Column 49- 53	**Field Length- 3, Decimals-1, Op Extender-1**
Column 54- 58	**HI LO EQ Resulting Indicators**

As you can see in Table 13-2, RPG Basic Operations. The operation is shown on the left, followed by the provided function on the right. The first area within the provided function is a one phrase shortened synopsis of the operation. This is then followed by a detailed description of the function and purpose of the operation and how it needs to be coded in RPG/400 to achieve the desired result. To prepare you better for the operations journey you are about to take, we begin this tour with a concise definition of the calculation specifications for RPG/400. The operation codes that we explain in the many tables that follow place the operation in column 28 of the calc spec.

Table 13-2 RPG Basic Operations

Operation	Provided Function
ADD	**Add two factors to produce a result.** The ADD operation adds Factor 1 to Factor 2 and places the sum in the Result field. If Factor 1 is not specified, the contents of Factor 2 are added to the Result field and the sum is placed in the Result field. Factor 1 and Factor 2 and the Result field must be defined as numeric.
CHAIN	**Random Retrieval from a File.** The CHAIN is a very powerful operation in RPG and is used in most programs for record retrieval by relative record # or by key. The operation retrieves a record from a full procedural file (F in position RPG/400 position 16) of the file description specifications), and places the data from the record into the input fields. The search

	argument is provided in Factor 1. It must contain the key or the relative record number used to retrieve the record. Factor 2 specifies the file or record format name (externally described files) that is to be read. For a WORKSTN subfile, the CHAIN operation retrieves a subfile record. Files specified as input, read all records without locks and position 53 in RPG/400 must be blank. **To lock all records**, the file must be specified as update and RPG/400 position 53 is blank. Specify an "N" in RPG/400 53 so that no lock should be placed on a record when it is read. **The HI positions** 54 and 55 must contain an indicator that is set on if no record in the file matches the search argument. The LO positions can contain an indicator that is on if the CHAIN operation is not completed successfully Error). The EQ positions must be blank. A successful chain repositions the file cursor such that if it is followed by a READ operation the next sequential record following the retrieved record is read. If an update (on the calculation or output specifications) is done on the retrieved record after a successful CHAIN operation and before other access to that file, the last record retrieved is updated.
DEBUG	**Shows internal variables and indicators.** Output is to A print device or a disk file at specific points in calculations
DELET	**Delete a specific record in a database.** If no search argument is placed in Factor1, this operation deletes the last locked record retrieved from the file or record format specified in Factor 2. If a search argument is specified and the record is found, it is deleted. A status indicator area is provided in the HI (not found) and LO (error) indicator areas to show the results of the DELET. When indicators are specified and none are turned on, the delete is successful.
DIV	**Divide two numbers, produce a result and a remainder.** The DIV operation divides Factor 1 by Factor 2. Without Factor 1 specified, it divides the result field by Factor 2. The quotient (result) is placed in the result field. If Factor 1 is 0, the result of the divide operation is 0. Any remainder resulting from the divide

	operation is lost unless the move remainder (MVR) (See this table) operation is specified as the next operation. If move remainder is the next operation, the result of the divide operation cannot be half-adjusted
DSPLY	**Display function - show values immediately**. The DSPLY operation allows the program to communicate with the display work station that requested the program. The operation can display a message from Factor 1 and send a value in the result field into which can be keyed a response. Specify a literal of a field in factor 1 to be used as the message to be displayed. The result field is optional. If specified, the user response is placed in it and the program can take immediate action based on this communication with the user. The result field can contain a field name, a table name, or an array element the contents of which are displayed and into which the response is placed. If no data is entered, the result field is unchanged. The DSPLY can also be used as a handy and quick interactive debugger because it does not mess up printouts or displays to get its job done.
EXCPT	**Calculation Time Exception Output.** This program described file operation initiates immediate output to internally described disk or printer files. Output triggered for this operation is coded with a special "E" designation for exception output record headers in place of the RPG cycle's typical H or D records. The lines of output can be conditioned or unconditioned using record indicators. The EXCPT also has a special name facility that can be specified in Factor 2. When the EXCPT is executed with an except name, only the records with that particular name in the output record area are written to a printer or a disk file.
EXFMT	**Write/then Read Interactive Screen Format.** This powerful operation is a combination put screen, wait, and get screen operation. The format name specified in Factor 2 is defined in a display file that is typically created using the Screen Design Aid. A display file can have multiple record formats (screen panels) defined. A screen panel with input fields can be sent using the EXFMT operation, specifying the panel name, to a

	display. The program then waits for the user to enter data and signify they are finished. The user provides input and presses Enter or a function key. Control is then returned to the statement in the RPG program following the EXFMT.
MOVE	**Move data field right to left.** This operation moves each corresponding character from the Factor 2 field to the Result field starting with the rightmost character and stopping with the leftmost character in Factor 2.
MOVEL	**Move data field left to right.** This operation moves each corresponding character from the Factor 2 field to the Result field starting with the leftmost position to the leftmost position and stopping with the last character in Factor 2.
MOVEA	**Move Array – Array to Field and Vice Versa.** This operation moves each corresponding character from the array or field specified in Factor 2 field to the field or array specified in the result field -- starting with the leftmost position of the field or array to the leftmost position and stopping with the last character in Factor 2.
MULT	**Multiply one field or value by another and store the result.** Factor 1 is multiplied by Factor 2 and the product is placed in the result field. The Result field must be large enough to hold it. If Factor 1 is not specified, Factor 2 is multiplied by the Result field and the product is placed in the Result field. The fields in Factor 1, Factor 2, and the Result field must be defined as numeric. You can specify half adjust (position 53 in RPG/400).
MVR	**Move the remainder after a divide operation into a field.** This operation moves the remainder from the previous DIV operation to a separate field named in the result field. Factor 1 and Factor 2 must be blank. The MVR operation must immediately follow and be processed after the DIV operation. The MVR operation Be careful with conditional operations surrounding an MVR operation. If the MVR operation is processed before the DIV operation, undesirable results occur.

READ	**Read a record from a file or format.** The READ operation reads the current record, from a procedural file (identified by an F in position 16 of the RPG/400 file description specifications). Factor 2 must contain the name of a file. A record format name in Factor 2 is allowed only with an externally described file (E in pos. 19 of the RPG/400 F spec. A READ-by- format-name operation will receive a different format than the one you specified in factor 2. If so, the READ operation ends in error. This operation is most often associated with database files but it also works with other files including WORKSTN files in which the record format name on the READ. In database operations, specify an indicator in the EQ indicator position to have the READ operation check for end of file (no more records left to read). The indicator turns on when the file is at end.
SQRT	**Take the square root of a number and store it in a field**. The SQRT operation derives the square root of the field named in Factor 2 and it places it in the Result field. If the value of the factor 2 field is zero, the Result field value is also zero. If the value of the Factor 2 field is negative, the RPG/400 exception/error handling routine receives control.
SUB	**Subtract one field from another and store the result.** Factor 2 is subtracted from Factor 1 and the difference is placed in the result field. If Factor 1 is not specified, the contents of Factor 2 are subtracted from the contents of the Result field and the results are stored in the Result field. Indicators specified in the HI, LO, EQ resulting indicator area are turned on depending on a +, -, or 0 value in the result field.
TIME	**Get time of day and store in a field.** The TIME operation accesses the system time of day and, if specified, the system date at any time during program processing. The system time is based on the 24-hour system clock. The result field must specify the name of a 6-, 12- or 14-digit numeric field (no decimal positions) into which the time of day or the time of day and the system date are written. To access the

	time of day only, specify the result field as a 6-digit numeric field. To access both the time of day and the system date, specify the result field as a 12- (2-digit year portion) or 14-digit (4-digit year portion) numeric field. The time of day is always placed in the first six positions of the result field in the following format: hhmmss (hh=hours, mm=minutes, and ss=seconds)
UPDAT	**Modify Existing Record** The UPDAT operation rewrites (updates) the last record retrieved for processing from an update disk file or subfile. Data changes that occurred through normal processing will be made to the updated record. No other file operation should be performed between the input operation that retrieved the record and the UPDAT operation. Specify the file or record format name in factor 2 for the database file to be updated. For externally defined files, the record format name must be used with this operation. UPDAT follows a read type operation. Data in the record's fields are updated in program memory and then the new contents of the record are written over the former contents of the record in the database.
WRITE	**Create new DB Records (Add Records) or write a format to a display screen.** The WRITE operation writes a new record to a file. In essence, it creates new records. Factor 2 must contain the name of a file or a record format name for an externally described file. A file name in factor 2 is required with a program described file. In this case, a data structure must be defined in for the result field. For program described files, the record is written directly from the data structure to the file. The result field must be blank if factor 2 contains a record format name. The Write operation can also be used to send a panel (screen) to a display station.
XFOOT	**Sums the elements of an array.** Powerful operator that adds each of the elements of an array to create a total and it stores the total in a result field.
Z-ADD	**Zero and Add.** Zeroes out a result field first and then adds the content of the Factor 2 field or constant to the zeroed total. Similar to a numeric MOVE operation. Indicators specified in the resulting HI, LO, EQ area are

	turned on according to their +, - 0, result.
Z-SUB	**Zero and Subtract** Zeroes out a result field first and Then subtracts the content of a Factor 2 field or a constant from the zeroed result creating in most cases a negative value. Has the same arithmetic result as subtracting a number from zero to create its negative. Indicators specified in the resulting HI, LO, EQ area are turned on according to their +, - 0, result.

Compare & Branch / Subroutine Operations

The basic operations in Table 13-2 consist of mostly database and arithmetic operators. Though you definitely need a computer with nice healthy disk drives for database, the arithmetic operations in Table 13-2 are done just as handily with a calculator. Besides database capability, the major factor which differentiates computers from calculators is logic. The logic of a computer machine provides the ability to compare values, and based on those values take different courses of actions. The term used for taking those "different courses of action" is called branching. Through comparisons of values and branches which alter the sequential pattern of instructions, computer systems achieve a somewhat human-like, logical capability that enable them to far surpass the capabilities of the most advanced calculator.

As a language known for providing an easy means to quantify business rules for programmed decision making, RPG is well equipped for the tasks necessary to provide programs the compare and branch instructions necessary to run the business on the AS/400 or System i5. Table 13-4 below summarizes all of the compare and branch operations that an RPG programmer their disposal. As you examine the table, it will help you in deciphering the meaning of the operations that provide the decision making capabilities for RPG. This notion also applies to the operations listed under Structured Operations as shown in Table 13-8.

A number of compare and branch and structured decision operations are listed with XX as the last two characters of the operations. XX is a

convenient means of specifying the type of test that you can call upon using the operation. For example, the IFXX is much more meaningful if it is coded as IFEQ (If Equal) or IFGT (IF Greater Than). The meaning of these operation suffixes are shown in Table 13-3.

.

Table 13-3 XX Operation Meanings

XX	Meaning
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Unconditional processing -- (CAS).

Table 13-4 Compare & Branch Operations

Operation	Provided Function
ANDxx	**And if another condition is true, then...** This operation links two operations together. If you specify this optional operation, it must immediately follow a ANDxx, DOUxx, DOQxx, IFxx, ORxx, or WHxx. With ANDxx, you can specify a complex condition for the DOUxx, DOWxx, IFxx, and WHxx operations. The ANDxx operation has higher precedence than the ORxx operation. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See COMP in this table.
BEGSR	**Beginning of Subroutine.** Defines the beginning point of a subroutine. Subroutine ends with an ENDSR opcode. Factor 1 contains the name of the subroutine.
COMP	**Compare Two values and set on status indicators** Based on the relationship of the first value (field or literal) in Factor 1 to the second value (Factor 2). Status indicators can be specified for HI, LO, EQ. The COMP operation compares Factor 1 with Factor 2. As a result of the comparison, indicators are set on in RPG/ 400 and RPG IV as follows: HI – if Factor 1 is greater than Factor 2; LO – if Factor 1 is less than Factor 2;

	EQ – if Factor 1 equals Factor 2. When specified, the resulting indicators are set on or off each time through the operation to reflect the results of the latest compare.
CABxx	**Compares two values and branches if the tested condition is satisfied.** The condition, xx = EQ or GT etc is specified as part of the op-code. The branch-to label is taken if the condition is satisfied. This combination operation first compares Factor 1 with factor 2. If the condition specified by xx in the op-code is true, the program branches to the TAG operation associated with the label specified in the result field. Otherwise, the program continues with the next operation in the sequence.
CASxx	**Conditionally Invoke Subroutine on Compare.** Very similar to the CABXX operations. Instead of a straight go to like branch, a specified subroutine is invoked if the conditions are met. This operation allows you to conditionally select a subroutine for processing. The selection is based on the relationship between Factor 1 and Factor 2, as specified by the "xx" portion of the op-code.. If the relationship denoted by xx exists (true) between Factor 1 and Factor 2, the subroutine specified in the result field is processed.
DO	**Starts a straight Do loop – Loop ends with ENDDO** The DO operation begins a group of operations and indicates the number of times the group will be processed. You specify an index field, a starting value, and a limit value. You also specify an ENDDO statement to mark the end of the DO group. Specify the numeric starting value in Factor1 with no decimal positions. If you do not specify Factor 1, the starting value is 1. Specify the limit value in Factor 2. If you do not specify a limit, the default is 1. Specify the current index value in the Result field. (The loop does not have to begin with "1.") Any value in the index field is replaced by Factor 1 when the DO operation begins. Factor 2 of the associated ENDDO operation specifies the value to be added to the index field, otherwise, 1 is the default.
DOUxx	**Starts a Do until loop xx = the relationship**. Loop

	Ends with ENDDO Do until xx is true. DO until and DO while are similar operations. The Do until (DOUxx) operation begins a group of operations you want to process more than once (but always at least once). The group of instructions in the loop are sandwiched between the statement defining the beginning of the loop (DOUxx) and the statement defining the end of the loop (ENDDO). Factor 1 and Factor 2 are required. On the DOUxx statement, you indicate a relationship xx. However, you can specify a more complex condition by pacing ANDxx or ORxx statements immediately following the DOUxx. The instructions within the DO loop are processed once, and then the group is repeated until the ENDDO test. At the end of the loop, when the ENDDO is processed, if the relationship xx exists between Factor 1 and Factor 2 or the specified condition exists, the DO group is finished and control passes to the next calculation operation after the ENDDO statement.
DOWxx	**Starts a Do while loop xx = the relationship.** Loop ends with ENDDO Do while xx continues to be true. DO while and DO until are similar operations. The Do while (DOWxx) operation begins a group of operations you want to process while the relationship xx exists between Factor 1 and Factor 2. The group of instructions in the loop are sandwiched between the statement defining the beginning of the loop (DOWxx) and the statement defining the end of the loop (ENDDO). Factor 1 and Factor 2 are required. On the DOWxx statement, you indicate a relationship xx. However, you can specify a more complex condition by pacing ANDxx or ORxx statements immediately following the DOUxx. If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation does not exist, the DO group is finished and control passes to the next calculation operation after the ENDDO statement. If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation exists, the operations in the DO group are repeated.

DO****	**This is a further Doxx explanation, not an operation**. DOUxx (=Do Until xx) tests at the end of the loop and therefore **always** gets executed at least once. DOWxx (=Do While xx) tests for the condition first, before executing the loop and thus it is possible that no instructions in the DOW loop get executed.
ELSE	**The ELSE operation is an optional part of the IFxx** operation. If the IFxx comparison is met, the instructions before ELSE are processed; otherwise, the instructions after ELSE are processed.
ENDSR	**End of Subroutine.** Use this as the ending statement in A subroutine that would begin with a BEGSR operation. There are no other operands (Factor 1, Factor 2, Result)
END	**Generic End of If or Do Operations**. This Generic operation will end a DO block or an If block of instructions. There are no other operands (Factor 1, Factor 2, Result)
ENDDO	**Ends DO, DOUXX or DOWXX Operations.** There Are no other operands (Factor 1, Factor 2, Result)
ENDIF	**Ends If Operations.** There are no other operands (Factor 1, Factor 2, Result)
EXSR	**Invoke Subroutine.** This operation branches to an in-line subroutine, causes the subroutine code to be executed, and when the subroutine ends, it passes control to the statement following the EXSR operation. The EXSR (execute subroutine) operation causes the RPG subroutine provided in Factor 2 to be performed. The subroutine name must be a unique symbolic name and must appear as Factor 1 of a BEGSR operation in another section of the same program. Whenever the EXSR appears in calculations, the subroutine that is named is immediately executed. Following the processing of all the subroutine, control is passed to the statement following the EXSR operation except when a GOTO within the subroutine sends the program to a label outside the subroutine or when the an error subroutine is being processed.
GOTO	**Unconditional branch statement.** This operation causes the next operational instruction to be the statement following a named (labeled) TAG Statement. The

	GOTO Label is provided in Factor 2.
IFxx	**If a condition xx is true, execute statements t**hat follow the IFxx statement and continue to the group-ending END or ENDIF statement. Other than the requisite RPG columnar formatting, the RPG standard IFxx operation is similar in function to the "IF" statement varieties in other languages. As such it allows a group of calculations to be processed if a certain relationship, specified by xx, exists between Factor 1 and Factor 2. The "ENDIF" statement is the last statement in an "IF Group." The If tests can be made more complex by adding the "ANDxx" and "ORxx." operations. The If statement itself can have conditioning indicators but it does not use resulting indicators. Factor 1 and Factor 2 are compared just like the RPG COMP operation. If the relationship specified by the IFxx and any associated ANDxx or ORxx operations exists, the statements are executed, if it does not exist, control passes to the calculation operation immediately following the associated ENDIF operation. If an "ELSE" is also specified as described above under ELSE, when the group is finished, control passes to the first calculation operation that can be processed following the ELSE operation.
ITER	**Iterate the instructions in a Do group** from the beginning. Run through Do group another time, bypassing the instructions between the ITER and the ENDDO. There are no operands (Factor 1, Factor2, or Result)
ORxx	**OR -- if another Condition is True, then...** the "or" operation works with a preceding operation that has established a condition, (HI, LO, EQ). This operation ties the prior operation with this operation so that if either are true, the statement is true .
OTHER	**The "Otherwise Select" or (OTHER) operation** Begins the sequence of operations to be processed if no WHxx condition is satisfied in a SELEC group. The Sequence ends with the ENDSL or END operation. See SELEC and WHxx operations. There are no operands (Factor 1, Factor2, or Result)

LEAVE	**Leave a Do Group.** A very similar statement to ITER, but very different. Whereas an ITER causes the loop to repeat, the Leave statement says that the Do loop is over… move on to an instruction after the DO. There are no operands (Factor 1, Factor2, or Result)
SELEC	**The SELEC operation begins a selection grouping.** The "select group" conditionally processes one of several alternative sequences of operations. It works with the WHxx group operation, and optionally with the OTHER group operation. A SELEC group ends with an ENDSL or END statement. The SELEC packaging consists of the SELEC statement, zero or more groups, an optional OTHER group, and the requisite ENDSL or generic END statement. It works like this: When the SELEC operation is processed, control passes to the statement following the first WHxx condition that is satisfied. All statements within the WHxx group are then executed until the next WHxx operation. Control then passes to the ENDSL statement. If no WHxx condition is satisfied and an OTHER action is specified, control passes to the statement following the OTHER operation. If no WHxx condition is satisfied and no OTHER operation is specified, control transfers to the statement following the ENDSL operation of the select group. There are no operands (Factor 1, Factor2, or Result)
TAG	**Destination label for a GOTO operation**. "Where a 'GOTO' goes.
TESTB	**Test Bit**– Individual Bit Testing. Each individual bit in a byte can be tested for on /off (0, 1) status. RPG has a number of bit operations that can be used to test and manipulate the individual bits in characters. The TESTB operation compares the bits identified in factor 2 with the corresponding bits in the field named as the result field. Only one character is tested in this operation so the result field must be a one-position character field. Resulting indicators in HI, LO, EQ, reflect the status of the result field bits. Factor 2 is always a source of bits for the result field. The bits to be tested are identified by the numbers 0 through 7. (0 is the leftmost bit.) The bit numbers must be enclosed in apostrophes, and the entry

	must begin in the first position of Factor 2.
TESTN	**Test Numeric:** The TESTN operation tests a character result field for the presence of zoned decimal digits and blanks. The result field must be a character field. To be considered numeric, each character in the field, except the low-order character, must contain a hexadecimal F zone and a digit (0 through 9). The low-order character is numeric if it contains a hexadecimal C, hexadecimal D, or hexadecimal F zone, and a digit (0 through 9). An indicator is turned on in the HI indicator area if either the result field contains numeric characters, or it contains a 1-character field that consists of a letter from A to R. An indicator is turned on in the LO area if the result field contains both numeric characters and at least one leading blank. For example, the values b123 or bb123 set this indicator on. An indicator is turned on in the EQ area when the result field contains all blanks
TESTZ	**Test Zone:** The TESTZ operation tests the zone of the leftmost character in the result field. Resulting indicators are set on according to the results of the test. The characters &, A through I, and any character with the same zone as the character A set on the indicator in the HI position. characters - (minus), J through R, and any character with the same zone as the character J set on the indicator in the LO position. . Characters with any other zone set on the indicator in the EQ position
WHxx	**"When True Then Select"** (WHxx) These are the operations of a select group which determine where control passes after the "SELEC (Begin a Select Group)" operation is processed. The WHxx conditional operation is true if factor 1 and factor 2 have the relationship specified by xx If the condition is true, the operations following the WHxx are processed until the next WHxx, OTHER, ENDSL, or END statement is encountered.

Call Operations (Inter-program)

One of the simplest ways to create applications is to build small programs / routines and link them together to form a cohesive complete application system. In Table 13-4, we introduced the notion of an inline subroutine which is a set of "stand-alone" instructions which get invoked independently throughout a program. Though the notion of building small modules does work somewhat with subroutines, being able to get outside your program and link to routines that are not buried within one big program provides an even more powerful and flexible way to build applications.

Without getting into a computer science discussion about call by name/value or call by reference, let's just say that the natural way to call a program in RPG/400 has always been to use the dynamic call. The beauty of this is that there is little thinking and as a new RPG programmer before you can even see the value of various calling mechanisms, you are using the AS/400's innate ability to dynamically call programs from one language to the same or another.

For argument purposes, let's just say that you need to write an application that needs some major calculations to produce your answers. Let's also say that there is an existing RPG/400 program written by your predecessor that provides all of these calculations. You merely have to pass it a few parameters and receive a few back and that's all there is. Let's also say that each set of answers that are provided need to be stored in the database and be indexed for later access. Let's also say that your predecessor has written a routine in CL that accepts parameters from a program and calls other programs and does all the work necessary to create a new database for each new program run. There is no reason for you to have to rewrite these programs. RPG/400 has a very nice and very easy to use dynamic program call facility that is at your disposal.

The operations to call, pass parameters, return, and deal with program resources are shown in Table 13-5. The list is not long and for RPG/400, that's all there is. With RPGIV, the language has been expanded to use what are called bound programs / procedures and/or prototyped procedures with various flavors of CALL operations. We'll pick that up in Chapter ****. For now, Table 13-5 shows you all you need to make inter program communication work for you in RPG/400.

Table 13-5 CALL Operations

Operation	Provided Function
CALL	**Calls a program** written in the same or a different language from the same system. The program name to call can be provided in a quoted literal name or in a variable. A parameter list can be provided via multiple PARM statements following the CALL operation or a PLIST operation provided in the result field
FREE	**Deactivate a Program.** The FREE operation removes a program from the list of activated programs that have been "called," frees static storage, and ensures that program initialization (first cycle processing) occurs the next time the program is called. This operation does not close files or unlock data areas. It is not supported in RPGIV.
PARM	**Identify Parameters for Program.** The "Identify Parameters" operation defines the parameters that compose a parameter list (PLIST). PARM operations can appear anywhere in calculations as long as they immediately follow a PLIST or a CALL operation to which they refer. PARM statements must be in the order expected by the called program. One PARM statement, or as many as 255 PARM statements in RPG/400, can follow a PLIST or CALL.
PLIST	**Identify a Parameter List for Program.** The "Identify Parameter List" (PLIST) operation defines a unique symbolic name for a list of parameters (PARMs) to be specified in a CALL operation. The PLIST operation must be immediately followed by at least one PARM operation. The name of the list is supplied in Factor 1. For programs that are passed parameters by a calling program a special factor 1 entry containing "*ENTRY" must be provided to catch the parameters passed by a calling program. The list is ended when an operation other than PARM is encountered.
RETRN	**Return to Caller (Calling Program).** The RETRN operation causes a return to the calling program. If a halt indicator (H1 to H9) is on, the program ends abnormally. In this case, all open files are closed and an error is sent to the caller. If no halt indicators are on, the last record

	LR indicator is checked. If it is on, the called program ends normally and closes files and data areas. If no halt condition exists and LR is not on, the program returns to the calling routine. Data is preserved for the next time the program is called. Files and data areas are not written.

Data Manipulation Operations

Business computer systems need lots of ways to manipulate and shape data for testing and for storage purposes. As a business-first language RPG has operations ranging from the bit level to the byte level to the field level that provide they types of data manipulation facilities that you would expect in a high-quality business programming language. Table 13-6 gives us a good look at these operations.

Table 13-6 Data Manipulation Operations

Operation	Provided Function
BITOF	**Set Bits Off – Individual Bit Manipulation.** Each individual bit in a byte can be set off to a binary "0" value. Similar to BITON.l
BITON	**Set Bits On -- Individual Bit Manipulation.** Each individual bit in a byte can be set on to a binary "1" value. The BITON operation causes bits identified in factor 2 to be set on (set to 1) in the result field. Bits not identified in factor 2 remain unchanged. Therefore, when using BITON to format a character, you should use both BITON and BITOF
CLEAR	**Clear a structure.** The CLEAR operation sets elements In a structure (record format, data structure, array, or table) or a variable (field, subfield, or indicator), to zero, blank or '0', depending on field type (numeric, char, or indicator). It is a convenient way to clear structures on a global basis, as well as element by element, during run time. Factor 1 must be blank unless factor 2 contains a DISK record format name; in which case, it can contain *NOKEY to indicate that all fields except key fields are to be cleared.

MOVE	Move data from one field to another right to left. See Table 13-2 Basic Operations
MOVEA	**Move data from an array to a field or vice verse, left to right.** See Table 13-2 Basic Operations
MOVEL	**Move data from one field to another, left to right.** See Table 13-2 basic Operations
MHHZO	**Move High to High Zone Nibble Operation.** The Move High to High Zone (MHHZO) operation moves a half-byte or "nibble" of data. It moves the zone portion of a character from the leftmost zone in the character field in Factor 2 to the leftmost zone in the Result field.
MHLZO	**Move High to Low Zone Nibble Operation.** The Move High to Low Zone (MHHZO) operation moves a half-byte or "nibble" of data. It moves the zone portion of a character from the leftmost zone in the character field in Factor 2 to the rightmost zone in the result field.
MLHZO	**Move Low to High Zone- Nibble Operation.** The Move Low to High Zone (MHHZO) operation moves a half-byte or "nibble" of data. It moves the zone portion of a character from the rightmost zone in the character field in Factor 2 to the leftmost zone in the result field.
MLLZO	**Move Low to Low Zone-- Nibble Operation.** The Move Low to Low Zone (MLLZO) operation moves a half-byte or "nibble" of data. It moves the zone portion of a character from the rightmost zone in the character field in Factor 2 to the rightmost zone in the result field.
RESET	**Reset a structure**. The RESET operation is similar to the CLEAR operation. It sets elements in a structure (record format, data structure, array, or table) or a variable (field, subfield, or indicator), to its initial value, It is a convenient way to reset structures on a global basis, as well as element by element, during run time. This initial value can be established using data structure initialization, or you can use the initialization subroutine to assign an initial value to the structure or variable. The RESET operation causes a snapshot of the variable or structure to be taken and this becomes the RESET value. Factor 1 must be blank unless factor 2 contains a DISK record format name; in which case, it can contain *NOKEY to indicate that all fields except key fields are

	to be reset.
SETON	**Turn Indicator ON.** This operation immediately sets the indicators specified in the HI LO EQ area to the "on" or "1" condition.
SETOF	**Turn Indicator OFF**. This operation immediately sets the indicators specified in the HI LO EQ area to the "off" or "0" condition

Database & Device File Operations

Just as a business computer system needs its compare and branch operations to provide business logic, a business programming language needs natural integrated database operations to maximize the productivity of programmers and minimize the need for high-priced database administrators. The RPG/400 and the RPGIV language compilers have been built from the ground up with full knowledge of the system's integrated database. As such programmers are not burdened with coding input or output or update specifications in RPG programs. The compiler fetches the descriptions right from the database at compile time. This makes RPG business programmers the most productive programmers in the world.

Add to this notion a series of powerful database record at a time operations unprecedented on any other computer system or any other language. For example the CHAIN operation provides random read or random read by key. The UPDAT operation is always ready to update the record just read by a CHAIN, or any type of READ operation. The WRITE operation adds records to a database file and of course the DELET operation can delete the record just read or it can be used to directly access a record via a search argument for deletion.

How many different ways do you want to read data? In addition to the CHAIN operation, for example the SETXX (EQ or GT) positions the database to a specific record from which a subsequent READ operation can be performed. So speaking of read operations, you can READ the next record consecutively, READP (READ prior) the prior record one back and one back after that etc. You can also READE (READ equal)

the next record whose key is equal to a search argument and you can specify. You can also REDPE (READ prior equal) if you want to read backwards by key. RPG programmers don't like RPG because it is simple. They like it because it is rich in function and easy to use as witnessed by the host of database operations shown in Table 13-7.

But, then again, that's just database. A number of these operations also work with other types of devices such as tape drives. Despite all of the accolades I can get from showing these types of operations against database and media, there is no set of operations more powerful than those dealing with the IBM-unique notion of a WORKSTN file. You just cannot beat it.

In its traditional Program Development Manager toolset, for example, and in its WebSphere Development Studio Client IDE (intelligent development environment) IBM offers a facility that enables workstation screens to be designed in a WYSIWYG style. Groups of these screens or panels as they are sometimes called are combined and compiled for use in a specific program. The result of the compilation is an AS/400 object called a display file. It is a *File type object but it is associated with a device that has the characteristics of an IBM 5250 terminal.

Just as RPG removes the requirement for programmers to code database input and output specifications, it provides the same facilities for WORKSTN device files. Each record format in a Workstation device file is in essence a display panel with both output and input fields available on the display.

When you write a program that is to use screen panels, you first send out prompts and then you read in the input that was entered. On many systems this is a very difficult task and requires highly skilled programming. IBM has made the notion of interactive much more easy than traditional methods by including compiler operation codes within the COBOL and RPG compilers.

So, if an RPG programmer wants to send out a screen panel, they would first specify the device file name as a WORKSTN file in the File Description specifications. Then, as you would expect the first operation to the screen would be a WRITE op-code. You would then want the program to wait a bit for input before continuing and as it turns out, if

you immediately follow-up with a READ operation, the program will wait for input before continuing. As simple as this is, it is actually more simple. The RPG compiler designers recognized that the typical sequence of operations is a WRITE followed by a READ, so they devised a new operation specific to the WORKSTN file that does both a WRITE and a READ. The operation is called EXFMT for Execute Format.

This operation as well as all of the wonderful RPG database and device file operations are included in Table 13-7 for your review and learning pleasure.

Table 13-7 Database & File Operations

Operation	Provided Function
ACQ	**Acquire a device for program use**
CHAIN	Random retrieval from a file. The chain is a read operation by either relative record number or via a search argument against a key value in an indexed file database.. See Table 13-2 Basic Operations.
CLOSE	**Close files.** This operation closes the file specified in Factor 2
COMIT	**Commit records to DB and post to journal.** For applications using commitment control with database journaling, this operation commits the last set of writes and updates to the database.
DELET	**Delete DB File Record.** See Table 13-2 Basic Operations
DSPLY	**Displays variable values during program execution.** Unlike Debug, this command sends its results immediately to the job's message queue from where it can be viewed. See table 13-2 Basic Operations
EXCPT	**Calculation Time Output – outside RPG cycle.** This commend immediately outputs disk records or print lines from within calculations rather than through the fixed logic cycle. See Table 13-2 Basic Operations
EXFMT	**Write/then read a screen format** for interactive Programming. This is the major op code for sending a Screen panel to a display and then waiting for input from The user to the program. When the user inputs data it is

	returned to the program in this one operation. It is a combination of a Write screen operation followed by a wait followed by a read screen. See table 13-2 Basic Operations.
FEOD	**Force End of Data on File.** The FEOD operation Signals the logical end of data for a primary, secondary, or full procedural database file. By placing a file at end of data it effectively means that the next READ operation for the file will return an "end of file" condition; no record will be read, an error indicator will be returned, and no data will be received by the program. A CHAIN or SETLL operation against a file will undo the FEOD and reposition the file cursor on some record someplace before end of data.
FORCE	**Force Certain DB File to Read Next RPG Cycle.** This operation permits the program to change the Normal sequencing of a primary or secondary file by picking the file to be read (input) and processed on the next RPG cycle.
NEXT	**Causes Specific Device to Be Read Next**. Similar in nature to FORCE, this operation selects the specific device in a multi-device file that will be selected next cycle for input.
OPEN	**Open file NOW for processing.** To simplify the language, RPG files are by default opened automatically when the program is invoked. However, there may be times that it would best serve a program to open the files only if and when they are needed. Files that are opened by the explicit RPG OPEN operation, named in factor 2, must be designated as user controlled by specifying "UC" (user control) in positions 71 and 72)of the RPG/400 file description specifications. The OPEN operation requires that Factor 2 contain the name of the file to be opened.
POST	**Post – Loads File Info Data Structure.** This infrequently used operation posts information in an INFDS (file information data structure). The information is either on the status of a specific program device or I/O feedback associated with a file. In Factor 1, you can specify a program device name to get information about

	that specific program device. If a program has no POST operation code, or if it has only POST operation codes with factor 1 specified, the INFDS is updated with each input/output operation or block of operations. If you leave factor 1 blank, you get I/O feedback information. Specify a file name in Factor 2. The information for this file is posted in the INFDS associated with this file.
READ	**Read a Record from a file** at the point of the file cursor. Also used to read a display format form a WORKSTN file. See Table 13-2 Basic Operations for more detail.
READC	**Read Next Changed Record – Subfile.** The READC operation can be used only with an externally described WORKSTN file to obtain the next changed record in a subfile. When a subfile is displayed to a user, the RPG program "disconnects." The user may then roll through the file changing any of many records. The Read changed (READC) operation looks for only the records that were changed after the disconnect and before the reconnect. to the RPG program. Factor 2 is required and must be the name of a record format defined as a subfile by the SFILE keyword on the file description specifications.
READE	**Read Equal by key.** The READE operation retrieves the next sequential record from a full procedural file. (F in position 16 of the RPG/400 file description specs) if the key of the record matches the search argument. If the key of the record does not match the search argument, the indicator that must be specified in RPG/400 positions 58 and 59 (EQ) is set on, and the record is not returned to the program.
READP	**Read Prior Record (Consecutive)** The READP operation reads the prior record from a full procedural file (F in position 16 of the F spec). This operation goes against the data and does not use an index. Place the name of a file or record format to be read in Factor 2. A record format name in factor 2 is allowed only with an externally described file. If a record format name is specified in factor 2, the record retrieved is the first prior record of the specified type (in the event of a multi

	format logical file). Intervening records are bypassed.
REDPE	**Read Prior Equal (Key)** The REDPE operation retrieves the next prior sequential record from a full procedural file if the key of the record matches the search argument. In essence it reads backwards by key and only picks those records equal to the search argument specified in Factor 1. If the key of the record does not match the search argument, the indicator in RPG/400 positions 58-59 (EQ) is set on and the record is not returned to the program. Factor 1, is optional and it identifies the record to be retrieved. If factor 1 is left blank and the full key of the next prior record is equal to that of the current record, the next prior record in the file is retrieved. The full key is defined by the record format or file specified for Factor 2.
REL	**Release the program device** (WORKSTN) acquired Via the ACQ operation.
ROLBK	**Roll Back uncommitted DB records.** When program operations write or update records to a database while under commitment control, the before and after images are also written to a journal. If after performing these operations a program wants to undo the transactions and updates, the ROLBK operation does just that. It takes the database back to where it was before the transaction began (transaction boundary).
SETGT	**Set Greater Than Key or Relative Record #** The SETGT operation positions a database file at the next record with a key or relative record number (RRN) that is greater than the key or relative record number specified in factor 1. The file must be a full procedural file (F in position 16 of FD). The operation requires that Factor 1 be specified as the key or RRN search argument. Factor 2 is also required and can contain either a file name or a record format (externally described file only).
SETLL	**Set Lower Limit – key or relative record #.** The SETLL operation positions a file at the next record that has a key or relative record number RRN that is greater than or equal to the search argument (key or RRN) as specified in factor 1. The file must be full procedural (F

	in position 16 of FD) Factor 1 is required. Factor 2 is also required and can contain either a file name or a record format (externally described file only).
UNLCK	**Unlock a Data Area or release a locked record.** Use the UNLCK operation to unlock data areas and release database record locks in a program. Specify the name of the data area or the name of an update disk file or the word, *NAMVAR. The data area must already be specified in the result field of an *NAMVAR DEFN statement. When *NAMVAR is specified in factor 2, all data areas in the program that are locked are unlocked. Using the UNLCK operation releases the most recently locked record for an update disk file.
UPDAT	**Modify an existing database record.** Follows a read Type operation. Data in the record's fields are updated In program memory and then the new contents of the record are written over the former contents of the record in the database. See Figure 13-2 Basic Operations.
WRITE	**Create (Add) new records to file.** Unlike a file update which writes an existing record back to the database, the WRITE operation creates or adds a brand new record to the database. See Figure 13-2 Basic Operations

Data Structure, Data Area, Table, Array, String Operations

In Table 13-6, we took a good look at the basic operations that are available in RPG to manipulate data once it arrives in memory. As you may from this section, recall the data manipulation operations that we studied are based on field, character, or bit data elements. Though the list in Table 13-6 is comprehensive for basic structures, it does not include all of the manipulation operations. We saved some special operations on some special structures for Table 13-8 below.

The operations in this table are provided to work with data structures. A data structure is a superset of the other types of data that we studied in Table 13-6. Other than the disk and device operations described in Table

13-7, a "field" is the largest data element that we have covered to this point. A data structure, as a superset of a field, can contain multiple fields or data elements. As the title of this section suggests, the four different types of data structures that we are about to discuss are as follows:

1. Data Structure
2. Table
3. Array
4. String

The various operations that can be used against these structures are described in detail in Chapter ****. You may have noticed that we did not list the "Data Area" as one of the structures under study. That is because it is not a data structure. However, as you will learn in Chapter ****, a Data Area is an AS/400 object, similar in nature to a one field, one record, database file. For the data that is contained in a Data Area to make sense when it arrives in a program, IBM has provided the ability for a Data Area to use a Data Structure for its data definition.

All of us have learned that we should not use the word we are defining as the definition of the word itself. Yet, here we are, about to explore the operations provided in Table 13-8 for all sorts of data structures and yet, against the rules, one of them is actually named Data Structure.

Data Structure (as a data structure)

A Data Structure is a particular structure of data that combines a number of different fields of varying length and type into a meaningful structure that can be referenced and manipulated by one name, rather than having to reference each item in the structure individually. In essence, a Data Structure is the memory equivalent of a record format in a database file, though it does not have to be associated with a database file.

So, the Data Structure is a memory artifact into which data can be applied and manipulated. Data Structures can be one record or multiple record in nature. When there are multiple records in a data structure, it is known as a Multiple Occurrence Data Staructure.

Table

A Table is a data structure (grouping of data) of similar sized and shaped elements in tabular form. Tables can either be one column or two columns wide. A single column table, for example might consist of the fifty states as in the United States, properly spelled. A programmer could check this table to assure that a valid state abbreviation was used in an application. A two-column table is officially called an Alternating Table. Carrying the states example into two columns, by adding a second column or alternating table, the name of the state could be provided in the alternate table. Then, in addition to verifying the validity of the state code in the look up, the operation could also return the name of the state in the same operation. Thus, PA would beget Pennsylvania and NJ would beget New Jersey, etc. As you will learn in Chapter ****, tables in RPG always begin with the three letters "TAB."

An Array is a data structure that consists of all similarly sized and typed data elements. Each element in an array for example is a field in its own right but once defined it is accessible only through array operations. A common application example for arrays over the years is to store monthly sales figures. By definition, all of the monthly totals would be stored in fields of the same size and type – often in the neighborhood of nine numeric digits, two of which would represent decimal positions. Programmers learned quickly that they could define an array of 108 characters to hold the twelve sales values for the year.

Arrays also permit indexing. Thus, January can be accessed in an array called SAL, for example as SAL,1. That is read SAL sub 1. In RPGIV, the index is in parentheses as in SAL(1). The index can also be a variable so that all elements of the array can be processed individually in a loop by adding 1 to the index value. Arrays can also be processed using the lookup that is often used with tables. Once the lookup operation completes, the specific array element can be processed, just as with a table.

Strings

A String in all computer languages is a group of alphameric characters. In the AS/400 RPG world, strings are held in fields. Thus, the String operations to which you are about to be introduced in Table 13-8 operate within the AS/400 type called "field," which, for the String operations, works just fine.

Though RPG's basic string operations provide many of the same facilities as provided by other languages, please do not read that to mean the structure of the string in RPG (a field) is the same as the structure of the string in languages which treat a string as something special to the language. For example, an RPG string is not compatible with a Java String object. Thus when RPG and other languages exchange strings, it is not a straight-forward process.

Table 13-8 Data Area, Table, Array, String Operations

Operation	Provided Function
CAT	**Concatenate two character strings.** The concatenate (CAT) operation combines the character string specified in Factor 2 to the end of the character string specified in Factor 1 and places it in the result field. If no Factor 1 is specified, factor 2 is concatenated to the end of the result field string.
CHECK	**Check for Certain Chars in a String (L to R)** The CHECK operation verifies that each character in the base string (Factor 2) is among the characters indicated in the comparator string (Factor 1). All of the string operations are complex instructions with many specifications. See Chapter *****. Checking begins at the leftmost character of factor 2 and continues character by character, from left to right. Each character of the base string is compared with the characters of factor 1. If a match for a character in factor 2 exists in factor 1, the next base string character is verified. If a match is not found, an integer value is placed in the result field to indicate the position of the first incorrect character. The whole base string does not have to be checked, however. You can specify a start position in factor 2, separating it from the base string by a colon. This start position is optional and defaults to 1. If it is > 1, the value in the result field is still relative to the leftmost position in the base string, regardless of the start position. If no incorrect characters are found, the result field is set to zero.
CHEKR	**Check Reverse for Chars in a String (R to L).** This command works very similarly to the CHECK

	command. However, checking is done from right to left. Although checking is done from the right, the position placed in the result field will be relative to the left. See Chapter *****
CLEAR	**Sets values in structure to zero or blank**. See details in Table 13-6 Data Manipulation Operations
IN	**Retrieve a Data Area Object into Program.** The IN operation retrieves a data area and optionally allows you to specify whether the data area is to be locked from update by another program. For a data area to be retrieved by the IN operation, it must be specified in the result field of an *NAMVAR DEFN statement. See "DEFN" for information on *NAMVAR DEFN. in Table 13-9, Pgm Control, Declarative & Informational Operations. Factor 1 can contain the reserved word *LOCK or can be blank. The lock is held until (1) an UNLCK operation or (2) an OUT operation with no Factor 1 or (3) when the RPG program ends. Factor 2 must be either the name of the result field used when you retrieved the data area or the reserved word *NAMVAR. When *NAMVAR is specified, all data areas defined in the program are retrieved.
LOKUP	**Look up argument in a memory table or array.** The LOKUP operation causes a search to be made for a particular element in an array or table. Factor 1 is the search argument (data for which you want to find a match in the array or table named). For a table LOKUP, the result field can contain the name of a second table from which an element (corresponding positionally with that of the first table) can be retrieved. The name of the second table can be used to reference the element retrieved. The result field must be blank if Factor 2 contains an array name. Resulting indicators can be assigned to EQ and HI or to EQ and LO. The program searches for an entry that satisfies either condition with equal given precedence; that is, if no equal entry is found, the nearest lower or nearest higher entry is selected.
MOVEA	**Move Array to Field and Vice Versa**. See Table 13-6 Data Manipulation Operations for details
OCUR	**Specifies occurrence of the DS to be used.** When

	there is more than one record in a data structure it is called a multiple **OCUR**rance data structure. Similar to an array index, the records in the DS can be retrieved by their relative position in the structure. For a DS, this is called an occurrence. For example, record 3 would be occurrence3. The OCUR operation code specifies which Occurrence (record) of the data structure that is to be used next within the program. After an OCCUR op is specified, the occurrence of the data structure that was established by the OCUR operation is used. Factor 1 is optional. If specified, it sets the occurrence of the DS. If blank, the value of the current occurrence of the data structure in factor 2 is placed in the result field during the OCUR operation. The result field is optional. The value of the current occurrence of the data structure specified in factor 2 is placed in the result field.
OUT	**Write out a data area object from a program.** The OUT operation updates the data area specified in Factor 2. The rules for this operation include the following: (1) The data area must also be specified in the result field of a *NAMVAR Statement. (See "DEFN" for information on *NAMVAR DEFN. in Table 13-9, Program Control, Declarative & Informational Operations.) and (2) The data area must have been locked previously by a *LOCK IN statement or it must have been specified as a data area data structure by a U in position 18 of the RPG/400 input specifications. The RPG language implicitly retrieves and locks data area data structures at program initialization. When factor 1 contains *LOCK, the data area remains locked after it is updated. When factor 1 is blank, the data area is unlocked after it is updated. Factor 2 must be either the name of the result field used when you retrieved the data area or the reserved word *NAMVAR. When *NAMVAR is specified, all data areas defined in the program are updated.
RESET	**Resets structure values**. See Table 13-6 Data Manipulation Operations for details
SCAN	**Scan Character String for a Certain String.** The SCAN operation scans a character string (base string) contained in Factor 2 for a substring (compare string)

contained in Factor 1. The scan begins at a specified location contained in Factor 2 and continues for the length of the compare string which is specified in Factor 1. Factor 1 must contain either the compare string or the compare string, followed by a colon, followed by the length. If no length is specified, it is that of the compare string. Factor 2 must contain either the base string or the base string, followed by a colon, followed by the start location of the SCAN. If no start location is specified, a value of 1 is used. The result field contains the numeric value of the leftmost position of the compare string in the base string, if found. The result field is set to 0 if the string is not found. If the result field is specified and the start position is greater than 1, the result field contains the position of the compare string relative to the beginning of the source string, not relative to the start position. If no result field is specified, a resulting indicator in RPG/400 positions 58 and 59 (EQ) must be specified. See Chapter XXX for more detail and string examples.

SUBST **Substring – Results in a Portion of a String** The SUBST operation returns a substring from factor 2, starting at the location specified in factor 2 for the length specified in Factor 1, and places this substring in the result field. If factor 1 is not specified, the length of the string from the start position is used. Factor 1 can contain the length value of the string to be extracted from the string specified in factor 2. Factor 2 must contain either the base character string, or the base character string followed by a colon, followed by the start location. If the start position is not specified, SUBST starts in position 1 of the base string. The result field must be character If the substring is longer than the field specified in the result field, the substring will be truncated from the right. See Chapter XXX for more detail and string examples.

SORTA **Sort an array in ascending or descending sequence.** Factor 2 contains the name of an array to be sorted. The array is sorted into sequence (ascending or descending), depending on the sequence specified for the array in

	position 45 of the RPG/400 extension specifications. If no sequence is specified, the array is sorted into ascending sequence.
XLATE	**Translate String (eg Upper Case to Lower Case).** Characters in the source string (Factor 2) are translated according to the from and to strings (both in Factor 1) and put into the result field. Source characters with a match in the from string are translated to corresponding characters in the to string. XLATE starts translating the source at the location specified in Factor 2 and continues character by character, from left to right. If a character of the source string exists in the from string, the corresponding character in the to string is placed in the result field. Any characters in the source field before the starting position are placed unchanged in the result field. Factor 1 must contain the from string, followed by a colon, followed by the to string. Factor 2 must contain either the source string or the source string followed by a colon and the start location. If no start location is specified, a value of 1 is used. The result field must be specified – typically as a character field. See Chapter XXX for more detail and string examples.
XFOOT	**Sum the elements of an array to a result field**. See Table 13-6 Data Manipulation Operations for details
UNLCK	**Unlock a data area or release a record.** See Table 13-7 - Database & File Operations for details

.

Program Control, Declarative, Informational & Other Operations

Table 13-9 provides a description for a number of operations that can make coding easier and / or provide additional information without a lot of work during the programming cycle. Starting with the Field Definition operation, which permits like fields and special constructs to be defined to the KFLD and KLIST operations to make using composite keys easier, to the PARM and PLIST operations which ehlp line up data for program

passage, to the DEBUG, SHTDN (shutdown) and TIME operations, this table offers a lot of variety and utility and the operations contained herein will come handy as you become an accomplished RPG programmer.

Table 13-9 Program Control, Declarative, Informational & Other Operations

Operation	Provided Function
DEBUG	**Show Internal Variables and Indicators.** See Table 13-2 RPG Basic Operations
DEFN	**Field Definition.** Depending on the Factor 1 entry, the declarative DEFN operation can do either of the following: (1) ***LIKE** -- Define a field based on the attributes (length and decimal positions) of another field. (2) ***NAMVAR** Define a field as a data area.You can use the DEFN operation anywhere within calculations The ***LIKE DEFN** operation defines a new field based up-on the attributes (length, decimal positions) of another field. Factor 2 must contain the name of the field being referenced, and the result field must contain the name of the field being defined. The ***NAMVAR DEFN** oper-ation associates an RPG program defined field, a data structure, a data-structure subfield, or a data-area data structure with an AS/400 data area (outside your RPG program). In factor 2, specify the external name of a data area. Use *LDA for the name of the local data area or use *PDA for the Program Initialization Parameters (PIP) data area. If you leave Factor 2 blank, the result field entry is both the RPG name and the external name of the data area. In the result field, specify the name of one of the following that you have defined in your program.
DUMP	**Program dump to help debug programs.** The UMP operation provides a dump (all fields, all files, indicators, data structures, arrays, and tables defined) of the program.
KFLD	**Define parts of a key search argument – by field name.** The KFLD operation is a declarative operation that indicates that a field is part of a search argument identified by a KLIST name. The result field must contain the name of a field that is to be part of the

	search argument. The result field cannot contain an array name or a table name. Each KFLD field must agree in length, data type (character or numeric), and decimal position with the corresponding field in the composite key of the record or file.
KLIST	**Define a composite key of KFLDs.** The KLIST operation is a declarative operation that gives a name to a list of KFLDs used as a search argument to retrieve records from files that have a composite key. Factor 1 must contain a unique name.
PARM	**Identify parameters for a program.** See Table 13-5 CALL Operations
PLIST	**Identify a parameter list for program.** See Table 13-5 CALL Operations.
TAG	**Tag – Destination for a GOTO.** Label "Where a 'GOTO' goes.
SHTDN	**Test for system shut down request.** The SHTDN operation provides a means for the programmer to test whether the system operator has requested a system shutdown. If the system operator has requested shutdown, the resulting indicator specified in RPG/400 positions 54 and 55 (HI) is set on.
TIME	**Get time of day and store in a field.** See Table 13-2 Basic Operations

Structured Operations

Some time after RPGIII was introduced, IBM substantially enhanced the RPG language with the addition of a number of structured operations. These are fully covered in Chapter ****. As you will see as you examine the structured operations in Table 13-10, each of these operations has been explained fully in another table. For your convenience in being able to recognize this powerful RPG structured operators, we repeat them below together and provide specific referene to the table in which they are more fully explained.

Table 13-10 Program Control, Declarative, & Informational Operations

Operation	Provided Function
ANDxx	**And If another condition is true, then**.. See Table 13-4 Compare & Branch Operations
CASxx	**Conditionally Invoke Subroutine on compare**. See Table 13-4 Compare & Branch Operations
DO	**Straight DO Loop.** See Table 13-4 Compare & Branch Operations
DOUxx	**Do until xx is true.** See Table 13-4 Compare & Branch Operations
DOWxx	**Do while xx is true.** See Table 13-4 Compare & Branch Operations
ELSE	**Else.** See Table 13-4 Compare & Branch Operations
ENDSL	**End of Select Group.** See Table 13-4 Compare & Branch Operations
ENDSR	**End of Subroutine.** See Table 13-4 Compare & Branch Operations.
ENDyy	**End a Group yy = IF or DO etc.** See Table 13-4 Compare & Branch Operations
IFxx	**If a condition is xx, perform a set of operations.** See Table 13-4 Compare & Branch Operations
ITER	**Iterate– run through do group another time.** See Table 13-4 Compare & Branch Operations
LEAVE	**Leave a do group after ENDxx.** See Table 13-4 Compare & Branch Operations
ORxx	**Or If another condition is true, then...** See Table 13-4 Compare & Branch Operations
OTHER	**Otherwise select operation in select group.** See Table 13-4 Compare & Branch Operations
SELEC	**Begin a select group**. See Table 13-4 Compare & Branch Operations
WHxx	**When true then select.** See Table 13-4 Compare & Branch Operations

All RPG Operations / Parameters

Now that we have examined all of the RPG/400 operations in this chapter, we have just one more task to accomplish. In the next short section, we provide a comprehensive table of all the RPG/400 operations that shows the Factor 1, Factor 2, and Result Field components and the potential status of resulting indicators after the operation.

To help in our examination of all these op-codes, there are a number of abbreviations and symbols that we must briefly define in Table 13-11. You may find these anywhere in Table 13-12 but primarily, you will see them in the resulting indicators area showing which columns turn on indicators for what purpose.

Table 13-12 can serve as a very handy summary guide to RPG operations. It shows how most of the RPG/400 operations that we described in terms of capabilities earlier in this chapter are used in actual calculation specifications. Additionally, the resulting indicators are explained according to the following key:

To make it much easier to look up operations in Table 13-12, the operations are presented in alphabetic sequence by op-code. If you need more information than the format and options of the operations in this table, check out the first column. It is a direct link to the first Table in which the operation is explained in more detail. For example, B is basic operations, C is Compare, D is database/device, M is manipulation, P is program call, S is data structure, and O is other operations.

Table 13-11 Op Code Symbols / Indicators

+	If the result is positive, the indicator placed in these columns is set on.
-	If the result is negative, the indicator placed in these columns is set on.
0	If the result is zero, the indicator placed in these columns is set on.
BL	Blank(s)
BN	Blank(s) then numeric
BOF	Beginning of file
EOF	End of database file has been reached via READ type operations
ER	Error – indicator specified in these columns is set on indicating that an error occurred in the operation
EQ	Equal condition also If factor 1 is equal to factor 2, set on indicator specified in these columns
FD	Found
HI	If factor 1 is greater than factor 2, set on indicator specified in these columns
IN	Indicator
LO	If factor 1 is less than factor 2, set on indicator specified in these columns
LR	Last record indicator
NA	Not applicable
NR	No record found – indicator is set on signifying no record found
NU	Numeric
Of	Indicator to be set off
On	Indicator to be set on
Z	If the result is zero, the indicator placed in these columns is set on.
ZB	Zero or blank

Table 13-12 All RPG Operations / Parameters Alphabetical

T	Code	Factor1	Factor2	Result	I1	I2	I3
D	Acq	Device name	Workstn file		NA	ER	NA
B	ADD	Addend	Addend	Sum	+	-	0
C	ANDxx	Comparand	Comparand				
C	BEGSR	Subr. Name					
M	BITOF		Bit #s in byte	Char field			
M	BITON		Bit #s in byte	Char field			

C	CABxx	Comparand	Comparand	Go to instr. label	HI	LO	EQ
P	CALL	&OS	'Program name' or variable name	Plist name optional	NA	ER	LR
M	CAT	Source str 1	Source string 2: #blanks	Target string			
C	CASxx	Comparand	Comparand	Subroutine label	HI	LO	EQ
B	CHAIN	Search arg.	File/record name		NR	ER	NA
M	CHECK	Comparator str	Base string: start pos	Left-pos	NA	ER	FD
S	CHEKR	Comparator str	Base string: start pos	Right-pos	NA	ER	FD
M	CLEAR	*nokey	Structure, variable, record				
D	CLOSE		File name		NA	ER	NA
D	COMIT	Boundary			NA	ER	NA
C	COMP	Comparand	Comparand		HI	LO	EQ
B	DEBUG	Identifier	Output file name	Debug Info			
M	DEFN	*like	Reference field	Defined field			
M	DEFN	*namvar or *extrn	External / internal data area	Internal program area			
B	DELET	Search arg.	File / record name		NR	ER	NA
B	DIV	Dividend	Divisor	Quotient	+	-	0
C	DO	Start value	Limit value	Index val			
C	DOUxx	Comparand	Comparand				
C	DOWxx	Comparand	Comparand				
O	DUMP	Identifier					
D	DSPLY	Msgid or literal	Outq or variable to view	Response	NA	ER	NA
C	ELSE						
C	END			Increment val			
C	ENDCS						
C	ENDDO			Increment value			
C	ENDIF						
C	ENDSL						
C	ENDSR	Label	Return point				
D	EXCPT		EXCPT name				

B	**EXFMT**		**Screen name**		**NA**	**ER**	**NA**
C	**EXSR**		**Subroutine name**				
D	**FEOD**		**File name**		**NA**	**ER**	**NA**
D	**FORCE**		**File name**				
P	**FREE**		**Program name**		**NA**	**ER**	**NA**
C	**GOTO**		**Program Label**				
C	**IFxx**	**Comparand**	**Comparand**				
S	**IN**	***Lock**	**Data Area name**		**NA**	**ER**	**NA**
C	**ITER**						
O	**KFLD**			**Key field name**			
O	**KLIST**	**Klist name**					
C	**LEAVE**						
S	**LOKUP**	**(Array) Search arg.**	**Array name**		**HI**	**LO**	**EQ**
S	**LOKUP**	**(Table) Search arg.**	**Table Name**	**Alternate Table name**	**HI**	**LO**	**EQ**
M	**MHHZO**		**Source field**	**Target fld**			
M	**MHLZO**		**Source field**	**Target fld**			
M	**MLHZO**		**Source field**	**Target fld**			
M	**MLLZO**		**Source field**	**Target fld**			
B	**MOVE**		**Source field**	**Target fld**	**+**	**-**	**ZB**
B	**MOVEL**		**Source field**	**Target fld**	**+**	**-**	**ZB**
B	**MOVEA**		**Source array or field**	**Target array/field**	**+**	**-**	**ZB**
B	**MULT**	**Multiplicand**	**Multiplier**	**Product**			
B	**MVR**			**Remainder**	**+**	**-**	**Z**
D	**NEXT**	**Program device**	**File name**		**NA**	**ER**	**NA**
S	**OCUR**	**Occurrence value**	**Data structure name**	**Occurrence value**	**NA**	**ER**	**NA**
D	**OPEN**		**File name**		**NA**	**ER**	**NA**
C	**ORxx**	**Comparand**	**Comparand**				
C	**OTHER**						
S	**OUT**	***lock**	**Data Area name**		**NA**	**ER**	**NA**
P	**PARM**	**Target field**	**Source Field**	**Parameter**			
P	**PLIST**	**Plist name**					
D	**POST**	**Program**	**File Name**	**INFDS**	**NA**	**ER**	**NA**

		device		name			
B	READ		File/record name	Data structure	NA	ER	EOF
B	READC		Record Name / SFL		NA	ER	EOF
D	READE	Search arg.	File/record name	Data structure	NA	ER	EOF
D	READP		File/record name	Data structure	NA	ER	BOF
D	REDPE	Search arg.	File/record name	Data structure	NA	ER	BOF
D	REL	Program device	File Name		NA	ER	NA
M	RESET	*NKEY	Structure or variable				
P	RETRN						
D	ROLBK				NA	ER	NA
S	SCAN	Comparator string : lgth	Base string : start	Left-most position	NA	ER	FD
C	SELEC						
D	SETLL	Search arg.	File/record name		NR	ER	EQ
D	SETGT	Search arg.	File/record name		NR	ER	NA
M	SETON				OF	OF	OF
M	SETOF				ON	ON	ON
O	SHTDN				ON	NA	NA
S	SORTA		Array name				
B	SQRT		Field name/ value	Square root			
B	SUB	Minuend	Subtrahend	Difference	+	-	Z
S	SUBST	Length to extract	Base string: start	Target string	NA	ER	NA
C	TAG						
C	TESTB	Label	Bit #s	Char field	OF	ON	EQ
C	TESTN			Char field	NU	BN	BL
C	TESTZ			Char field			
B	TIME			Num field			
D	UNLCK		Data Area or file name		NA	ER	NA
B	UPDAT		File / record format name	Data structure	NA	ER	NA
C	WHxx	Comparand	Comparand				
B	WRITE		File / record format name	Data structure	NA	ER	NA
S	XFOOT		Array name	Sum	+	-	Z
S	XLATE	Fron : To	String : Start	Target	NA	ER	NA

			string			
B	**Z-ADD**	**Addend**	**Sum**	**+**	**-**	**Z**
B	**Z-SUB**	**Subtrahend**	**Difference**	**+**	**-**	**Z**

Chapter 14

RPGIV Operations and Built-In Functions

After studying all the RPG/400 operations in Chapter 7, the best place to start now for RPGIV is to examine what is the same. Let's look at what you already know and how much of your new knowledge is portable to RPGIV.

Of the 101 RPG/400 op codes that we studied in Chapter 7, only one, the FREE op-code has been eliminated. Surely IBM has its reasons but you can think of it as IBM's attempt to bring the number of operations to an even hundred. IF Big Blue had not added another 22 instructions (Table 14-2) and a bunch of built-in functions (Table 14-9) to RPGIV, then maybe that supposition would stand. To make it easy for you to know how much you already know about RPGIV, we have included the 100 operations and the name of its equivalent in RPGIV. It should be a pretty quick exercise because only fifteen operations have changed and the change in all cases was for readability. So, without more ado, take a look at Table 14-1 below so you can learn about how much you already know.

Table 14-1 RPG/400 and RPGIV OP-Code Differences

RPG/400 OP-Code	RPGIV OP-Code	RPG/400 OP-Code	RPGIV OP-Code
ACQ	ACQ	CLEAR	CLEAR
ADD	ADD	CLOSE	CLOSE
ANDxx	ANDxx	COMIT	COMMIT *
BEGSR	BEGSR	COMP	COMP
BITOF	BITOFF *	DEBUG	DEBUG
BITON	BITON	DEFN	DEFINE *
CABxx	CABxx	DELET	DELETE *
CALL	CALL	DIV	DIV
CAT	CAT	DO	DO
CASxx	CASxx	DOUxx	DOUxx
CHAIN	CHAIN	DOWxx	DOWxx
CHECK	CHECK	DUMP	DUMP
CHEKR	CHECKR *	DSPLY	DSPLY

RPG/400 OP-Code	RPGIV OP-Code	RPG/400 OP-Code	RPGIV OP-Code
ELSE	ELSE	PARM	PARM
END	END	PLIST	PLIST
ENDCS	ENDCS	POST	POST
ENDDO	ENDDO	READ	READ
ENDIF	ENDIF	READC	READC
ENDSL	ENDSL	READE	READE
ENDSR	ENDSR	READP	READP
EXCPT	EXCEPT *	REDPE	READPE *
EXFMT	EXFMT	REL	REL
EXSR	EXSR	RESET	RESET
FEOD	FEOD	RETRN	RETURN *
FORCE	FORCE	ROLBK	ROLBK
FREE	NA	SCAN	SCAN
GOTO	GOTO	SELEC	SELECT *
IFxx	IFxx	SETLL	SETLL
IN	IN	SETGT	SETGT
ITER	ITER	SETON	SETON
KFLD	KFLD	SETOF	SETOFF *
KLIST	KLIST	SHTDN	SHTDN
LEAVE	LEAVE	SORTA	SORTA
LOKUP	LOKUP	SQRT	SQRT
LOKUP	LOOKUP *	SUB	SUB
MHHZO	MHHZO	SUBST	SUBST
MHLZO	MHLZO	TAG	TAG
MLHZO	MLHZO	TESTB	TESTB
MLLZO	MLLZO	TESTN	TESTN
MOVE	MOVE	TESTZ	TESTZ
MOVEL	MOVEL	TIME	TIME
MOVEA	MOVEA	UNLCK	UNLOCK *
MULT	MULT	UPDAT	UPDATE *
MVR	MVR	WHxx	WHENxx *
NEXT	NEXT	WRITE	WRITE
OCUR	OCCUR	XFOOT	XFOOT
OPEN	OPEN	XLATE	XLATE
ORxx	ORxx	Z-ADD	Z-ADD
OTHER	OTHER	Z-SUB	Z-SUB
OUT	OUT		

RPGIV-Only Operations

In 1994, the major changes in no particular order that everybody was talking about for the "new" RPGIV were the following:

1. Columns expanded to support longer field names.
2. Keywords for column functions
3. EVAL statement with extended Factor 2
4. Elimination of the E specification
5 D specification for defining fields, arrays, and structures
6. New date operations including date arithmetic.

Some might suggest that the major change immediately in 1994 was the introduction of the EVAL operation that provided RPG with the ability to enable equations and expressions following this operation code. Eventually RPGIV became a fully functional ILE language and IBM gave many more facilities to the language. Not all of these were provided with new operation codes but a number were.

Procedures and prototyped procedures were quick to arrive giving the language some of the flavor of the traditional block structured languages. As the supply of RPG-trained programmers dwindled, this language change helped C programmers from other platforms more readily understand RPG and its benefits. Pointers and pointer operations were also added to the language extending it into areas that had been reserved for low-level functions written in other languages.

To make the language even more likeable to those who had become accustomed to the FOR Loops in BASIC, the FOR Loop was also made available. In the last few years additional expression logic was added to the EVAL statement along with a fully free form of RPGIV. In essence, IBM enhanced the EVAL statement and then eventually removed its requirement for free- form operations

Error monitoring facilities were also added as operations to make the language more similar in its ability to trap various errors during execution. Additionally, IBM began work on providing full XML facilities into the language with the first installment being included in the operations shown in Table 14-2.

Table 14-2 RPGIV-Only Operations

Operation	Provided Function
ADDDUR	**Add a Duration to a Date** (Days, Months, Years) The ADDDUR is an original RPGIV operation that works with dates. It adds the duration specified in Factor 2 to a date or time or timestamp field or constant specified in the Result field and places the resulting Date, Time or Timestamp in the result field. Factor 1 is optional if The programmer prefers the longer way of coding the operation. If factor 1 is not specified the duration is added to the field specified in the result field. Factor 2 is required and contains two subfactors. The first is a numeric duration and the second must be a valid duration code indicating the type of duration (year *Y, month *M, etc.). The duration code must be consistent with the result field data type. For example, You can add a year, month or day duration but not a minute duration to a date field. For list of duration codes and their short forms see Table 14-3.
ALLOC	**Allocates main storage and sets a pointer.** RPGIV has extended the RPG language with pointer operations. The ALLOC operation allocates storage in the default heap of the numeric length specified in Factor 2. The Result field is a pointer set to point to the new heap storage. This storage, though allocated is uninitialized and thus needs additional work to be usable. The result field must be a basing pointer scalar variable (a standalone field, data structure subfield, table name, or array element).
CALLB	Call Bound Procedure written in any ILE Language. RPGIV is an ILE language. As such it uses the ILE programming model which permits incomplete modules to be bound (linked) together to create executable programs or *PGM objects. RPGIV also supports procedures. Procedures are most often referred to as the natural building blocks for ILE applications. You can think of a procedure then as a hybrid between subroutines and external called programs. The CALLB operation is used to call bound procedures written in any of the ILE languages. The notion of an operation

	extender as implemented in the RPG/400 half-adjust column has been expanded with RPGIV. Extenders are now suffixes to normal op-codes. The operation extender "D" may be used to include operational descriptors for the procedure call. Operational descriptors provide the programmer with run-time resolution of the exact attributes of character or graphic strings passed (that is, length and type of string). Factor 2 is required and must be a literal or constant containing the name of the procedure to be called, or a procedure pointer containing the address of the procedure to be called. See RPGIV procedure operations in Chapter **** for more details.
CALLP	**Call Prototyped Procedure or Program.** CALLP uses a "free-form" syntax. You use the name operand to specify the name of the prototype of the called program or procedure, as well as any parameters to be passed. (similar to calling a BIF) The compiler uses the prototype name to obtain an external name, if required, for the call. If the keyword EXTPGM is specified on the prototype, the call will be a dynamic external call; otherwise it will be a bound procedure call. A prototype for the program or procedure being called must be included in the definition specifications preceding the CALLP. See RPGIV procedure operations in Chapter **** for more details
DEALLOC	**Deallocates storage back to the default heap**. The operation frees one previous allocation of heap storage. The pointer name that you provide in Factor 2 is a pointer that must be the value previously set by a heap-storage allocation operation (either an ALLOC operation in RPG, or some other heap-storage allocation mechanism). It is not sufficient to simply point to heap storage; the pointer must be set to the beginning of the specific allocation that is to be deallocated. The storage pointed to by the pointer is freed for subsequent allocation by this program or any other in the activation group. If operation code extender N is specified, the pointer is set to *NULL after a successful deallocation.
DOU	**Do Until (Free Form)** RPGIV also brings with it free

	format operations (covered in Chapter ****.) The DOU operation code is a free format RPG operation that precedes a group of operations which you want to execute at least once and possibly more than once. Its function is similar to that of the DOUxx operation code. As with the DOUxx, the associated ENDDO statement marks the end of the group. It differs in that the logical condition is expressed by what is called an "indicator valued expression." An example of such an operation follows: **dou *in01 or (Field2 > Field3).** As with normal DO operations, those instructions within the loop are performed until the indicator valued expression is true. There are also two op code extenders, "M" & "R" available which may be needed to affect the precision of the operation.
DOW	**Do While (Free Form)** DOW is a free format operation code that precedes a loop of instructions, which you want to process when a given condition exists. It is very similar in function to that of the DOWxx operation code but it differs in form. An associated ENDDO statement marks the end of the Do group. Rather than a mix of op-code and factors, the logical condition of the DOW is expressed by an "indicator valued expression." The loop is performed while the indicator valued expression is true. "M" and "R" op-code extenders are available to affect the precision of the expression if necessary.
ENDFOR	**Ends a FOR Group.** An ENDFOR operation indicates the end of the FOR group. There are no operands.
ENDMON	**Ends a Monitor Group.** An ENDMON operation Indicates the end of the MONITOR group. There are no operands.
ELSEIF	**Else and IF Combination Operation.** The ELSEIF operation is a clever combination of an ELSE operation with an IF operation. Its major advantage over the split operation is that it avoids the need for an additional level of nesting. It uses the extended Factor 2 facility of RPGIV to provide the space for what is called the "indicator valued expression." The IF part of the operation code allows a series of operation codes to be

	processed if a condition is met. Its function is similar to that of the IFxx operation code. Rather than comparing Factors, the IF expression is evaluated. The operations controlled by the ELSEIF operation are performed when the expression in the indicator-expression operand is true (and of course the expression for the previous IF or ELSEIF statement was false).
EVAL	**Evaluate Expression.** The EVAL operation code permits semi-free form expressions to be used in the Extended Factor 2 area of the RPG calculations statement. It evaluates an assignment statement of the form result=expression. The expression is evaluated and the result placed in result (left side of equal sign). The expression may yield any of the RPG data types. On a free-form calculation specification, the EVAL operation code name itself may be omitted if no op-code extenders are needed.
EVALR	**Evaluate Expression – Right Adjust Result.** The EVALR is similar to EVAL. However, the result will be right justified and padded with blanks on the left, or truncated on the left as required. Unlike the EVAL operation, the result of EVALR can only be of type character, graphic, or UCS-2.
EXTRCT	**Extracts part of date / time/ timestamp into a field.** The EXTRCT operation code is a very powerful RPGIV operator which returns to the Result field, the requested sub field from the date, time, or timestamp specified in Factor 2. This can be (1) the year, month or day part of a date or timestamp field, (2) the hours, minutes or seconds part of a time or timestamp field, (3) the microseconds part of the timestamp field to the field specified in the Result field. The duration code (Table 14-3) must be consistent with the data type of Factor 2. For a character result field, the data is put left adjusted into the result field.
FOR	**FOR loop with index and increment.** FOR is another loop type made famous in the BASIC language that is similar in function to a DO loop. The FOR operation begins the loop which consists of a group of operations and it controls the number of times the group will be

	processed. The operation uses only the Extended Factor 2 form and is thus specified in much the same way as an Extended Factor 2 expression. To indicate the number of times the group of operations is to be processed, you specify an index name, a starting value, an increment value, and a limit value. The optional starting, increment, and limit values can also be used in an RPG free-form expression. The ENDFOR or an associated END statement marks the end of the FOR group.
IF	**If statement (Free form).** The IF operation uses the Extended Factor 2 form of RPGIV calculations. The operation code starts a group that allows a series of operation codes to be processed if a condition is true. Its function is similar to that of the IFxx operation code The difference is that the logical condition is expressed by an "indicator valued expression." The operations controlled by the IF operation are performed when the expression is true.
LEAVESR	**Exits Subroutine from Any Point.** 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. There are no operands.
MONITOR	**Begin Monitor Group -- Monitors for Errors with ON-ERROR.** To enable more control of exception handling in RPG IV, the MONITOR operation code (Or group) is added. It consist of the following: (1) A MONITOR block, (2) One or more ON-ERROR blocks, and (3) an ENDMON operation (Or END opcode). If an error occurs when the monitor block is processed, control is passed to the appropriate ON-ERROR group. There are no operands.
ON-ERROR	**Specifies Types of Errors to Monitor.** This operation works with Extended Factor 2 to provide a list of error IDs. You specify which error conditions the on-error block handles in the list of exception IDs. You can specify any combination of the following, separated by colons: (1) nnnnn -- A status code, (2) *PROGRAM – Handles all program-error status codes, from 00100 to

	00999, (3) *FILE -- Handles all file-error status codes, from 01000 to 09999, (4) *ALL – This default handler, takes care of both program-error and file-error codes, from 00100 to 09999. When all the statements in an on-error block have been processed, control passes to the statement following the ENDMON statement.
REALLOC	**Reallocate main storage with a new length.** This operation alters the prior memory allocation by changing the length of the heap storage pointed to by the Result-field pointer to the new length as specified in Factor 2. The result field of REALLOC contains the basing pointer variable, which must contain the value previously set by a heap-storage allocation operation (either an ALLOC or REALLOC operation in RPG -- or some other valid heap-storage function.) As with the DEALLOC, it is not sufficient to simply point to heap storage; the pointer must be set to the beginning of an allocation. The new storage amount is allocated and the value of the old storage is copied to the new storage. Following this, the old storage is deallocated. If the new length is shorter, the value is truncated on the right. If the new length is longer, the new storage to the right of the copied data is uninitialized. The Result field pointer is set to point to the new storage.
SUBDUR	**Subtract a Duration to a Date (Days, Months, Years).** The SUBDUR operation can be used to subtract a duration specified in factor 2 from a field or constant specified in factor 1 and place the resulting Date, Time or Timestamp in the field specified in the Result field. If factor 1 is not specified then the duration is subtracted from the field specified in the result field. Factor 2 is required and contains two subfactors. The first is a numeric field. The second subfactor must be a valid duration code indicating the type of duration (YR, Mo Day, etc. See Table 14-3 for valid duration codes. The Result field must be a date, time or timestamp data type field, array or array element. The SUBDUR operation can also be used to calculate a duration between: two dates, a date and a timestamp, two times, a time and a timestamp, and two timestamps. The result is

	a number of whole units, with any remainder discarded. For example, 62 minutes is equal to 1 hour and 57 minutes is equal to 0 hours. The result field consists of two subfactors. The first is the name of a numeric element in which the result of the operation will be placed. The second subfactor contains a duration code with the type of duration.
XML-INTO	**Bring in an XML document.** The XML-INTO operation has two forms as follows: (1) XML-INTO{ (EH) } variable %XML(xmlDoc { : options }); (2) XML-INTO{ (EH) } %HANDLER(handler : commArea) %XML(xmlDoc { : options }); The newest IBM RPGIV op codes include XML-INTO which reads the data from an XML document in one of two ways: (1) directly into a variable or (2) gradually into an array parameter that it passes to the procedure specified by %HANDLER. Various options may be specified to control the operation. The first operand specifies the target of the parsed data. It can contain a variable name or the % HANDLER built-in function. The second operand contains the %XML built-in function specifying the source of the XML document and any options to control how the document is parsed. It can contain XML data or it can contain the location of the XML data. From the looks of this XML operation and the next, XML and its RPG debut are not really ready for prime time. Look at how simple the other RPG codes have been constructed to understand that IBM has lots of work to do in this area..
XML-SAX	**Parse XML using SAX.** The newest IBM RPGIV op codes include XML-SAX which initiates a SAX parse for the XML document specified by the %XML built-in function. The syntax of this basically free-form expression is as follows: **XML-SAX{ (e) } %HANDLER(eventHandler : commArea) %XML(xmldocument { : saxOptions });** The XML-SAX operation begins by calling an XML parser which begins to parse the document. When the parser discovers an event such as finding the start of an

	element, finding an attribute name, finding the end of an element etc., the parser calls the eventHandler with parameters describing the event. The commArea operand is a variable that is passed as a parameter to the eventHandler providing a way for the XML-SAX operation code to communicate with the handling procedure. When the eventHandler returns, the parser continues to parse until it finds the next event and calls the eventHandler again.

Table 14-3 Duration Codes for Date Operations

Unit	**Built-In Function**	**Duration Code**
Year	%YEARS	*YEARS or *Y
Month	%MONTHS	*MONTHS or *M
Day	%DAYS	*DAYS or *D
Hour	%HOURS	*HOURS or *H
Minute	%MINUTES	*MINUTES or *MN
Second	%SECONDS	*SECONDS or *S
Microsecond	%MSECONDS	*MSECONDS or *MS

Taking a look at the old and new RPGIV op-codes makes it easy to conclude that RPGIV is both the same and lots more than RPG/400. The additions to RPGIV have stretched the capabilities of the language to the point that it can provide the best business function as well as very powerful operations that may be rightfully categorized in the computer science area.

Table 14-4 takes these new operations described in detail in Table 14-2 and places them in their most simplistic form –> op-codes with a mission and a format. Each of the operations described in Table 14-2 are outlined in terms of their parameters and their format in Table 14-4. After checking out table 14-2 and its adjunct, Table 14-3, Table 14-4 is the right medicine for the RPG programmer wanting to see how the operations look when in action.

Table 14-4 All New RPG IV Operations / Parameters Alphabetical

Code	Factor1	Factor2	Result	I1	I2	I3
ADDDUR	**Date/Time**	**Duration: Duration code**	**Date/Time**	**NA**	**ER**	**NA**
ALLOC(E)		**Length**	**Pointer**	**NA**	**ER**	**NA**
CALLB (D,E)		**Procedure name or procedure pointer**	**PLIST name**	**NA**	**ER**	**LR**
CALLP		**name{ (parm1 {:parm2...}) }**	**Extended Factor 2**	**NA**	**NA**	**NA**
DEALLOC (E/N)			**Pointer-name**	**NA**	**ER**	**NA**
DOU (M/R)		**indicator-expression**	**Extended Factor 2**			
DOU (M/R)		**indicator-expression**	**Extended Factor 2**			
ENDFOR						
ENDMON						
ELSEIF (M/R)		**indicator-expression**	**Extended Factor 2**			
EVAL (H M/R)		**Assignment Statement**	**Extended Factor 2**			
EVALR		**Assignment**	**Extended**			

(M/R)		**Statement**	**Factor 2**			
EXTRCT(E)		**Date/Time: Duration Code**	**Target**	**NA**	**ER**	**NA**
FOR		**index-name = start-value BY increment TO | DOWNTO limit**	**Extended Factor 2**			
IF (M/R)		**indicator-expression**	**Extended Factor 2**			
LEAVESR						
MONITOR						
ON-ERROR		**List of exception IDs**	**Extended Factor 2**			
REALLOC		**Length**	**Pointer**	**NA**	**ER**	**NA**
SUBDUR(E) (Duration)	**Date/time / Timestamp**	**Date/Time/ Timestamp**	**Duration: Duration code**	**NA**	**ER**	**NA**
SUBDUR(E) (New Date)	**Date/time / Timestamp**	**Duration: Duration code**	**Dat/Time / Timestamp**	**NA**	**ER**	**NA**
XML-INTO		***receiver* %XML(*xmlDoc* {: *options* })**	**Extended Factor 2**			
XML-INTO		**%HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: *options* })**	**Extended Factor 2**			
XML-SAX		**%HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: *options* })**	**Extended Factor 2**			

RPGIV Built-In Functions (BIFs)

Besides additional function provided via new op-codes, RPGIV also provides a wealth of new and/or easier to use function by its “Buuilt-In Functions or BIFs. The BIFS are similar to operation codes in that they perform operations on data that you specify. Built-in functions can be used in expressions in Free-Form RPG IV or with the EVAL statement

described in more detail with an example below. Additionally, constant-valued built-in functions can be used in named constants. These named constants can be used in any specification. All built-in functions have the percent symbol (%) as their first character. The general syntax of RPGIV 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. An expression argument can also include a built-in function. The following example illustrates a compound expression with multiple BIFs. We'll pick BIFs up later in the book as we demonstrate practical code in RPG/400, RPGIV, and RPGIV with BIFs.

The list of all BIFs and the functions they provide is included in Table 14-8. Even before getting there, you can examine the individual built-in function descriptions used in the examples in Figures 14-6 and 14-7 for a look at the types of arguments that are allowed in BIF operations. It helps to remember that unlike operation codes, built-in functions return a value rather than placing a value in a calc spec Result field.

The examples in Figure 14-6 and 14-7 illustrate this difference. Let's set up the examples now so they have more meaning as you look at the first very powerful one line BIF in Figure 14-6. It springs from the EVAL operation and it will give you your first look see at the RPGIV Extended Factor 2 operation for the first time. Explanations of the three built-in functions used in the Figure 14-6 example are shown in Table 14-5.

Table 14-5 Three BIFS for the Example

%TRIM	**Purpose: Trims blanks at edges**; Format: %TRIM**(string)** Returns string less any leading and trailing blanks.
%SUBST	**Purpose: Get SubString;** Format: %SUBST**(string:start{:length})** The %SUBST returns a portion of an argument string – a.k.a a substring. It may also be used as the result of an assignment with the EVAL operation code. The start parameter represents the starting position of the

	substring. The length parameter represents the length of the substring.
%SIZE	**Purpose: Get Size in Bytes**. Returns size of variable or literal Formats: %SIZE(variable) %SIZE(literal) %SIZE(array{:*ALL}) %SIZE(table{:*ALL}) %SIZE(multiple occurrence data structure{:*ALL}) %SIZE returns the number of bytes occupied by the constant or field. If the argument is an array name, table name, or multiple occurrence data structure name, the value returned is the size of one element or occurrence. If *ALL is specified as the second parameter for %SIZE, the value returned is the storage taken up by all elements or occurrences. Returns size of variable or literal

The short example immediately below in figure 14-6 returns a value to the field named RES that has been defined elsewhere in the RPGIV program. RES will contain the trimmed string that consists of a field named “A” containing “Toronto” and the substring of “Ontario, Canada” starting with the C in Canada for a length provided by the %Size function of a field named “B” (30 characters) containing ‘Ontario Canada ’ minus the constant digit 20. At the end, RES will contain ‘Toronto, Canada’ with just one blank between the comma and the “C” in Canada. The first line of the example in Figure 14-5 shows the right-side of the C-spec with the extended Factor 2 form of RPGIV.

Figure 14-6 RPGIV Extended Factor 2, EVAL, & BIF

```
C…+Opcode(E)+Extended-factor2+++++++++++++++++++++++++ * *
C …+ EVAL    RES = %TRIM(A + %SUBST(B:11:%SIZE(B) - 20))
```

To evaluate this statement it helps to know the following:

A   is equal to the string, ‘ Toronto,”
B   is equal to the string, ‘ Ontario, Canada ’

RES becomes the string, 'Toronto, Canada'

The above example shows a complex expression with multiple nested built-in functions.

%TRIM takes as its argument a string.

In this example, the argument is the concatenation of string A and the string returned by the %SUBST built-in function. The %SUBST BIF returns a substring of string B starting at position 11 and continuing for the length returned by %SIZE minus 20. %SIZE will return the length of string B.

If A is the string ' Toronto,' and B is the string ' Ontario, Canada ' then the argument for %TRIM will * be ' Toronto, Canada ' and RES will have the value 'Toronto, Canada'.

Now, let's make this all a bit easier by providing a full RPGIV program complete with data definitions in Figure 14-7. The BIFs used don't make real sense unless you know how big the fields really are. To make the code readable in this narrow context, as you can see, we took some liberties by chopping off some space within the D & C specification (Op-code and Factor 1) and we squeezed the LR into the picture though it deserves its own column way out to the right.

Figure 14-7 Full RPGIV Program to demonstrate BIFS

```
DName++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++
DA             S                 12     INZ('    Toronto,')
DB             S                 30     INZ(' Ontario, Canada ')
DRES1          S                 20
DRES2          S                  4S 0
DRES3          S                 21
DLEFT          S                  4
CL0N01Factor1++OpcodeExtExtended-factor2+++++++++
C                EVAL   RES1=%TRIM(A+%SUBST(B:11:%SIZE(B)-20)
C                EVAL   RES2=%SIZE(B)
C                EVAL   RES3=%SUBST(B:11:%SIZE(B)- 20)
C      'RES1='   DSPLY                        RES1
C      'RES2='   DSPLY                        RES2
C      'RES3='   DSPLY                        RES3
C                MOVEL      'LEFT'            LEFT
C      'LEFT='   DSPLY                        LEFT
```

```
C                    SETON                                       LR
```

So, what does this code in Figure 14-7 do? The major BIF itself has already been explained. However, when you are learning RPG or any language, it is good to learn the language incrementally for example by walking through the smallest parts of big expressions. That's what we did. Notice we took the one big EVAL and made two more out of it. RES2 returns the length value that is actually used in the %SUBST BIF. RES2 brings back the substring value "Canada," so you can see how the trimmed version RES1 actually gets built. Notice that the length of field "B" is 30. This code does not work as well if it is anything else.

Once the program calculates the three results (RES1, RES2, and RES3) to communicate the results to us, we use the very handy DSPLY operation. I use DSPLY all the time for debugging when I don't believe my problem is serious enough for the fine DEBUG tools that are available. The three DSPLY operations project the constant in Factor 1 to the job log and next to it DSPLY places the value of the field in the Result field. You can even put an indicator in the Result field if that is what you are interested in examining. Finally, you see a four position alpha field called LEFT getting filled with a MOVEL operation with the word "LEFT." I put this in so that on the DSPLY view, you would know the leftmost position of the data being shown so that you would believe there were no blanks to the left of the trimmed RES1 field. Finally, so the RPG program knows that it is OK to end, the code sets on LR using the SETON operation. The job log results are shown in Figure 14-8.

Figure 14-8 Job Log "Printout" of DSPLY Operations

```
*N
DSPLY  RES1=    Toronto, Canada
*N
DSPLY  RES2=      30
*N
DSPLY  RES3=    Canada
*N
DSPLY  LEFT=    LEFT
```

Now that we have taken a big byte out of the mystery of BIFS, let's get adventurous and show them all with a brief description in Table 14-9. Then, let's follow that with a more lengthy description of each BIF in Table 14-10. After these two charts, you will have a pretty good idea of what BIFS are available in RPGIV and how they can be valuable in your coding.

Since all of these BIFS operate without any help from Factor1, Factor2, the Result field or the resulting indicators, there is no need for a formatted operation table as we did in Chapter 7 for all operations and as we did in Table 14-4 for the new RPGIV operations.

Table 14-9 BIFs and Functions Provided

BIF Name	Provided Function
%ABS	Numeric absolute value of expression
%ADDR	Variable name address of variable
%ALLOC	# of bytes storage to allocate pointer storage
%BITAND	Char, numeric bit wise ANDing bits of all args
%BITNOT	Char, numeric bit-wise reverse of bits of the args
%BITOR	Char, numeric bit-wise ORing bits of all args
%BITXOR	Char, numeric bit-wise exclusive ORing two args
%CHAR	Graphic, UCS-2, numeric, date, etc. in char fmt.
%CHECK	Check for Certain Chars in a String (L to R)
%CHECKR	Check Reverse for Chars in a String (R to L)
%DATE	Date -- system date if none is specified
%DAYS	# days as a duration
%DEC	Changes expression to packed decimal
%DECH	Changes expression to packed decimal – rounded up
%DECPOS	Numeric expression -- # of decimal digits
%DIFF	Difference between two dates, times
%DIV	Divide two #s function
%EDITC	Edit value using an edit code.
%EDITFLT	Convert to Float External Representation.
%EDITW	Edit value using an Edit word:
%ELEM	# of elements or occurrences
%EOF	Test for End of File
%EQUAL	Return exact match condition.
%ERROR	Most recent operation was an error

%FIELDS	List of fields to be updated not applicable
%FLOAT	Convert value to floating format.
%FOUND	Successful found record
%GRAPH	Expression in graphic format
%HOURS	# of hours as a duration
%INT	Change to integer format
%INTH	Change to integer format – rounded up
%KDS	Data structure with keys
%LEN	Get or set length.
%LOOKUPxx	argument: array with index
%MINUTES	# of minutes as a duration
%MONTHS	# of months as a duration
%MSECONDS	# of microseconds as a duration
%NULLIND	Null-capable field name value in indicator
%OCCUR	Current occurrence of multiple-occurrence DS
%OPEN	Opens a closed file
%PADDR	Get procedure address
%PARMS	# of parameters passed to procedure
%REALLOC	Numeric pointer: to allocated storage
%REM	Division - the remainder from div of 2 args
%REPLACE	Replacement string
%SCAN	Returns searched for value or zero
%SECONDS	# of seconds as a duration
%SHTDN	Returns value indicating shutdown (1 or 0)
%SIZE	Returns size of variable or literal
%SQRT	Square root of a numeric value
%STATUS	0 if no I/O error for file
%STR	String characters addressed by pointer argument
%SUBARR	Return a subset of an array
%SUBDT	Returns a portion of date or time value
%SUBST	Returns a substring
%THIS	The class instance for the native method
%TIME	Brings back system time if none is specified
%TIMESTAMP	Brings back current timestamp if none specified
%TLOOKUPxx	Checks for match and returns '*ON' or '*OFF'
%TRIM string	Trims string with left, right blanks or specified
%TRIML string	Trims string with left blanks or specified
%TRIMR string	Trims string with right blanks or specified
%UCS2	Brings back value in UCS-2 format
%UNS	Brings back value in unsigned format
%UNSH	Brings back rounded-up value - unsigned format
%XFOOT	Array expression sum of the elements
%XLATE	Translate String (eg Upper Case to Lower Case).
%YEARS	# of years as a duration

Table 14-10 BIFs and Functions Details

BIF Name	Provided Function
%ABS	**Numeric absolute value of expression. Format:** %ABS(numeric expression) Example: F**8 = %abs (F8);** %ABS returns the absolute value of the numeric expression specified as the parameter. %ABS may be used either in expressions or as parameters to keywords.
%ADDR	**Variable name address of variable. Format:** %ADDR(variable); %ADDR(variable(index)); %ADDR(variable(expression)) Example: **IF %ADDR (CHAR10) = %ADDR (SUBF);** %ADDR returns a value of type basing pointer. The value is the address of the specified variable. It may only be compared with and assigned to items of type basing pointer.
%ALLOC	**# of bytes storage to allocate pointer storage. Format:** %ALLOC(num) Example: **Pointer = %ALLOC(200);** %ALLOC returns a pointer to newly allocated heap storage of the length specified. The newly allocated storage is uninitialized. The length specified must be between 1 and 16776704.
%BITAND	**Char, numeric bit wise ANDing bits of all arguments. Format::** %BITAND (Bitwise AND Operation) %BITAND(expr:expr{:expr...}) Example below from IBM manual courtesy of IBM: %BITAND returns the bit-wise ANDing of the bits of all the arguments. That is, the result bit is ON when all of the corresponding bits in the arguments are ON, and OFF otherwise. The arguments to this BIF can be either character or numeric. For numeric arguments, if they are not integer or unsigned, they are first converted to integer. %BITAND can have two or more arguments. All arguments must be the same type, either character or numeric. The result type is the same as the types of the arguments. For numeric arguments, the result is unsigned if all arguments are unsigned, and integer otherwise. The length is the length of the largest operand. If the arguments have different lengths, they are padded on the left with bit zeros for numeric arguments. Shorter character arguments are padded on the right with bit ones. %BITAND can be coded in any expression. It can also be coded as the argument to a File or Definition Specification keyword if all

arguments are known at compile-time. If all arguments of this built-in function are hex literals, the compiler produces a constant-folded result that is a hex literal.

Examples for &BITAND and &BITOR

```
D const       c                x'0007'
D ch1         s           4a   inz(%BITNOT(const))
 * ch1 is initialized to x'FFF84040'

D num1        s           5i 0 inz(%BITXOR(const:x'000F'))
 * num is initialized to x'0008', or 8

D char2a      s           2a
D char2b      s           2a
D uA          s           5u 0
D uB          s           3u 0
D uC          s           5u 0
D uD          s           5u 0

C                 eval   char2a = x'FE51'
C                 eval   char2b = %BITAND(char10a : x'0F0F')
 * operand1   = b'1111 1110 0101 0001'
 * operand2   = b'0000 1111 0000 1111'
 * bitwise AND:   0000 1110 0000 0001
 * char2b = x'0E01'
C                 eval   uA = x'0123'
C                 eval   uB = x'AB'
C                 eval   uc = x'8816'
C                 eval   uD = %BITOR(uA : uB : uC)
 * operand1  = b'0000 0001 0010 0011'
 * operand2  = b'0000 0000 1010 1011'(fill w x'00')
 * operand3  = b'1000 1000 0001 0110'
 * bitwise OR:    1000 1001 1011 1111
 * uD = x'89BF'
```

%BITNOT **Char, numeric bit-wise reverse of bits of the arguments.**

Format: %BITNOT(expr)

Example:

```
D const   c          x'0007'
D ch1     s       4a   inz(%BITNOT(const))
```

*** ch1 is initialized to x'FFF84040'**

%BITNOT returns the bit-wise inverse of the bits of the argument. That is, the result bit is ON when the corresponding bit in the argument is OFF, and OFF otherwise.

The argument to this built-in function can be either character or numeric. For numeric arguments, if they are not integer or unsigned, they are first converted to integer. If the value does not fit in an 14-byte integer, a numeric overflow exception is issued. %BITNOT takes just one argument. The result type is the same as the types of the arguments. For numeric arguments, the result is unsigned if all arguments are unsigned, and integer otherwise. The length is the length of the largest operand. If the arguments have different lengths, they are

padded on the left with bit zeros for numeric arguments.

%BITOR **Char, numeric bit-wise ORing bits of all arguments.**

Format: %BITOR(expr:expr{:expr...})

Example: See under %BITAND

%BITOR returns the bit-wise ORing of the bits of all the arguments. That is, the result bit is ON when any of the corresponding bits in the arguments are ON, and OFF otherwise. The arguments to this built-in function can be either character or numeric. For numeric arguments, if they are not integer or unsigned, they are first converted to integer. If the value does not fit in an 14-byte integer, a numeric overflow exception is issued. %BITOR can have two or more arguments. All arguments must be the same type, either character or numeric. However, when coded as keyword parameters, these two BIFs can have only two arguments. The result type is the same as the types of the arguments. For numeric arguments, the result is unsigned if all arguments are unsigned, and integer otherwise. The length is the length of the largest operand. If the arguments have different lengths, they are padded on the left with bit zeros for numeric arguments. Shorter character arguments are padded on the right with bit zeros. %BITOR can be coded in any expression. It can also be coded as the argument to a File or Definition Specification keyword if all arguments are known at compile-time. If all arguments of this built-in function are hex literals, the compiler produces a constant-folded result that is a hex literal.

%BITXOR **Char, numeric bit-wise exclusive ORing two arguments.**

Format: Format: %BITXOR(expr:expr)

Examples: **&BITXOR X'12' X'22'** *results in* X'30'
&BITXOR X'1211' X'22' *results in* X'3011'

%BITXOR returns the bit-wise exclusive ORing of the bits of the two arguments. That is, the result bit is ON when just one of the corresponding bits in the arguments are ON, and OFF otherwise. The argument to this BIF can be either character or numeric. For numeric arguments, if they are not integer or unsigned, they are first converted to integer. If the value does not fit in an 14-byte integer, a numeric overflow exception is issued. %BITXOR takes just two arguments. The result type is the same as the types of the arguments. For numeric arguments, the result is unsigned if all arguments are unsigned, and integer otherwise. The length is the length of the largest operand. If the arguments have different lengths, they are padded on the left with bit zeros for numeric arguments. Shorter character arguments are padded on the right with bit

	zeros. %BITXOR can be coded in any expression. It can also be coded as the argument to a File or Definition Specification keyword if all arguments are known at compile-time. If all arguments of this built-in function are hex literals, the compiler produces a constant-folded result that is a hex literal.
%CHAR	**Graphic, UCS-2, numeric, date, etc. in char fmt. Format:** %CHAR(expression{:format}) Example: **Res = 'Time now ' + %SUBST (%CHAR(time):1:5) + '.';** %CHAR converts the value of the expression from graphic, UCS-2, numeric, date, time or timestamp data to type character. The converted value remains unchanged, but is returned in a format that is compatible with character data. If the parameter is a constant, the conversion will be done at compile time.
%CHECK	**Check for Certain Chars in a String (L to R). Format:** %CHECK(comparator : base {: start}) Example: **pos = %check (delimiters : string);** %CHECK returns the first position of the string base that contains a character that does not appear in string comparator. If all of the characters in base also appear in comparator, the function returns 0. The check begins at the starting position and continues to the right until a character that is not contained in the comparator string is found. The starting position defaults to 1. The third parameter is optional.
%CHECKR	**Check Reverse for Chars in a String (R to L). Format:** %CHECKR(comparator : base {: start}) Example: **%len(string1) = %checkr(padChars:string1);** %CHECKR returns the last position of the string base that contains a character that does not appear in string comparator. If all of the characters in base also appear in comparator, the function returns 0. The check begins at the starting position and continues to the left until a character that is not contained in the comparator string is found. The starting position defaults to the end of the string The first parameter must be of type character, graphic, or UCS-2, fixed or |varying length. The third parameter is optional.
%DATE	**Date --returns system date if no parms specified. Format:** %DATE{(expression{:date-format})} Example: **date = %date(string:*MDY0);** %DATE converts the value of the expression from character, numeric, or timestamp data to type date. The converted value remains unchanged, but is returned as a date. The first parameter is the value to be converted. If you do not specify a

	value, %DATE returns the current system date. The second parameter is the date format for character or numeric input. Regardless of the input format, the output is returned in *ISO format. If the first parameter is a timestamp, *DATE, or UDATE, do not specify the second parameter. The system knows the format of the input in these cases.
%DAYS	**# days as a duration. Format:** %DAYS(number) Example: newdate = date + %DAYS(5); %DAYS converts a number into a duration that can be added to a date or timestamp value. %DAYS can only be the right-hand value in an addition or subtraction operation. The left-hand value must be a date or timestamp. The result is a date or timestamp value with the appropriate number of days added or subtracted. For a date, the resulting value is in *ISO format.
%DEC	**Convert to packed decimal format. Format:** %DEC(numeric expression{:precision:decimal places}) Example: **Result = %dec (s9 : 5: 0);** %DEC converts the value of the numeric expression to decimal (packed) format with precision digits and decimal places decimal positions. The 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. Parameters precision and decimal places may be omitted if the type of numeric expression is not float. If these parameters are omitted, the precision and decimal places are taken from the attributes of numeric expression.
%DECH	**Convert to packed decimal format Half adjust. Format::** %DECH(numeric expression :precision:decimal places) Example: **Result = %dech (f8: 5: 2);** %DECH is the same as %DEC except that if numeric expression is a decimal or float value, half adjust is applied to the value of numeric expression when converting to the desired precision. Unlike, %DEC, all three parameters are required.
%DECPOS	**Numeric expression -- # of decimal digits. Format:** %DECPOS(numeric expression) Example: Result = %decpos (p7); %DECPOS returns the number of decimal positions of the numeric variable or expression. The value returned is a constant, and so may participate in constant folding. The numeric expression must not be a float variable or expression.

%DIFF	**Difference between 2 dates, times, timestamps. Format:** 1. %DIFF(op1:op2:*MSECONDS|*SECONDS| *MINUTES| *HOURS|*DAYS|*MONTHS|*YEARS) 2. %DIFF(op1:op2:*MS|*S|*MN|*H|*D|*M|*Y) Example: Num_days = %DIFF (loandate: duedate: *DAYS); %DIFF produces the difference (duration) between two date or time values. The first and second parameters must have the same, or compatible types. Many combinations are possible.
%DIV	**Divide two #s function Return integer quotient: Format:** %DIV(n:m) Example: Result = %DIV(A:B); %DIV returns the integer portion of the quotient that results from dividing operands n by m. The two operands must be numeric values with zero decimal positions.
%EDITC	**Edit value using an edit code. Format:** %EDITC(numeric : editcode {: Value)) Value choices = *ASTFILL | *CURSYM | currency-symbol Example: **EVAL Result = 'Annual salary is ' +** **%trim(%editc(salary * 12:'A': *CURSYM))** The &EDITC function returns a character result representing the numeric value edited according to the edit code. In general, the rules for the numeric value and edit code are identical to those for editing numeric values in output specifications. The third parameter is optional, and if specified, must be one of the values shown above. The result of %EDITC is always the same length, and may contain leading and trailing blanks.
%EDITFLT	**Convert to Float External Representation. Format:** %EDITFLT(numeric expression) Example: **Reslt = 'Float value is' + %editflt (f8 - 4E4) + '.';** %EDITFLT converts the value of the numeric expression to the character external display representation of float. The result is either 14 or 23 characters. If the argument is a 4-byte float field, the result is 14 characters. Otherwise, it is 23 characters
%EDITW	**Edit value using an Edit word: Format**: %EDITW(numeric : editword) Example: **D editwd C '$, , **Dollars& &Cents'** **Result = 'Annual salary ' + %editw(salary * 12 : editwd);** This function returns a character result representing the numeric value edited according to the edit word as in the above example. The rules for the numeric value and edit word are identical to those for editing numeric values in output specifications. The edit word must be a character constant.
%ELEM	**# of elements or occurrences: Format:**

%ELEM(table_name) %ELEM(array_name)
%ELEM(multiple_occurrence_data_structure_name)
Example: **Resultary = %elem (arr1d);**
Resulttbl = %elem (table);
ResultDS = %elem (mds);
%ELEM returns the number of elements in the specified array, table, or multiple-occurrence data structure. The value returned is in unsigned integer format (type U). It may be specified anywhere a numeric constant is allowed in the definition specification or in an expression in the extended Factor 2 field. The parameter must be the name of an array, table, or multiple occurrence data structure.

%EOF **Test for End of File. Format:**
%EOF (Return End or Beginning of File Condition)
%EOF{(file_name)}
Example: **IF %EOF(FILE1) AND %EOF(FILE2);**
%EOF returns '1' if the most recent read operation or write to a subfile ended in an end of file or beginning of file condition; otherwise, it returns '0'. The operations that set %EOF are: READ, READC, READE (Read Equal Key), READP, READPE, WRITE (subfile only). The following operations, if successful, set %EOF(filename) off. If the operation is not successful, %EOF(filename) is not changed. %EOF with no parameter is not changed by these operations: CHAIN, OPEN, SETGT, SETLL. When a full-procedural file is specified, this function returns '1' if the previous operation in the list above, for the specified file, resulted in an end of file or beginning of file condition. For primary and secondary files, %EOF is available only if the file name is specified. It is set to '1' if the most recent input operation during *GETIN processing resulted in an end of file or beginning of file condition. Otherwise, it returns '0'. This function is allowed for input, update, and record-address files; and for display files allowing WRITE to subfile records.

%EQUAL **Return exact match condition. Format:**
%EQUAL{(file_name)}
Examples: Setll Cust CustRec;
if %equal;
C WHEN %EQUAL
%EQUAL returns '1' if the most recent relevant operation found an exact match; otherwise, it returns '0'. The operations that set %EQUAL are: SETLL (Set Lower Limit), LOOKUP (Look Up a Table or Array Element) If %EQUAL is used without the optional file_name parameter, then it returns the

	value set for the most recent relevant operation. For the SETLL operation, this function returns '1' if a record is present whose key or relative record number is equal to the search argument. For the LOOKUP operation with the EQ indicator specified, this function returns '1' if an element is found that exactly matches the search argument.
%ERROR	**Most recent operation was an error**. **Format:** %ERROR (Return Error Condition) Example: if %error; exsr ErrorSub; endif; Format: %ERROR returns '1' if the most recent operation with extender 'E' specified resulted in an error condition. This is the same as the error indicator being set on for the operation. Before an operation with extender 'E' specified begins, %ERROR is set to return '0' and remains unchanged following the operation if no error occurs. All operations that allow an error indicator can also set the %ERROR built-in function. The CALLP operation can also set %ERROR.
%FIELDS	**List of fields to be updated not applicable**
%FLOAT	**Convert value to floating format. Format:** %FLOAT(numeric expression) Example; **Result = %float (p1) / p2;** %FLOAT converts the value of the numeric expression to float format. This built-in function may only be used in expressions.
%FOUND	**Successful found record. Format:** %FOUND{(file_name)} Example: If %found (Master) and not %found (Gold); %FOUND returns '1' if the most recent relevant file operation found a record, a string operation found a match, or a search operation found an element. Otherwise, this function returns '0'. The operations that set %FOUND are: CHAIN, DELETE, SETGT, SETLL, CHECK, CHECKR, LOOKUP, SCAN (Scan String – however, the %SCAN BIF does not change the value of %FOUND.) If %FOUND is used without the optional file_name parameter, then it returns the value set for the most recent relevant operation. When a file_name is specified, then it applies to the most recent relevant operation on that file.
%GRAPH	**Expression in graphic Format:** %GRAPH(char-expr | graph-expr | UCS-2-expr { : ccsid }) Example:

```
D*Name+++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D  char      S              5A   inz('abcde')
 * %GRAPH built-in function is used to initialize a graphic field
D  Result    S             10G   inz (%graph ('oAABBCCDDEEi'))
```

```
D  ufield    S                2C   inz (%ucs2 ('oFFGGi'))
```

/FREE

Result = %graph (char) + %graph (ufield);

%GRAPH converts the value of the expression from character, graphic, or UCS-2 and returns a graphic value. The result is varying length if the parameter is varying length. The second parameter, ccsid, is optional and indicates the CCSID of the resulting expression. The CCSID defaults to the graphic CCSID related to the CCSID of the job. If CCSID (*GRAPH : *IGNORE) is specified on the control specification or assumed for the module, the %GRAPH built-in is not allowed. If the parameter is a constant, the conversion will be done at compile time. In this case, the CCSID is the graphic CCSID related to the CCSID of the source file. If the conversion results in substitution characters, a warning message is issued at compile time. At run time, status 00050 is set and no error message is issued.

%HOURS **# of hours as a duration. Format:**

%HOURS(number)

Example **// Determine the time in 3 hours**
Newtime = time + %HOURS(3);

%HOURS converts a number into a duration that can be added to a time or timestamp value. %HOURS can only be the right-hand value in an addition or subtraction operation. The left-hand value must be a time or timestamp. The result is a time or timestamp value with the appropriate number of hours added or subtracted. For a time, the resulting value is in *ISO |format.

%INT **Change to integer format. Format:**

%INT(numeric expression)

Example **Result = %int (p7) + 0.011;**

%INT converts the value of the numeric expression to integer. Any decimal digits are truncated. This built-in function may only be used in expressions. %INT can be used to truncate the decimal positions from a float or decimal value allowing it to be used as an array index.

%INTH **Change to integer format – rounded up. Format:**

%INTH(numeric expression)

Example Result1 = %int (p7)

%INTH is the same as %INT except that if the numeric expression is a decimal or float value, half adjust is applied to the value of the numeric expression when converting to integer type.

%KDS **Data structure with keys. Format:**

	%KDS(data-structure-name{:num-keys}) Example: Chain %kds(CustRecKeys) custRec; %KDS is allowed as the search argument for any keyed Input/Output operation (CHAIN, DELETE, READE, READPE, SETGT, SETLL) coded in a free-form group. The search argument is specified by the subfields of the data structure name coded as the first argument of the built-in function. The key data structure may be (but is not limited to), an externally described data structure with keyword EXTNAME(...:*KEY) or LIKEREC(...:*KEY).. The first argument must be the name of a data structure. This includes any subfield defined with keyword LIKEDS or LIKEREC. The second argument specifies how many of the subfields to use as the search argument. The individual key values in the compound key are taken from the top level subfields of the data structure. Subfields defined with LIKEDS are considered character data.
%LEN	**Get or set length. Format:** %LEN(expression) Example: Length = %len(num1); %LEN can be used to get the length of a variable expression or to set the current length of a variable-length field. The parameter must not be a figurative constant.
%LOOKUPxx	**Look up an array element: Generic Format:** %LOOKUP xx with xx = type of match %LOOKUP(arg : array {: startindex {: numelems}}) – exact match %LOOKUPLT(arg : array {: startindex {: numelems}}) – closest but less than %LOOKUPGE(arg : array {: startindex {: numelems}}) -- equal or closest but less than %LOOKUPGT(arg : array {: startindex {: numelems}}) -- closest but greater than %LOOKUPLE(arg : array {: startindex {: numelems}}) – equal or closest but greater than Example: **Result = %LOOKUP('Paris':arr);** If no value matches the specified condition, zero is returned. The search starts at index startindex and continues for ***numelems*** elements. By default, the entire array is searched. The first two parameters can have any type but must have the same type. They do not need to have the same length or number of decimal |positions. The third and fourth parameters must be non-float numeric values with zero decimal positions.

%MINUTES	**# of minutes as a duration Format:** %MINUTES(number) Example: **// Determine the time in 3 minutes** **Newtime = time + %MINUTES(3);** **%**MINUTES converts a number into a duration that can be added to a time or timestamp value. %MINUTES can only be the right-hand value in an addition or subtraction |operation. The left-hand value must be a time or timestamp. The result is a time or timestamp value with the appropriate number of minutes |added or subtracted. For a time field, the resulting value is in *ISO |format.
%MONTHS	**# of months as a duration. Format:** %MONTHS(number) Example: **Resultdate = duedate - %MONTHS(6);** %MONTHS converts a number into a duration that can be added to a date or timestamp value. %MONTHS can only be the right-hand value in an addition or subtraction |operation. The left-hand value must be a date or timestamp. The result is a date or timestamp value with the appropriate number of months added or subtracted. For a date, the resulting value is in *ISO format.
%MSECONDS	**Convert to # of microseconds as a duration. Format:** %MSECONDS(number) Example: **// Determine the time in 360 microseconds** **Newtime = time + %MSECONDS(360);** %MSECONDS (convert to microsoeconds) converts a number into a duration that can be added to a time or timestamp value. %MSECONDS can only be the right-hand value in an addition or subtraction operation. The left-hand value must be a time or timestamp. The result is a time or timestamp value with the appropriate number of microseconds added or subtracted. For a time, the resulting value is in *ISO format.
%NULLIND	**Null-capable field name value in indicator. Format:** %NULLIND(fieldname) Example: **if %nullind (DBField1);** The %NULLIND BIF can be used to query or set the null indicator for null-capable fields. %NULLIND can only be used in expressions in extended factor 2 or free-form RPG. When used on the right-hand side of an expression, this function returns the setting of the null indicator for the null-capable field. The setting can be *ON or *OFF. When used on the left-hand side of an expression, this function can be used to set the null indicator for null-capable fields to *ON or

	*OFF. The content of a null-capable field remains unchanged.
%OCCUR	**Returns current record of multi-occurrance DS. Format:** %OCCUR(dsn-name) Example: **/FREE** **Form 1: Result = %OCCUR(mds);** **Form 2: %OCCUR(mds) = 7;** %OCCUR gets or sets the current position of a multiple-occurrence data structure. When this function is evaluated for its value, it returns the current occurrence number of the specified data structure as an unsigned numeric value. When this function is specified on the left-hand side of an EVAL statement, or a free form equation, the specified number becomes the current occurrence number.
%OPEN	**Opens a closed file. Format:** %OPEN(file_name) Example: **If not %open (PRINTFILE);** **Open PRINTFILE;** %OPEN returns '1' if the specified file is open. A file is considered "open" if it has been opened by the RPG program during initialization or by an OPEN operation, and has not subsequently been closed. If the file is conditioned by an external indicator and the external indicator was off at program initialization, the file is considered closed, and %OPEN returns '0'.
%PADDR	**Get procedure address. Format:** %PADDR(string|prototype) Example: **EVAL PROC1 = %PADDR ('NextProg')** %PADDR returns a value of type procedure pointer. The value is the address of the entry point identified by the argument. %PADDR may be compared with and assigned to only items of type procedure pointer. The parameter to %PADDR must be a character constant or a prototype name. When a character constant is used, this identifies the entry point by name. The prototype must be a prototype for a bound call. The EXTPGM keyword cannot be used. The entry point identified by the prototype is the procedure identified in the EXTPROC keyword for the prototype. If the EXTPROC keyword is not specified, the entry point is the same as the prototype name (in upper case).
%PADDR	**Used with a Prototype. Same format** Example C Eval procptr = %paddr(TheProc) The argument of %PADDR can be a prototype name, with

	restrictions: (1) It must not be a prototype for a Java method. (2) It must not have the EXTPGM keyword. (3) If its EXTPROC keyword has a procedure pointer for an argument, %PADDR |cannot be used in definition specifications. |
%PARMS	**# of parameters passed to procedure. Format:** %PARMS – returns the # of parms Example: **IF %PARMS < 1** %PARMS returns the number of parameters that were passed to the procedure in which %PARMS is used. For the main procedure, %PARMS is the same as *PARMS. The value returned by %PARMS is not available if the program or procedure that calls %PARMS does not pass a minimal operational descriptor. The ILE RPG compiler always passes one, but other languages do not. So if the caller is written in another ILE language, it will need to pass an operational descriptor on the call. If the operational descriptor is not passed, the value returned by %PARMS cannot be trusted.
%REALLOC	**Numeric pointer: to allocated storage**. **Format:** %REALLOC(ptr:num) Example: **RESpointer = %REALLOC(pointer:500);** %REALLOC changes the heap storage pointed to by the first parameter to be the length specified in the second parameter. The newly allocated storage is uninitialized. The first parm must be a basing pointer value. The second parm must be a non-float numeric value with zero decimal places. The length specified must be between 1 and 16776704. The function returns a pointer to the allocated storage. This may be the same as ptr or different.
%REM	**Division - remainder from div of 2 arguments: Format:** %REM(n:m) Example: Result = %REM(A:B); %REM returns the remainder that results from dividing operands n by m. The two operands must be numeric values with zero decimal positions
%REPLACE	**Replacement string. Format** %REPLACE(replacement string: source string{:start position {:source length to replace}}) Example: **Result = %replace ('Scranton': result);** %REPLACE returns the character string produced by inserting a replacement string into the source string, starting at the start position and replacing the specified number of characters. The first and second parm must be of type character, graphic, or UCS-2 and can be in either fixed- or variable-length format. The second parm must be the same

	type as the first. If the third parm is not specified, the starting position is at the beginning of the source string. The value may range from one to the current length of the source string plus one. The fourth parm represents the number of characters in the source string to be replaced. If zero is specified, then the replacement string is inserted before the specified starting position. If the parm is not specified, the number of characters replaced is the same as the length of the replacement string.
%SCAN	**Returns searched for value or zero. Forat:** %SCAN(search Argument : source string {: start}) Example: Position = %scan ('D' : source : 2); %SCAN returns the first position of the search argument in the source string, 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 type of the return value is unsigned integer. This BIF can be used anywhere that an unsigned integer expression is valid. Unlike the SCAN operation code, %SCAN cannot return an array containing all occurrences of the search string and its results cannot be tested using the %FOUND built-in function.
%SECONDS	# of seconds as a duration. Format: %SECONDS(number) Example: **// Determine the time in 36 seconds** **Newtime = time + %SECONDS(36);** %SECONDS converts a number into a duration that can be added to a time or timestamp value. %SECONDS can only be the right-hand value in an addition or subtraction operation. The left-hand value must be a time or timestamp. The result is a time or timestamp value with the appropriate number of seconds added or subtracted. For a time, the resulting value is in *ISO format.
%SHTDN	**Returns value indicating shutdown (1 or 0) Format:** %SHTDN Example: **IF %SHTDN;** **QuitProgram();** **ENDIF;** %SHTDN returns '1' if the system operator has requested shutdown; otherwise, it returns '0'.
%SIZE	**Returns size of variable or literal. Formats:** %SIZE(variable) %SIZE(literal)

%SIZE(array{:*ALL})
%SIZE(table{:*ALL})
%SIZE(multiple occurrence data structure{:*ALL})
Example: **Result = %SIZE(field1);**
%SIZE returns the number of bytes occupied by the constant or field. If the argument is an array name, table name, or multiple occurrence data structure name, the value returned is the size of one element or occurrence. If *ALL is specified as the second parameter for %SIZE, the value returned is the storage taken up by all elements or occurrences. Returns size of variable or literal

%SQRT **Square root of a numeric value. Format:**
%SQRT(numeric expression)
Example: **Result = %SQRT(239874);**
%SQRT returns the square root of the specified numeric expression.

%STATUS **0 if no I/O error for file. Format**
%STATUS{(file_name)}
Example: **When %status = 01331;**
Exsr SUBOVER999
%STATUS returns the most recent value set for the program or file status. %STATUS is set whenever the program status or any file status changes, usually when an error occurs. If %STATUS is used without the optional file_name parameter, then it returns the program or file status most recently changed. If a file is specified, the value contained in the INFDS *STATUS field for the specified file is returned. The INFDS does not have to be specified for the file. %STATUS starts with a return value of 00000 and is reset to 00000 before any operation with an 'E' extender specified begins. %STATUS is best checked immediately after an operation with the 'E' extender or an error indicator specified, or at the beginning of an INFSR or the *PSSR subroutine.

%STR **String characters addressed by pointer argument. Format:**
%STR (Get or Store Null-Terminated String)
1. Get: %STR(basing pointer{: max-length})(right-hand-side)
2. Store: %STR(basing pointer : max-length)(left-hand-side)
Example 1 Get: **ResultGet = '<' + %str(String1 : 2) + '>';**
Example 2 Store **%str(StoreStr(25))= 'abcdef';**
%STR is used to create or use null-terminated character strings, which are very commonly used in C and C++ applications. The first parameter must be a basing-pointer variable. The second parameter, if specified, must be a numeric value with zero decimal positions. If not specified, it defaults

	to 65535. The first parameter must point to storage that is at least as long as the length given by the second parameter When used on the right-hand side of an expression, this function returns the data pointed to by the first parameter up to but not including the first null character (x'00') found within the length specified. When used on the left-hand side of an expression, %STR(ptr:length) assigns the value of the right-hand side of the expression to the storage pointed at by the pointer, adding a null-terminating byte at the end. The maximum length that can be specified is 65535. This means that at most 65534 bytes of the right-hand side can be used, since 1 byte must be reserved for the null-terminator at the end. The length indicates the amount of storage that the pointer points to. This length should be greater than the maximum length the right-hand side will have. The pointer must be set to point to storage at least as long as the length parameter. If the length of the right-hand side of the expression is longer than the specified length, the right-hand side value is truncated. Make sure that the length parameter is not greater than the actual length of data addressed by the pointer and that the length of the right-hand side is not greater than or equal to the actual length of data addressed by the pointer. You must keep track of the length that you have allocated.
%SUBARR	**Return a subset of an array. Format:** %SUBARR(array:start-index{:number-of-elements}) Example: **ResultArr = %subarr(a:4:n);** %SUBARR returns a section of the specified array starting at start-index. The number of elements returned is specified by the optional number-of-elements parameter. If not specified, the number-of-elements defaults to the remainder of the array. The first parameter of %SUBARR must be an array. That is, a standalone field, data structure, or subfield defined as an array. The first parameter must not be a table name or procedure call. The start-index parameter must be a numeric value with zero decimal positions. A float numeric value is not allowed. The value must be greater than or equal to 1 and less than or equal to the number #of elements of the array. The optional number-of-elements parameter must be a numeric value with zero decimal positions. A float numeric value is not allowed. The value must be greater than or equal to 1 and less than or equal to the number of elements remaining in the array after applying the start-index value. Generally, %SUBARR is valid in any expression where an unindexed array is allowed.
%SUBDT	**Returns a portion of date or time value.**

	Format 1 %SUBDT(value:*MSECONDS|*SECONDS |*MINUTES|*HOURS|*DAYS|*MONTHS|*YEARS) Format 2 %SUBDT(value:*MS|*S|*MN|*H|*D|*M|*Y) Example: **Numval = %subdt(date:*YEARS);** %SUBDT extracts a portion of the information in a date, time, or timestamp value. It returns an unsigned numeric value. The first parameter is the date, time, or timestamp value. The second parameter is the portion that you want to extract. The following values are valid: For a date: *DAYS, *MONTHS, and *YEARS. For a time: *SECONDS, *MINUTES, and *HOURS. For a timestamp: *MSECONDS, *SECONDS, *MINUTES, *HOURS, *DAYS, *MONTHS, and *YEARS. For this function, *DAYS always refers to the day of the month not the day of the year (even if you are using a Julian date format).
%SUBST	**Returns a substring. Two Formats** **%SUBST Form1 : Used for its value** Format: %SUBST(string: start postion, length) Example: **C IF %SUBST(CITY:C+1) = 'Scranton'** **%SUBST Form 2 : Used as the Result of an Assignment** Format: %SUBST(string: start postion, length) Example: **C EVAL %SUBST(A:3:4) = '****'** **%SUBST used for its value** (Form 1) returns a substring from the contents of the specified string. The substring begins at the specified starting position in the string and continues for the length specified. If length is not specified then the substring continues to the end of the string. **%SUBST used as the result of an assignment** (Form 2) refers to certain positions of the argument string. The result begins at the specified starting position in the variable and continues for the length specified. If the length is not specified then the string is referenced to its end. If the length refers to characters beyond the end of the string, then a run-time error is issued. When %SUBST is used as the result of an assignment, the first parameter must refer to a storage location such as a field or structure. Any valid expressions are permitted for the second and third parameters of %SUBST when it appears as the result of an assignment with an EVAL operation.
%THIS	**The class instance for the native method.** Format %THIS -- returns object valuereference to class Example: **C Eval Id_Num = getId(%THIS)**

	%THIS returns an Object value that contains a reference to the class instance on whose behalf the native method is being called. %THIS is valid only in non-static native methods. This BIF gives non-static native methods access to the class instance. A non-static native method works on a specific instance of its class. This object is actually passed as a parameter to the native method by Java, but it does not appear in the prototype or procedure interface for the native method In a Java method, the object instance is referred to by the Java reserved word this. In an RPG native method, the object instance is referred to by the %THIS BIF.
%TIME	**Brings back system time if none is specified. Format:** %TIME{(expression{:time-format})} Example: **Time = %time(string:*USA);** %TIME converts the value of the expression from character, numeric, or timestamp data to type time. The converted value remains unchanged, but is returned as a time. The first parameter is the value to be converted. If you do not specify a value, %TIME returns the current system time. The second parameter is the time format for numeric or character input. Regardless of the input format, the output is returned in *ISO format.
%TIMESTAMP	**Brings back current timestamp if none specified.** Format: %TIMESTAMP{(expression{:*ISO|*ISO0})} Example: Times = %timestamp(string); %TIMESTAMP converts the value of the expression from character, numeric, or |date data to type timestamp. The converted value is returned as a timestamp. The first parameter is the value to be converted. If you do not specify a value, %TIMESTAMP returns the current system timestamp. The second parameter is the timestamp format for character input. Regardless of the input format, the output is returned in *ISO format.
%TLOOKUPxx	**Checks for match and returns '*ON' or '*OFF.'** The "xx" is the specific type of match Formats: %TLOOKUP(arg : search-table {: alt-table}) – Exact match %TLOOKUPLT(arg : search-table {: alt-table}) – Closest but less than %TLOOKUPGE(arg : search-table {: alt-table}) – Exact match or closest but less than %TLOOKUPGT(arg : search-table {: alt-table}) – Closest but greater than

	%TLOOKUPLE(arg : search-table {: alt-table}) – Exact or closest but greater than Example: **IF %TLOOKUP('Goose Bay':tab1:tab2);** If a value meets the specified condition, the current table element for the search table is set to the element that satisfies the condition, the current table element for the alternate table is set to the same element, and the function returns the value *ON. If no value matches the specified condition, *OFF is returned. Unlike the %LOOKUP operation code, %TLOOKUP applies only to tables. To look up a value in an array, use the %LOOKUP BIF.
%TRIM string	**Trims string with left, right blanks or specified. Format:** %TRIM(string) Example: **Name = %trim (FirstName) + ' ' + %trim (LastName);** Returns string less any leading and trailing blanks.
%TRIML string	**Trims string with left blanks or specified. Format** %TRIML(string) Example: **Location = %triml(' Wilkes-Barre, PA'));** %TRIML returns the given string less any leading blanks When specified as a parameter for a definition specification keyword, the string parameter must be a constant.
%TRIMR string	**Trims string with right blanks or specified** Format: %TRIMR(string) Example: **Location = %trim (' Wilkes-Barre, PA ');** %TRIMR returns the given string less any trailing blanks. When specified as a parameter for a definition specification keyword, the string parameter must be a constant.
%UCS2	**Brings back value in UCS-2 format. Format:** %UCS2 --converts to UCS2 type of value Example: **C eval ufield = %UCS2(char) + %UCS2(graph)** %UCS2 converts the value of the expression from character, graphic, or UCS-2 and returns a UCS-2 value. The second parameter, ccsid, is optional and indicates the CCSID of the resulting expression. The CCSID defaults to 13488. If the parameter is a constant, the conversion will be done at compile time.
%UNS	**Brings back value in unsigned format. Format:** %UNS(numeric expression) Example **Result = %uns (NumValue);** %UNS converts the value of the numeric expression to unsigned format. Any decimal digits are truncated. %UNS can be used to truncate the decimal positions from a float or decimal value allowing it to be used as an array index.

%UNSH	**Brings back rounded value - unsigned format. Format:** %UNSH(numeric expression) Example **Result = %unsh (FloatValue);** %UNSH is like %UNS except that if the numeric expression is a decimal or a float value, half adjust is applied to the value of the numeric expression when converting to unsigned type.
%XFOOT	**Array expression sum of the elements. Format:** %XFOOT(array-expression) Example: Result = %xfoot(MothSales) %XFOOT results in the sum of all elements of the specified numeric array expression. The precision of the result is the minimum that can hold the result of adding together all array elements, up to a maximum of 30 digits. The number of decimal places in the result is always the same as the decimal places of the array expression. This built-in function is similar to the XFOOT operation, except that float arrays are summed like all other types, beginning from index 1 on up.
%XLATE	**Translate String (eg Lower Case to Upper Case). Format:** %XLATE(from:to:string{:startpos}) Example: **String = %XLATE(locase:upcase:'our dept');** **// string now contains 'OUR DEPT'** %XLATE translates string according to the values of from, to, and startpos. The first parameter contains a list of characters that should be replaced, and 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. 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. The first three parameters can be of type character, graphic, or UCS-2. All three must have the same type. The value returned has the same type and length as string. The fourth parameter is a non-float numeric with zero decimal positions.
%YEARS	**# of years as a duration Format: Format:** %YEARS(number) Example: **FutureDat = Today + %YEARS(4);** *add 4yr to date.* %YEARS converts a number into a duration that can be added to a date or timestamp value. %YEARS can only be the right-hand value in an addition or subtraction operation. The left-hand value must be a date or timestamp. The result is a date or timestamp value with the appropriate number of years added or subtracted. For a date, the resulting value is in *ISO format. If the left-hand value is February 29 and the resulting year is

not a leap year, February 28 is used instead. Adding or subtracting a number of years to a February 29 date may not be reversible. For example, |20014-02-29 + %YEARS(1) – %YEARS(1) is 20014-02-28.

Table 14-11 BIF Arguments and Values

Name	Arguments	Value Returned
%ABS	numeric expression	absolute value of expression
%ADDR	variable name	address of variable
%ALLOC	number of bytes to allocate	pointer to allocated storage
%BITAND	character, numeric	bit wise ANDing of the bits of all the arguments
%BITNOT	character, numeric	bit-wise reverse of the bits of the argument
%BITOR	character, numeric	bit-wise ORing of the bits of all the arguments
%BITXOR	character, numeric	bit-wise exclusive ORing of the bits of the two arguments
%CHAR	graphic, UCS-2, numeric, date, time, or timestamp expression {: date, time, or timestamp format}	value in character format
%CHECK	comparator string:string to be checked{:start position}	first position of a character that is not in the comparator string, or zero if not found
%CHECKR	comparator string:string to be checked{:start position}	last position of a character that is not in the comparator string, or zero if not found
%DATE	{value {: date format}}	the date that corresponds to the specified *value*, or the current system date if none is specified
%DAYS	number of days	number of days as a duration
%DEC	numeric expression {:digits:decpos} character expression: digits:decpos	value in packed numeric format

<table>
<tr><td></td><td>date, time or timestamp expression {:format}</td><td></td></tr>
<tr><td>%DECH</td><td>numeric or character expression:
digits:decpos</td><td>half-adjusted value in packed numeric format</td></tr>
<tr><td>%DECPOS</td><td>numeric expression</td><td>number of decimal digits</td></tr>
<tr><td>%DIFF</td><td>Date or time expression: date or time expression: unit</td><td>difference between the two dates, times, or timestamps in the specified unit</td></tr>
<tr><td>%DIV</td><td>dividend: divisor</td><td>the quotient from the division of the two arguments</td></tr>
<tr><td>%EDITC</td><td>non-float numeric expression:edit code {:*CURSYM | *ASTFILL | currency symbol}</td><td>string representing edited value</td></tr>
<tr><td>%EDITFLT</td><td>numeric expression</td><td>character external display representation of float</td></tr>
<tr><td>%EDITW</td><td>non-float numeric expression:edit word</td><td>string representing edited value</td></tr>
<tr><td>%ELEM</td><td>array, table, or multiple occurrence data structure name</td><td>number of elements or occurrences</td></tr>
<tr><td rowspan="2">%EOF</td><td rowspan="2">{file name}</td><td>'1' if the most recent cycle input, read operation, or write to a subfile (for a particular file, if specified) ended in an end-of-file or beginning-of-file condition; and, when a file is specified, if a more recent OPEN, CHAIN, SETGT or SETLL to the file was not successful</td></tr>
<tr><td>'0' otherwise</td></tr>
<tr><td rowspan="2">%EQUAL</td><td rowspan="2">{file name}</td><td>'1' if the most recent SETLL (for a particular file, if specified) or LOOKUP operation found an exact match</td></tr>
<tr><td>'0' otherwise</td></tr>
<tr><td rowspan="2">%ERROR</td><td rowspan="2"></td><td>'1' if the most recent operation code with extender 'E' specified resulted in an error</td></tr>
<tr><td>'0' otherwise</td></tr>
</table>

%FIELDS	list of fields to be updated	not applicable
%FLOAT	numeric or character expression	value in float format
%FOUND	{file name}	'1' if the most recent relevant operation (for a particular file, if specified) found a record (CHAIN, DELETE, SETGT, SETLL), an element (LOOKUP), or a match (CHECK, CHECKR, SCAN)
		'0' otherwise
%GRAPH	character, graphic, or UCS-2 expression	value in graphic format
%HANDLER	handling procedure : communication area	not applicable
%HOURS	number of hours	number of hours as a duration
%INT	numeric or character expression	value in integer format
%INTH	numeric or character expression	half-adjusted value in integer format
%KDS	data structure containing keys {: number of keys}	not applicable
%LEN	any expression	length in digits or characters
%LOOKUPxx	argument: array{:start index {:number of elements}}	array index of the matching element
%MINUTES	number of minutes	number of minutes as a duration
%MONTHS	number of months	number of months as a duration
%MSECONDS	number of microseconds	number of microseconds as a duration
%NULLIND	Null-capable field name	value in indicator format representing the null indicator setting for the null-capable field
%OCCUR	multiple-occurrence data structure name	current occurrence of the multiple-occurrence data structure
%OPEN	file name	'1' if the specified file is open
		'0' if the specified file is closed
%PADDR	procedure or prototype name	address of procedure or prototype
%PARMS	none	number of parameters passed to

		procedure
%REALLOC	pointer: numeric expression	pointer to allocated storage
%REM	dividend: divisor	the remainder from the division of the two arguments
%REPLACE	replacement string: source string {:start position {:source length to replace}}	string produced by inserting replacement string into source string, starting at start position and replacing the specified number of characters
%SCAN	search argument:string to be searched{:start position}	first position of search argument in string or zero if not found
%SECONDS	number of seconds	number of seconds as a duration
%SHTDN		'1' if the system operator has requested shutdown
		'0' otherwise
%SIZE	variable, array, or literal {:*ALL}	size of variable or literal
%SQRT	numeric value	square root of the numeric value
%STATUS	{file name}	0 if no program or file error occurred since the most recent operation code with extender 'E' specified
		most recent value set for any program or file status, if an error occurred
		if a file is specified, the value returned is the most recent status for that file
%STR	pointer{:maximum length}	characters addressed by pointer argument up to but not including the first x'00'
%SUBARR	array name:start index{:number of elements}	array subset
%SUBDT	Date or time expression: unit	an unsigned numeric value that contains the specified portion of the date or time value
%SUBST	string:start{:length}	substring
%THIS		the class instance for the native method
%TIME	{value {: time format}}	the time that corresponds to the

		specified *value*, or the current system time if none is specified
%TIMESTAMP	{(value {: timestamp format})}	the timestamp that corresponds to the specified *value*, or the current system timestamp if none is specified
%TLOOKUPxx	argument: search table {: alternate table}	'*ON' if there is a match
		'*OFF' otherwise
%TRIM	string {: characters to trim}	string with left and right blanks or specified characters trimmed
%TRIML	string {: characters to trim}	string with left blanks or specified characters trimmed
%TRIMR	string {: characters to trim}	string with right blanks or specified characters trimmed
%UCS2	character, graphic, or UCS-2 expression	value in UCS-2 format
%UNS	numeric or character expression	value in unsigned format
%UNSH	numeric or character expression	half-adjusted value in unsigned format
%XFOOT	array expression	sum of the elements
%XLATE	from-characters: to-characters: string {: start position}	the string with from-characters replaced by to-characters
%XML	Xml document { : options }	not applicable
%YEARS	number of years	number of years as a duration

Chapter 8

RPG/400 & RPGIV Structured Programming Operations

Extension Specifications

Extension specifications describe all record address files, table files, and array files
used in the program. The information includes:

- Name of the file, table, or array
- Number of entries in a table or array input record
- Number of entries in a table or array
- Length of the table or array entry.

Line Counter Specifications

Line counter specifications describe the page or form on which output is printed.
The information includes:

- Number of lines per page
- Line of the page where overflow occurs.

Input Specifications

Input specifications describe the records, fields, data structures and named constants
used by the program. The information in the input specifications includes:

- Name of the file
- Sequence of record types
- Whether record-identifying indicators, control-level indicators, field-recordrelation
indicators, or field indicators are used
- Whether data structures, lookahead fields, record identification codes, or match
fields are used
- Type of each field (alphanumeric or numeric; packed-decimal, zoned-decimal,
or binary format)
- Location of each field in the record
- Name of each field in the record
- Named constants.

Calculation Specifications

Calculation specifications describe the calculations to be done on the data and the
order of the calculations. Calculation specifications can also be used to control
certain input and output operations. The information includes:

- Control-level and conditioning indicators for the operation specified
- Fields or constants to be used in the operation

[1] The operation to be processed
[1] Whether resulting indicators are set after the operation is processed.

```
        THE DOWALLOBY COMPANY GROSS PAY REGISTER BY STATE          6/09/06

ST     CITY          EMP#    EMPLOYEE NAME        RATE     HOURS      CHECK


PA     WILKES-BARRE  001     BIZZ NIZWONGER       7.80     35.00     273.00
PA     WILKES-BARRE  002     WARBLER JACOBY       7.90     40.00     316.00


                             TOTAL CITY  PAY FOR WILKES-BARRE        589.00

PA     SCRANTON      003     BING CROSSLEY        8.55     65.00     555.75


                             TOTAL CITY  PAY FOR SCRANTON            555.75

                             TOTAL STATE PAY FOR PA                1,144.75

AK     FAIRBANKS     004     UPTAKE N. HIBITER    7.80     25.00     195.00
AK     FAIRBANKS     005     FENWORTH GRONT       9.30     33.00     306.90
AK     FAIRBANKS     007     BI NOMIAL            8.80     39.00     343.20


                             TOTAL CITY  PAY FOR FAIRBANKS           845.10

AK     JUNEAU        008     MILLY DEWITH         6.50     40.00     260.00
AK     JUNEAU        009     SARAH BAYOU         10.45     40.00     418.00


                             TOTAL CITY  PAY FOR JUNEAU              678.00

                             TOTAL STATE PAY FOR AK                1,523.10

NJ     NEWARK        010     DIRT MCPUG           6.45     35.50     228.97

NJ     NEWARK        011     No Time Card this pay period for below:
NJ     NEWARK        011     BANDAID JONES        4.50 ** No Time Card **


                             TOTAL CITY  PAY FOR NEWARK              228.97

                             TOTAL STATE PAY FOR NJ                  228.97


                                FINAL TOTAL PAY                    2,896.82
```

```
                    Create Data Area (CRTDTAARA)

 Type choices, press Enter.

 Data area . . . . . . . . . . . > PAYPERIOD     Name
   Library . . . . . . . . . . .     *CURLIB     Name, *CURLIB
 Type . . . . . . . . . . . . . . > *CHAR        *DEC, *CHAR, *LGL, *DDM
 Length:
   Length . . . . . . . . . . . . > 5            1-2000
   Decimal positions . . . . . .                 0-9
 Initial value . . . . . . . . . > '12021'
 Text 'description' . . . . . . . > 'PAYPERIOD & NEXT EMPLOYEE #'

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

```
                    Display Spooled File
File . . . . . :   QPRINT2                  Page/Line   1/6
Control . . . . .                           Columns     1 - 78
Find . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
        THE DOWALLOBY COMPANY Select PAY History List
6/15/06
STATE NAME    CITY         EMP# EMPLOYEE NAME   DEP
NAME  Y GROSS   >HRS SAL?
```

PENNSYLVANIA WILKES-BARRE 001 BIZZ NIZWONGER MILLING 273.0 35.00 N
PENNSYLVANIA WILKES-BARRE 002 WARBLER JACOBY SANDING 316.0 40.00 N
PENNSYLVANIA SCRANTON 003 BING CROSSLEY SANDING 442.3 .00 Y
ALASKA FAIRBANKS 004 UPTAKE N. HIBIT SANDING 557.6 .00 Y
ALASKA FAIRBANKS 005 FENWORTH GRONT MILLING 306.9 33.00 N
ALASKA FAIRBANKS 007 BI NOMIAL MILLING 343.2 39.00 N
ALASKA JUNEAU 008 MILLY DEWITH GRINDING 260.0 40.00 N
ALASKA JUNEAU 009 SARAH BAYOU GRINDING 418.0 40.00 N
NEW JERSEY NEWARK 010 DIRT MCPUG SANDING 228.9 35.50 N
NEW JERSEY NEWARK 011 BANDAID JONES MILLING .0 .00 N
NEW JERSEY NEWARK 021 ROBINSON CRUSOE MILLING .0 .00 N
NEW JERSEY NEWARK 022 SERMELLA NIPTUC SANDING .0 .00 N
NEW JERSEY NEWARK 023 SIMON TEMPLAR MILLING .0 .00 N
NEW JERSEY NEWARK 024 BURGER BALL MILLING .0 .00 N

More...

Chapter 15

RPG Operations in Action

The Once and Future PAREG2

PAREG2 as shown in Figure 15-1 is the reincarnation of PAREG without the use of the RPG cycle. In all fairness, the program is not really 248 statements because in order to demonstrate the new routines more clearly we used more commenting within the program. In fact, there are 61 additional comment lines. over and above those in the program described version of PAREG.

In Figure 5-2, we introduced the program described version of PAREG with 69 statements, 13 of which were comments. Thus, there were 56 operative statements in this PAREG. So, if we take the 61 additional plus the 13 original comments from PAREG2 (74 comment statements in total) from the 248 statements in this program, there are and we le 174 operative statements in PAREG2 gram.

Yes, we did add a little bit of new function. For example, we added two additional fields to the Employee Master, – a department # (EMPDPT) and a Salary code. We also added a new file. Rather than extend the Payroll master further, we borrowed a technique from the days of old when disk space was expensive. Since these programs exist today, we placed the new salary payroll option in its own file. Thus, to calculate pay with a salary employee, we must access this file using a random read (CHAIN). Overall, it took eleven statements necessary for processing the salary file and the department # in PAREG2 after they were added to the mix. That brings us to 163 operative statements to perform the same function as PAREG in Figure 5-2.

Figure 15-1 PAREG2 Program – Register with No Cycle, MR, Levels

```
     *PAREG2P internally described PAREG - no MR, No LX totals
001  H* RPG HEADER (CONTROL) SPECIFICATION FORMS
002  H
003  F*
004  F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005  F*
006  FEMPMAST IF  F      70  3AI      1 DISK
007  FTIMCRD  IF  F       7  3AI      1 DISK
008  FSALFILE IF  F       9  3AI      1 DISK
009  FQPRINT  O   F      77     OF      PRINTER
010  FERROR   O   E                     PRINTER
011  I*
012  I* RPG INPUT SPECIFICATION FORMS
013  I*
014  I*
015  I*  EMPMAST is the employee master file
016  I*  One record per employee - pay rate and dept
017  I*  For salaried employees, Salary is in SALFILE
018  I*
019  IEMPMAST AA  01
020  I                                         1  70 EREC
021  I                                         1   30EMPNO
022  I                                         4  23 EMPNM
023  I                                         4  33 EMPNAM
024  I                                        34  382EMPRAT
025  I                                        39  58 EMPCTY
026  I                                        59  60 EMPSTA
027  I                                        61  650EMPZIP
028  I                                        66  66 EMPSCD-SALCOD
029  I                                        67  70 EMPDPT-DPTCOD
030  I*
031  I*  TIMCRD is updated in an independent process.
032  I*  Provides current time records for the PAYROLL process
033  I*  For salaried employees, no hours provided but TIMcard
034  I*  is needed for person to be paid.
035  I*
036  ITIMCRD  AB  02
037  I                                         1   30EMPNO
038  I                                         4   72EMPHRS
039  I*
040  I*  SALCRD is updated in an independent process.
041  I*  Mimics an extension to the PAYMAST file
042  I*  For salaried employees, Salary stored in this file
043  I*
044  ISALFILE AC  03
045  I                                         1   30SALENO
046  I                                         4   90SALYR
047  I*  HLDMAST is the working file for level chacking
048  I*
049  IHLDMST      DS
050  I                                         1   30HLDNO
051  I                                         1  70 HREC
052  I                                         4  33 HLDNAM
053  I                                         4  23 HLDNM
```

```
054  I                                        34  382HLDRAT
055  I                                        39  58 HLDCTY
056  I                                        59  60 HLDSTA
057  I                                        61  650HLDZIP
058  I                                        66  66 HLDSCD
059  I                                        67  70 HLDDPT
060  I*
061  C*
062  C* RPG CALCULATION SPECIFICATION FORMS
063  C*
064  C*  Run default register with no prompt input
065  C                   EXSR RUNREG
066  C                   SETON                    LR
067  C*
068  C*  Body of Code- Controls running of Payroll Register
069  C          RUNREG    BEGSR
070  C*  Check to see if there is a missing master
071  C                   EXSR CHKMST
072  C*  Clear fields from CHKMST run to begin fresh register
073  C                   EXSR CLR
074  C                   CLOSEEMPMAST
075  C                   OPEN EMPMAST
076  C*  First read ahead to be able to check for Levels
077  C                   READ EMPMAST                 91
078  C                   EXCPTHEADER
079  C                   EXSR PROCES
080  C*  Run register until end of file
089  C          *IN91     DOUEQ*ON
090  C*  SECOND READ -- UNTIL EOF NEED FOR LEVEL CHECK
091  C                   READ EMPMAST                 91
092  C   91              LEAVE
093  C*  Replaces L1 coding as in PAREG
094  C*  LEVEL 1 TEST -- See if current city is different
095  C          EMPCTY    IFNE HLDCTY
096  C          EMPSTA    ORNE HLDSTA
097  C                   SETON                    L1
098  C                   EXSR LEVEL1
099  C                   ENDIF
100  C*  Level 2 test -- See if state changed
101  C          EMPSTA    IFNE HLDSTA
102  C                   SETON                    L2
103  C                   EXSR LEVEL2
104  C                   ENDIF
105  C                   SETOF                    L1L2
106  C                   EXSR PROCES
107  C                   ENDDO
108  C                   EXSR LEVEL1                   L1Break
109  C                   EXSR LEVEL2                   L2Break
110  C                   EXCPTLROUT                    LRBreak
111  C                   ENDSR
112  C*
113  C*  Level 1 Subroutine - Control break on City
114  C*
115  C          LEVEL1    BEGSR
116  C          CTYPAY    ADD  STAPAY    STAPAY  92
```

```
117  C                     EXCPTL1OUT
118  C                     ENDSR
119  C*
120  C*  Level 2 Subroutine - Control break on State
121  C*
122  C           LEVEL2    BEGSR
123  C           STAPAY    ADD  TOTPAY    TOTPAY  92
124  C                     EXCPTL2OUT
125  C                     ENDSR
126  C*
127  C*  PAYCLC  Calculates Gross PAY from HRS or Salary
128  C*  Also calculates "net pay" and updates YTD files.
129  C*  Calculate pay for HELD record
130  C*  If Salaried, do not use RATE multiplier
131  C*
132  C           PAYCLC    BEGSR
133  C                     SETOF                     9298
134  C                     Z-ADD0         HLDSAL  60
135  C* REPLACES MR CYCLE WORK
136  C           HLDNO     CHAINTIMCRD               92    No TC
137  C   92                EXCPTNOTIME                     ERROR
138  C  N92      HLDRAT    MULT EMPHRS    HLDPAY  72      CALCPAY
139  C  N92                Z-ADDEMPHRS    HLDHRS  92
140  C  N92      HLDPAY    ADD  CTYPAY    CTYPAY  92      ADDCity
141  C  N92      HLDNO     CHAINSALFILE              98   Get Sal
142  C  N98N92             Z-ADDSALYR     HLDSAL  60
143  C  N98N92   SALYR     DIV  52        HLDPAY          CalcSal
144  C  N98N92   HLDPAY    ADD  CTYPAY    CTYPAY  92      City
145  C  N98N92             Z-ADD0         HLDHRS          No HRS
146  C                     ENDSR
147  C*
148  C* Write error msg for no master to separate Ext print file
149  C*
150  C           NOMAST    BEGSR
151  C                     WRITEHDR
152  C                     WRITEDTL
153  C                     ENDSR
154  C*
155  C*  Process line item
156  C*
157  C           PROCES    BEGSR
158  C                     EXSR MOVMS1
159  C                     EXSR PAYCLC
160  C                     EXSR PRNTLN
161  C                     ENDSR
162  C*
163  C* Print Detail Line on Register
164  C*
165  C           PRNTLN    BEGSR
166  C   OF                EXCPTHEADER
167  C   OF                SETOF                     OF
168  C                     EXCPTPRTLN1
169  C                     ENDSR
170  C*
171  C* Move Fields to Hold Area- Level Info/ Comparison
```

```
172  C*
173  C           MOVMS1    BEGSR
174  C                     Z-ADDEMPNO     HLDNO
175  C                     MOVELEMPNAM    HLDNAM
176  C                     Z-ADDEMPRAT    HLDRAT
177  C                     MOVELEMPCTY    HLDCTY
178  C                     MOVELEMPSTA    HLDSTA
179  C                     Z-ADDEMPZIP    HLDZIP
180  C                     MOVELEMPSCD    HLDSCD
181  C                     MOVELEMPDPT    HLDDPT
182  C                     ENDSR
183  C*
184  C* CLR Clear fields used in the missing master test
185  C*
186  C           CLR       BEGSR
187  C                     Z-ADD0         EMPNO
188  C                     MOVE *BLANKS   EMPNAM
189  C                     Z-ADD0         EMPRAT
190  C                     MOVE *BLANKS   EMPCTY
191  C                     MOVE *BLANKS   EMPSTA
192  C                     Z-ADD0         EMPZIP
193  C                     MOVE *BLANKS   EMPSCD
194  C                     MOVE *BLANKS   EMPDPT
195  C                     Z-ADD0         EMPHRS
196  C                     ENDSR
197  C*
198  C*  CHKMST Read time cards for missing masters & report
199  C*
200  C           CHKMST    BEGSR
201  C           *IN93     DOUEQ*ON
202  C                     READ TIMCRD                 93
203  C           EMPNO     CHAINEMPMAST            94
204  C   94                EXSR NOMAST
205  C                     ENDDO
206  C                     ENDSR
207  O*
208  O* RPG OUTPUT SPECIFICATION FORMS
209  O*
210  OQPRINT  E  206           HEADER
211  O                                  32 'THE DOWALLOBY COMPA'
212  O                                  55 'GROSS PAY REGISTER '
213  O                                  60 'STATE'
214  O                         UDATE Y   77
215  O        E  3             HEADER
216  O                                   4 'ST'
217  O                                  13 'CITY'
218  O                                  27 'EMP#'
219  O                                  45 'EMPLOYEE NAME'
220  O                                  57 'RATE'
221  O                                  67 'HOURS'
222  O                                  77 'CHECK'
223  O        E 11             NOTIME
224  O                         HLDSTA     4
235  O                         HLDCTY    29
236  O                         HLDNO     27
```

```
237  O                                    53 'No Time Card th pay'
238  O                                    71 ' period for below:'
239  O         E 01             PRTLN1
230  O                          HLDSTA      4
231  O                          HLDCTY     29
232  O                          HLDNO      27
233  O                          HLDNM      52
234  O                 N92      HLDPAY1B   77
235  O                 N92      HLDHRS1B   67
236  O                          HLDRAT1    57
237  O                  92                 76 '** No Time Card **'
238  O         E 22             L1OUT
239  O                                    51 'TOTAL CITY  PAY FOR'
240  O                          HLDCTY     72
241  O                          CTYPAY1B   77
242  O         E 02             L2OUT
243  O                                    51 'TOTAL STATE PAY FOR'
244  O                          HLDSTA     54
245  O                          STAPAY1B   77
246  O         E 2              LROUT
247  O                          TOTPAY1    77
248  O                                    50 'FINAL TOTAL PAY'
```

So, what happened that it cost us almost three times the code in order to perform the same functions without the RPG cycle. The additional lines of code can be characterized as in Table 15-2.

Table 15-2 Reasons for Code Increases from PAREG to PAREG2

Reason For Code Increase	# Stmts
No RPG Cycle Input	**5**
No RPG Cycle Output	**13**
No Matching records logic – Includes missing master test	**26**
No Cycle Control Break testing / Processing	**44**
Miscellaneous DOs and IFs	**15**

As we decode the major areas of this program, it will become clear as to why it takes so much coding to replace the RPG cycle when the objective is to prepare a simple report. The net of it is, however, that the RPG cycle does lots of work for you that you never see. When there is no RPG cycle, you must do that work.

Looking at the PREG2 program in file description for example, you notice that there is no primary and no secondary files. Therefore, at input time, you must do your own reading. At output time, since there is no handy cycle to do the printing as you designate, you must take over the action using exception output (EXCPT op-code) during calculations to get the heading and detail and total lines printed.

Since there is no opportunity to place an A in column 18 of file description for ascending and / or a D for descending sequence, as you could easily do for a primary or secondary file, you must do sequence checking yourself to assure your input is in the proper order (by EMPNO in this program). Since we did not really ant to complicate this program further by introducing a sequence check routine, and since we rigged the input so that if the data were in EMPNO sequence, it would also be in City within State sequence, we took the easy way out on sequence checking. We used the AS/400 database facilities to define the new EMPMAST and the TIMCRD files, as well as the new SALFILE for salaries, as indexed files.

When you compare the file description of EMPMAST in Figure 5-3 with the primary file version in Figure 15-4, it is all but too obvious that there are major differences. By defining the key field information for EMPMAST in line 6 of Figure 15-3, we tell RPG that the key is 3 positions long, that there will be alphabetic keys, the file is indexed, and the key starts in position 1. By defining the key information in the RPG program that tells RPG to process the file by key.

So, what does that have to do with assuring ascending sequence by EMPNO? Notice that the EMPMAST, TIMCRD, and SALFILE all have an "F" designation in column 16. In RPG parlance, that makes the file use in the program fully procedural. Fully procedural means that operations supporting random reads, writes, and updates as well as operations supporting sequential reads, writes and updates are fully supported against this file

Figure 15-3 PAREG2 No Cycle File Description – Internally Described

```
004  F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005  F*
006  FEMPMAST IF  F      70  3AI     1 DISK
007  FTIMCRD  IF  F       7  3AI     1 DISK
```

```
008  FSALFILE IF  F       9  3AI     1 DISK
009  FQPRINT  O   F      77     OF     PRINTER
010  FERROR   O   E                    PRINTER
```

Figure 15-4 RPG Cycle Program PAREG – Internally Described Data

```
0004.00 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
0005.00 F*
0006.00 FEMPMAST IPEAF      55            DISK
0007.00 FTIMCRD  ISEAF       7            DISK
0008.00 FQPRINT  O   F      77     OF     PRINTER
```

In line 77 of the PAREG2 program, you can see the following coded operation:

077   C...   READ EMPMAST...   91

This operation reads the EMPMAST file using the RPG READ consecutive operation. Since the file is defined to be processed by key in File Descriptions, when a READ is issued against EMPMAST, the index, which is always maintained in sequence, just as the index in the back of a book, is read first under the covers. Then RPG uses the address of the record found in the index to bring the record into the program. For the next read, the next sequential index record is read and its data record is brought into the program. From RPG's perspective the data is being processed in city within state sequence and thus for PAREG2, it is always read in EMPNO sequence by key. The database in this program assures that the data will be presented in EMPNO sequence. Without having made the EMPAST file an indexed file, however, there would have been lots more work involved.

Is the Data All Wrong Here?

If the program report is sequenced city within state, then how is it that by sorting on EMPNO or using EMPNO as the key causes the data to be sequenced in city within state sequence? The answer is simple. That's

how the system is designed. Of course it is poor design. Try to add a record to the file and maintain this sequencing. Suppose you want to add a new employee in Scranton, for example. Let's say you want to add the next sequential employee #, employee 12, to Scranton and the data must be maintained in EMPNO sequence. What happens to the report sequencing?

Employee 12's Scranton PA record would be in the file logically and physically after Newark. Walking down the report in Figure 4-1, that would men that long after the totals for Scranton were taken in the report, another Scranton record would be read. This just does not work. This apparently intentional poor design did not matter in the PAREG cycle program since that environment was quite controlled and adding employees was not something that a neophyte programmer needed to be concerned about. But as this little report grows in to a bigger application, the structure of the data needs to be discussed and understood.

One way the designer could have dictated that the charade be kept going would be to increase the employee # by one or two columns. When the EMPDPT and EMPSCD fields were added for PAREG2, this would have been an ideal time. But the designer did not change the design. If he or she had, then it would permit employees to be inserted (added) into the proper state / city slot by assigning an appropriate employee number. For example, employees 00301 to 00399 would be Scranton slots. So, the next employee for Scranton could be employee # 00301.

You may recall that BING CROSSLEY is employee # 003. If the number were expanded by two digits, an additional 99 entries could be added to Scranton PA after CROSSLEY's number was changed to 00300. Another way would be to change the employee numbers within the three character constraint so that CORSSLEY was employee # 030 instead of 003. In this case, without a change to the file or the programs, up to 9 additional numbers could be assigned to Scranton, PA.

Data Design Matters

In data design, there are lots of ways to get the same objective accomplished. You may have noticed that the sorting within the City and States fields was not really in any sequence – other than to assure that all "Scranton" records for example were together, and all Wilkes-Barre

records were together. But, looking at the report first presented in Chapter 4 as Figure 4-2, Wilkes-Barre (W) city comes before Scranton (S) in the report and Pennsylvania (P) comes before Alaska (A). Being a Pennsylvania boy myself, you can appreciate why the sequencing is as it is.

Because the sequencing is hand – picked and not really in ascending or descending, there is no real straight forward way to maintain this sequence in a report other than to tie it to some other numbering mechanism as we did with EMPNO. We could have created a new sequencing file just for the report and used it as our own index to print lines in our desired sequence. This is a common technique and used often to assure that balance sheets and P& L statements can print properly regardless of the GL account # sequencing.

If, of course we did not want to maintain a hand-picked sequence in which Pennsylvania was the first state on the list, we could have sorted the data in city within state sequence thereby giving Fairbanks Alaska as the first entry in our report. If we chose to use an indexed approach, we could have created an alternate index on employee number within city within state. These three keys would give us the employees sequenced within city and the cities sequenced within states and the states themselves would be sequenced, giving us Alaska first. The coding for the logical file approach would be mostly the same with the READ operation delivering data from the index in employee within city within state sequence – no logic change in the program.

Overall, this would have been a better design but it is not the design that we find for this PAREG2 program or the original PAREG program. But, it does show that Level changes do not care if the next value is in sequence. Level changes do care however, whether the next value is different from the current value. And when different, either the RPG cycle, or you as the programmer as in PAREG2 have to account for the difference and fire off the appropriate level of totals on the printout.

Matching Records Processing

Whether you use the cycle as in PAREG or you use the manual method (PAREG2), matching records processing always depends on data being in

the sequence in which you wish to report. Thus, all of the considerations for sequencing which we just discussed apply.

In the PAREG2 example, there is no opportunity to place M1 next to fields that logically should be matched in a program. Therefore, there is no opportunity for the MR indication to be used to tell the program when there is a match. Likewise, there is no opportunity for the NMR indication to be used to tell the program when there is no match. Again we must do the work ourselves. And, again the fact that we changed the EMPMAST file from a consecutive data file to an indexed data file helps us in this task.

We also changed the file type of the TIMCRD file from consecutive (sequential) to indexed and this change enables us to greatly simplify the processing of the time card file. Additionally, the same logic works for the SALFILE so as you can see in File Descriptions (Figure 15-3) it too has been designed as an indexed file.

As an aside, if the SALFILE were in the PAREG RPG cycle program, it would have had to be coded as a tertiary file in column 16. Since there is no tertiary designator for column 16, it would have had to be coded as a secondary. To make it a tertiary file, it would be the third input file defined in File Descriptions using a P or S designation in column 16. In PAREG2, the same type of operation that brings in the matching TIMCRD record is used to bring in the SALFILE record. Other than the different record lengths, you can see that each file, EMPMAST, TIMCRD, and SALFILE use the same basic coding to assure they are processed as indexed files, in a fully procedural fashion in the PAREG2 program.

Since these three files are "sorted" in report processing sequence through the EMPNO key field of the respective files, we could have written the PAREG2 program to fully simulate the IBM MR logic. In other words, we could have processed all three files in an input sequential manner by using the index to deliver the records in sequence. To do this, we would have had to do all the coding and checking and "looking ahead" to see which record would be processed next. We would have to set on an equivalent of MR when we had matches and we would have had to set off MR when there were no matches.. Though this could be done, it would

have been lots more work than the key access processing method that we selected.

One advantage of processing all three files in a manual matching scenario would be that we would intrinsically know which records in each file were matched and which ones were unmatched. Line 77 of the PAREG program is where we start the action in this program by reading in the Master file sequentially. Both the Salary file and the Time Card files are processed randomly using the CHAIN operation.

Each employee, whether hourly or salary is to submit a time card each pay. Therefore, when we use the EMPNO field that the program read from the EMPMAST file to access the time card file randomly via the CHAIN operation, we expect that there will be a time card record present for the employee. If there is no time card, there is an error condition, signaled as shown below by indicator 92.

```
136 C...         HLDNO   CHAINTIMCRD...      92
137 C... 92              EXCPTNOTIME
```

In the HI indicator area of the CHAIN operation, you specify an indicator (92 in this case) that will turn on if the time card record is not found in the Time card file. If there is no time card, as you can see in statement 137, the program issues an EXCPT operation to an exeception print line named NOTIME.

The EXCPT operation is how we control printing from calculations. Having received its name in RPGII from its ability to perform output that was not the norm (the RPG cycle) but was the exception, the EXCPT has all of the capabilities of PRINT operations or WRITE operations or PUT operations in other languages. The EXCPT with an exception name (NOTIME in this case) prints the lines in output that have been coded with an "E" (exception) in column 15, rather than the H (Header) or the D (Detail). See Figure 15-5.

Moreover, it prints them immediately. Since there is only one RPG cycle in the PAREG2 program, using the D entries cannot work. The output coding for the No Time Card error message is shown below in Figure 15-

5, Note the E in column 15 (D in FMT) of the Record ID and Control statement # 223. Note also that when the PRTLN1 exception line prints at statement # 229, that an additional "** No Time Card **' message – also conditioned by indicator 92 -- appears on the normal employee print line for the report. While you are noting things, you may have noticed that the field names in Figure 15-5 do not begin with EMP as in EMPMAST. That's because they are not from EMPMAST. This will be explained below when we take a deep look a the control break logic.

Figure 15-5 PAREG2 Exception Output Records for Printing

```
FMTO ONamDFBAN03Excnam...............................
223  O...  E 11   NOTIME
224  O...         HLDSTA      4
225  O...         HLDCTY     29
226  O...         HLDNO      27
227  O...                    53 'No Time Card this pay'
228  O...                    71 ' period for below:'
229  O...  E 01   PRTLN1
230  O...         HLDSTA      4
FMTP ….......N03Field+YBEnd+PConstant/editword+++++
231  O...         HLDCTY     29
232  O...         HLDNO      27
233  O...         HLDNM      52
234  O...      N92HLDPAY1B   77
235  O...      N92HLDHRS1B   67
236  O...         HLDRAT1    57
237  O...       92...        76 '** No Time Card **'
```

When indicator 92 is not on, that means that we have successfully chained to the TIMCRD file and there is a time card for the employee. This is the equivalent function to the MR ID turning on. Thus, N92 means MR and it means that the information from the master and the time card are in memory and ready for processing.

There is one more file to bring into the picture, the new SALFILE. The operations to access the SALFILE and the operations to prepare the SALARY to replace the PAY calculation are shown in Figure 15-6. Let's decode a line at time to see what this little routine actually does.

Figure 15-6 Salary Routine

```
141  C  N92       HLDNO     CHAINSALFILE                  98
142  C  N98N92              Z-ADDSALYR      HLDSAL  60
143  C  N98N92    SALYR     DIV  52         HLDPAY
144  C  N98N92    HLDPAY    ADD  CTYPAY     CTYPAY  92
145  C  N98N92              Z-ADD0          HLDHRS
```

In statement 141, the employee # from the master record being processed is used to CHAIN (random read by key) to the SALFILE. THE HI indicator is 98 and this comes on if there is no SALARY record. You may recall that there is an EMPSCD field that was newly added to the employee master file to contain a Y if the employee is salaried. However, this code is not used in this program. Instead, after adding the code to the master, the designers decided to use the absence or presence of a salary record as the decision point for whether an employee is to be processed with a salary or whether hours worked are to be multiplied by rate to produce the pay.

So, if indicator 98 is on, that means we have an hourly employee. It is not an error condition by design. Following #141, you see four lines of code conditioned by the negative of indicator 98 and the negative of indicator 92. In English this says that each of the four calculations from 142 to 145 will be executed only if there is a time card record (company policy) and there is a salary record (salary routine). So, if there is no time card, the exception output at statement 137 delivers the missing time card message and the salary computations do not take place. Inn fact, as you can see in statement 141, the CHAIN to the salary file does not even take place if there is no time card.

So what happens in the salary routine if the salary record is found – also meaning there was a time card record? At statement 142, the yearly salary is moved into the current payroll record being processed (HLDSAL). At statement 143, the yearly salary is divided by 52 to create the weekly salary and it is stored in the PAY record for the current employee (HLDPAY). At 144, the pay is added to the CIT total for control break printing. Finally, at 145, the current employees hours are erased so that salaried employees can be differentiated in reports by not having hours printed.

That's about all of the extra code required to process matching records function with the salary option in this program – except for one thing. Because we are reading the EMPMAST file as the driver for this program's function, we have no way of knowing through this means as to whether we may have a missing master or a time card record that possibly does not match a master record.

Since we access a time card record directly by key only after first reading an EMPMAST record, we do not necessarily process each of the time card records in the file. Therefore, it is possible that we have a time card record and there is no matching master for it.

In PAREG with real MR logic and the cycle, this was not a problem. On line 43 pf PAREG, for example we printed a message that there was no matching master merely by using the NMR status and the record ID of the time card file (02). There was not routine needed to find a missing master. The RPG cycle provided this status with the NMR 02 indication. Boy, was that easy. It is not that easy doing it manually with a random file. As noted previously, if we had read all fields sequentially, the NMR conditions would be easier to spot but the code to process all files sequentially with no MR assistance by RPG would be a lot of coding. So, we opted for the easy way.

Now, we have to solve the problem that this solution created. How do we find missing masters for our time card records. The answer actually is simple. We must read each time card record and check each master to see if it exists for that particular employee time card record. In the beginning of the program at statement 71, as shown below, we launch the subroutine that gets this job done.

```
070 C* Check to see if there is a missing master
071 C                             EXSR CHKMST
```

What is a subroutine?

In chapter **** we discussed subroutines to an extent in their role in structured operations. This explanation is better and it is appropriate for the work done in the PAREG2 program.

All computer languages have subroutines. These are little chunks of code that can be isolated from all other code without the risk of one routine bumping into another. Using subroutines as tools to attack lower-level tasks in a program is a much better approach than using straight line code with decisions and branches. In the latter there is ample opportunity for a hunk of code to be inadvertently executed merely because "it is there."

In computer science, there are all kinds of techniques for writing programs and all of them like code segregation to avoid the problems of straight-line coding. Some like bottom up programming and some like top down programming and there are many wrinkles in between such as stepwise refinement. Since this is not a computer science theory book per se, the use of subroutines in the sample programs is similar in a sense to the top-down and bottom up approaches. The question has to do with when do you write the subroutines.

In bottom up, you would write the subroutines first while in top down you would write the calls to the subroutines and then provide some pseudo code for what you expect the subroutine to perform. In RPG, one would typically place the subroutine call at the top of the program and then create the subroutine at the bottom of the program. There would typically be no pseudo code because RPG is actually a high enough language that it is not really necessary to write pseudo code.

The other thing that the use of subroutines mimics, but not strictly is the HIPO (Hierarchical Input Process Output) methodologies which promote the use of separate modules to provide the major functions of the program. In RPG/400, the tool that you have to create these modules within a program is the EXSR statement. To create modules outside of the RPG program, RPG offers the CALL statement which is not used in the PAREG2 program.

Check for Missing Master

If you examined the code at the top of the PAREG2 program that initiates the check for the missing master, you would first find the EXSR CHKMST at statement 71. It is within the RUNREG subroutine that is at statement 69. The RUNREG subroutine begins as shown on statement

69 in Figure 15-7. It is called from the top of Calculations at statement 65 as shown in Figure 15-7.

Notice immediately after this statement that there is a SETON LR statement at 66. Since the entire Payroll Register is printed within the RUNREG subroutine, there are no real straight line calculations in this program. Only two statements in the program come before the first subroutine, 65 and 66. Here is how a subroutine works in RPG.

At statement 65, RPG encounters the EXSR RUNREG and immediately branches to the subroutine beginning with statement 69, the BEGSR statement for RUNREG. The first thing that happens in this very modularized program is that RUNREG executes the CHKMST code which is shown in Figure 15-8. Eventually the RUNREG finishes all of tis work and finds itself at statement 111 as shown in Figure 15-1.

When a subroutine hits its ENDSR, it is finished with its work. The ENDSR operation defines the end of an RPG/400 subroutine. It must be the last statement in the subroutine. The ENDSR operation ends the subroutine (RUNREG in this case) and its default is to causes a branch back to the statement immediately following the EXSR operation. Notice that the statement following the EXSR RUNREG in the PAREG2 program is SETON LR..

Though there are only two mainline calculations, statement 65 and 66, and because there is no second RPG cycle in this program, when the PAREG subroutine is finished, the program is finished. Actually since the mainline would have just one statement without he SETON LR being included, there would be no place for the EXSR to return. In reality it would return to the end of detail calculations (prior to the first subroutine) but it would gets stuck since there is no second cycle in a program without a primary or secondary file.

Thus, once the detail calculations are over, the program has no place to go. Moreover, since there is no primary or secondary file to force an LR condition naturally through the cycle when there are no more records to be read, IBM has made it a requirement that where there is no primary or secondary file, the programmer must set on LR to tell RPG that it is done. Statement 66 fulfills this requirement. When the RUNREG subroutine completes and branches back to the statement following the EXSR

RUNREG, it is at statement 66. It then sets on the LR indicator, and it gracefully ends.

Figure 15-7 Start the Main PAREG2 Register Subroutine

```
064 C*  Run default register with no prompt input
065 C…                  EXSR RUNREG
066 C…                  SETON                        LR
067 C*
068 C*  Body of Code- Controls Payroll Register
069 C…    RUNREG    BEGSR
070 C*  Check to see if there is a missing master
071 C…                  EXSR CHKMST
072 C*  Clear CHKMST fields to begin fresh register
073 C…                  EXSR CLR
074 C…                  CLOSEEMPMAST
075 C…                  OPEN EMPMAST
076 C*  First read ahead to check for Level breaks
077 C…                  READ EMPMAST                    91
```

The RPG/400 BEGSR operation at line 69 defines the beginning of a subroutine (RUNREG) and the ENDSR operation at statement 111 defines the end of the RUNREG subroutine. All the code in between the BEGSR and the ENDSR are considered part of the subroutine. At statement 71 the RUNREG subroutine executes the subroutine CHKMST to see if there are any missing payroll masters. This subroutine is shown in Figure 15-8. Following the CHKMST subroutine, at statement 206 as shown in Figure 15-8, control is passed back to the executable statement following the EXSR CHKMST (statement 73) in Figure 15-7.

Because the CHKMST routine as you can see in Figure 15-8 runs through the TIMCRD file sequentially and it randomly positions the EMPAST records and it puts values in the fields, it is wise to perform some housekeeping (refresh fields). The CLR subroutine handles this. Moreover, as a simple means of repositioning the file cursor for the first PAREG2 READ of the PAYMAST, at statements 74 and 75, the program closes the EMPMAST file and then reopens it. This has the effect of setting the file cursor at record 1.

Note: The file cursor is a place holder for a file so that the file can remember where it is when it when it is reading sequentially. Closing and reopening the file sets it back to record 1 thereby permitting the PAREG2 program to produce the register as if the CHKMST subroutine had never altered the file cursor.

Figure 15-8 Check for Missing Payroll Master

```
197  C*
198  C*  CHKMST Read time cards for missing masters & report
199  C*
200  C           CHKMST    BEGSR
201  C           *IN93     DOUEQ*ON
202  C                     READ TIMCRD                 93
203  C           EMPNO     CHAINEMPMAST            94
204  C   94                EXSR NOMAST
205  C                     ENDDO
206  C                     ENDSR
-----------------
147  C*
148  C* Write error msg for no master to separate Ext print file
149  C*
150  C           NOMAST    BEGSR
151  C                     WRITEHDR
152  C                     WRITEDTL
153  C                     ENDSR
```

In Figure 15-8, you can see the CHKMST routine. It is basically very simple. It starts with the BEGSR in statement 200. The next step is a Do until equal at statement 201. This is covered more in Chapter **** Structured Programming. We use it in this program because it works best. In essence it says Do the routine between the next statement and the statement before the ENDO at 205 until indicator 93 turns on.

So, how does indicator 93 turn on? Notice that in statement 202, the READ operations is coded to read the TIMCRD file one record at a time. The DOUEQ (Do until equal) will keep this loop (201 to 204) repeating until indicator 93 turns on. Indicator 93 is coded in the EQ portion of the READ statement and if you were to go back to Tables 13-2 and Table, you would see that it turns on when all the records in the file have been processed and the READ was unable to be satisfied since the file is at end with no more records to give.

So, how do you know if you have a missing master? In statement 203, the code says to CHAIN to the EMPMAST file and if there is no hit, turn on indicator 94. We are looking for time cards with no masters. The default is time cards with masters. So, indicator 94 tells the program that there is no master for this particular time card. When that is the case, statement 204, conditioned by indicator 94 executes an error subroutine called NOMAST.

For your convenience, the NOMAST subroutine is included at the bottom of Figure 15-8. Because the program executes the CHKMST at the beginning of the program even before it prints headings on the PAREG2 Register, it did not make sense to print error messages on the register report for missing time cards. Therefore, the program includes a new print file called ERROR, defined at statement 10 in Figure 15-1 and repeated below:

```
010   FERROR    O    E...                PRINTER
```

The E in column 19 differentiates this printer file from the QPRINT file defined in Line 9. The E stands for externally described. The information provided by the print file called ERROR could have been coded using O specifications in QPRINT but this technique increases your learning of RPG/400 and RPGIV capabilities.

Statements 51 and 52 send out two formats from the external print file, HDR for the header record and DTL for the detail record. For each error, a header and detail is written using the WRITE operation The EXCPT works only with program described files so the WRITE operation was the only choice to effect this error message.

Following the NOMAST subroutine, its ENDSR causes a return to the CHKMST subroutine as shown in Figure 15-8, statement 205. To DO or not To DO, that is the question. If indicator 93 has not yet turned on, it means that there are more time cards to check for missing masters. On the other hand, if indicator 93 is on, the CHKMST subroutine ands and control is passed to statement 73 of the RUNREG subroutine as shown in Figure 15-7.

Getting back to the NOMAST subroutine in Figure 15-8 for one more look, the externally described printer file used with this program is shown in Figure 15-9.

Figure 15-9 ERROR Externally Defined Printer File

```
FMTP  TName+RLen+TDpBLPosFunctions+++++++++++++++++++++++
03 A*******************************************************
04 A…RHDR
05 A*******************************************************
06 A*******************************************************
07 A…                     SKIPB(001)
08 A…                     SPACEA(002)
09 A…                   14
10 A…                     'This Error Report is a result of r-
11 A…                     eading the Time'
12 A…                   14
13 A…                     'Card file completely and finding a-
14 A…                      missing employee '
15 A…                      SPACEB(001)
16 A…                   14
17 A…                     'master record. The time card recor-
18 A…                      d was either    '
19 A…                   SPACEB(001)
20 A…                   14
21 A…                     'keyed wrong or the master has been-
22 A…                      inadvertently  '
23 A…                      SPACEB(001)
24 A…                   14
25 A…                     'deleted.  Check payroll input data-
26 A…                     .               '
27 A…                      SPACEB(001)
28 A*******************************************************
29 A*******************************************************
30 A…RDTL
31 A*******************************************************
32 A*******************************************************
33 A…                      SPACEB(001)
34 A…                   18
35 A…                    'Employee Number & HRS entered :'
36 A… EMPN      3S 0O  +1
37 A… EMPHRS    4S 2O  +3EDTCDE(1)
38 A*******************************************************
```

It is not the intention in this book to teach you how to create printer files, however, having the DDS as shown in Figure 15-9 is better than half the battle. As you can see in Figure 15-9, there are two record formats called

HDR (Statement 4) and DTL (Statement 30) and these are used in the NOMAST routine described above.

In the sample data for this program, there is one missing master. There is a time card record for EMPNO 6 but there is no master. When the PAREG2 program encounters this, it takes the compiled formats in the ERROR DDS shown in Figure 15-9 and it prints out a message that in the System i spool queue looks like the one shown in Figure 15-10.

Figure 15-10 ERROR Report – No Master for Time Card

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+
             This Error Report is a result of reading the Time
             Card file completely and finding a missing employee
             master record. The time card record was either
             keyed wrong or the master has been inadvertently
             deleted.  Check payroll input data.
                 Employee Number & HRS entered : 006   40.00
```

This completes our look at the code necessary to replace the matching records in the PAREG program with the manual methods using READ and CHAIN as well as the method needed to find the missing PAYMAST records. Now, it is time to examine the last and most substantial part of the new code to support the creation of a report without the RPG cycle. The following section shows the code necessary to perform control level breaks and control level output in RPG/400 without using the RPG cycle.

Control Level Breaks – No RPG Cycle

What causes a control level break? We have answered that a few times in this book but to remind you again, a control break occurs when the next record to be processed is different from the record currently being processed. How do you know what the next record is? Ho do you read a record and have it not be the record that your processing? Early RPG programs often used a facility called look-ahead fields, which are less frequently used today and so they are not used in this program. Instead, in order to simulate the notion of having two records in RPG memory at a time and always processing the lead record, we chose to use a data structure to store the record being processed.

Since we need to read the EMPMAST file, the first thing we do to assure two records in memory is to read in the first record at Statement 77 as shown in Figure 15-11. Statement 77 is executed once and only once in the program.

Figure 15-11 Level Calculations

```
076  C*  First read ahead to check for Levels
077  C…              READ EMPMAST…        91
078  C…              EXCPTHEADER
079  C…              EXSR PROCES
080  C*  Run register until end of file
089  C…    *IN91     DOUEQ*ON
090  C*  SECOND READ UNTIL EOF - FOR LEVEL CHECK
091  C…              READ EMPMAST…        91
092  C…91…           LEAVE…
093  C*  Replaces L1 coding as in PAREG
094  C*  LEVEL1 TEST-- See if current city different
095  C…    EMPCTY    IFNE HLDCTY
096  C…    EMPSTA    ORNE HLDSTA
097  C…              SETON…               L1
098  C…              EXSR LEVEL1
099  C…              ENDIF…
100  C*  Level 2 test -- See if state changed
101  C…    EMPSTA    IFNE HLDSTA
102  C…              SETON…               L2
103  C…              EXSR LEVEL2
104  C…              ENDIF
105  C…              SETOF…               L1L2
106  C…              EXSR PROCES
107  C…              ENDDO
108  C…              EXSR LEVEL1
109  C…              EXSR LEVEL2
110  C…              EXCPTLROUT
111  C…              ENDSR
```

Since statement 77 is executed just once and there will always be more than one records in PAYMAST file, the program does not have to do any work with the end of file indicator 91 from the READ statement. Instead, since this is the first record read for the Payroll Register, the next statement at 78 performs exception output to the lines marked header in Figure 15-1. The two output lines (210 and 215) are almost identical to the

header lines in the PAREG program except that there is an E instead of an H in column 15 as shown below:

```
210  OQPRINT  E  206                    HEADER
215  O        E  3                      HEADER
```

At statement 79 as shown in Figure 15-11, the next operation is to execute the PROCES subroutine. As you can see in Figure 15-1, the PROCES subroutine at line 79 in RUNREG executes from its BEGSR operation at line 157. It calls the following subroutines and then comes back to the RUNREG subroutine at statement 89.

- ✓ **MOVMS1   Statement 173**
- ✓ **PAYCLC   Statement 132**
- ✓ **PRNTLN   Statement 165**

The MOVMS1 Subroutine – First Record

The MOVMS1 subroutine as shown in Figure 15-1 starts at line 173. It sets up the program for keeping two records in memory. After this first READ statement in RUNREG, and the execution of the MOVMS1 subroutine, the fields of EMPMAST exist in two locations – the EMPMAST record itself and a data structure named HLDMST as shown in Figure 15-12. Through several Z-ADD statements and a number ov MOVE statements the individual fields in the PAYMAST record are moved to the individual fields of the HLDMST data structure. At this time in the program the contents of the current record (in process) as stored in the HLDMST data structure is exactly the same as the contents of the last EMPAST record read.

Before we continue let's examine the question, what is a data structure? Chapter **** goes over structures in detail. For now, consider a data structure as a means of defining one record in memory that has no

necessary association with a database file. Once the MOVMS1 subroutine finishes its job and returns to the PROCES subroutine, that one record structure is populated with the data from the first PAYMAST record , and that is the data that will be processed in the subsequent subroutines in PROCES, namely, PAYCLC and PRNTLN.

Figure 15-12 HLDMST Data Structure to Hold Current Record

```
049  IHLDMST      DS
050  I...                  1    30HLDNO
051  I...                  1   70 HREC
052  I...                  4   33 HLDNAM
053  I...                  4   23 HLDNM
054  I...                 34   382HLDRAT
055  I...                 39   58 HLDCTY
056  I...                 59   60 HLDSTA
057  I...                 61   650HLDZIP
058  I...                 66   66 HLDSCD
059  I...                 66   66 HLDDPT
```

The PAYCLC Subroutine - First Record

The PAYCLC subroutine as shown at statement 132 in Figure 15-1 calculates the Gross Pay by multiplying the hours from the time card by the rate from the employee master unless there is a salary record. If there is a salary record the routine uses the SALARY as the pay and stores it in the HLDPAY field which is defined within the PAYCLC routine itself. In addition to this the subroutine adds the HLDPAY amount to the city Level 1 total bucket, CITPAY. The PAYCLC routine then returns to the PROCES subroutine at line 160..

Line 160 in the PROCES subroutine is an EXSR PRNTLN statement. So, at this point control is passed to the PRNTLN routine at statement 165. The PRNTLN routine as shown in Figure 15-1 does 3 tasks. It asks that Headers be printed via the EXCPT HEADER statement at line 166 if the program senses that OVERFLOW has occurred on the printer. Additionally it sets off OF at line 167 if it is in. If OF is not on, the routine prints a detail line via the EXCPT PRNTLN1 statement at line 168 of the PRNTLN subroutine. The PRNTLN's ENDSR is at line 168 and at this point the ENDSR causes the program to branch back to

statement 161 which is the line following the EXSR PRNTLN line in the PROCES subroutine. This happens to be an ENDSR for the PROCES subroutine. From here the ENDSR causes the program to branch back to statement 89 in the RUNREG subroutine.

> **Caution: There are a few statement numbers that are skipped between lines 79 and 89 in the RUNREG subroutine. Do not assign any significance to this.**

At this point we have (1) read the EMPAST first record, (2) stored it in HLDMST, and (3) calculated gross pay and stored that in HLDPAY, and (4) printed the headings and the first detail line. Now we are finished with the unique things that must be done with the first record read when checking for control breaks. Right now it resides in EMPMAST and HLDMST and it has been processed.

Looking back at Figure 15-11, you see that statement 89 is s DO until statement. It is waiting for indicator 91 to turn on before it will end the DO loop. From Figure 15-11, you can see that this DOUEQ loop extends to statement 107 where there is an ENDO statement. So all of the statements from 90 to 106 if properly conditioned will execute as often as the loop permits. Indicator 91 comes on when the last EMPMAST record has been processed and the program is trying to read a record after the last record that is not there. This ends the loop. Now, let's see what else happens in this very important loop.

At line 91, there is a READ statement. This means that for every iteration of the DO loop, a record will be read from the PAYMAST file. When the end of file indicator (91) is turned on with no record returned at statement 91, (No intention of confusion with statement 91 and indicator 91 – it happens) the next line tests to see if the last loop iteration should continue. As you can see in Figure 15-11, statement 92, if end of file has been reached and thus no record has been read into PAYMAST, the loop is over and for that matter the program is almost over. The LEAVE operation takes the program to statement 108 which causes the Level 1 subroutine, Level 2 subroutine and Last record subroutine to execute. Following this, at statement 111 shown in Figure 15-11, the ENDSR for RUNREG is encountered. This takes the program to the statement following the EXSR RUNREG or statement 66. Here the LR is turned on and the program falls to the end of detail calculations. Since LR is on the program ends.

If there are records to process however, indicator 91 does not turn on. Therefore, the LEAVE operation is not taken and the loop in Figure 15-11 continues to line 93. The code from 93 to 97 is repeated below

```
093  C*  Replaces L1 coding as in PAREG
094  C*  LEVEL1 TEST- city different?
095  C…    EMPCTY    IFNE HLDCTY
096  C…    EMPSTA    ORNE HLDSTA
097  C…              SETON…           L1
098  C…              EXSR LEVEL1
099  C…              ENDIF…
```

Control Break Level 1 Processing

Since a state change by definition means that a City has also changed, the code in lines 95 and 96 tests to see if the current record being processed (stored in the HLDMST data structure) has a city or state change. The chances of a Wilkes-Barre, Alaska and a Wilkes-Barre Pennsylvania following each other in a payroll application are remote indeed but this code solves the problem if need be for other applications as well as this one.

What is an IF statement? Statement 95 contains an IF not equal operation (IFNE). It tests to see if the contents of the EMPCTY field which comes from the record just read but is not yet in process is not equal (different) to the contents of the HLDCTY field in the record being processed.

What is an OR statement? An OR operation extends and IF statement by adding another set of circumstances for the test to be true. So, if by chance the two cities were equal making the first condition in 95 false but the state comparison in the ORNE (or not equal) statement in line 96 was true (states are different, the result of the IF would be true and the lines between the IF and OR statements (97 to 98) would therefore be executed. On the other hand, if both conditions are false, meaning no

city change and no state change, then the operations preceding the ENDIF statement at line 99 would not be executed.

If you got comfortable with level logic while decoding the PAREG program, then this logic makes sense to you because it mimics the control break logic that RPG uses in the cycle. If you have not gotten comfortable with the L1 part of the cycle, this type of coding probably makes more sense to you.

So, if there is no control break, nothing happens in between records, but is there is a level 1 control break, statement 97 turns on the L1 indicator as a marker and then executes the L1 processing subroutine named LEVEL1 at statement 98. The ENDIF statement at statement 99 ends the IF statement that's started with statement 95. Each IF statement that you use in a program must be end with an ENDIF. If the either, any or all conditions in an ORed IF statement are true, the statements in between execute. If none are true they do not execute. If all conditions in an ANDed IF operation are true, the statements in between execute. If any condition in an ANDed IF is not true, the statements do not execute.

Before we come back for the L2 tests, let's look at the LEVEL1 subroutine that gets called when there is a control break. You can imagine what must happen. Just as with the PAREG program, the calculations done at L1 detail time would be need to be done in this LEVEL 1 subroutine. In other words, the CITPAY would be added to STAPAY Additionally the functions done at L1 output time (lines 238 to 241 in Figure 15-1) would need to be done to print the City totals. As you can see in the code snippet below the LEVEL 1 subroutine does exactly this.

```
113  C*  Level 1 Subroutine City Control break
114  C*
115  C...    LEVEL1    BEGSR
116  C...    CTYPAY    ADD  STAPAY    STAPAY  92
117  C...              EXCPTL1OUT
118  C...              ENDSR
```

Control Break Level 2 Processing

So now that we have processed L1 totals without the RPG cycle, let's move on to Level 2 totals. By the way, similar logic works as you extend the number of total levels to three and four and so on. In Figure 15-11, the RUNREG code dealing with Level 2 processing is as follows:

```
100   C*   Level 2 test Has state changed?
101   C… EMPSTA     IFNE HLDSTA
102   C…            SETON…            L2
103   C…            EXSR LEVEL2
104   C…            ENDIF
```

The Level 2 test is even easier than the Level 1 test. In this case, we already know that there is an L1 break. We just don't know if it was caused by a state change or not. SO we have to test to see if the state has changed. Again, the current record is in the HLDMST data structure and the next record to be processed is the record that was just read from PAYMAST. The two state fields SMPSTA and HLDSTA are compared to see if there is a state change. If there is no state change then none of the statements get executed but if the IFNE statement is evaluated as true and the states have changed, then line 102 and 103 are executed.

Line 103 is a marker to set on the L2 indicator in case we need it someplace in the program. The L1 and L2 indicators can be used as regular indicators if you choose as in this program. Once L2 is turned on, the LEVEL 2 subroutine is executed from line 103.

Before we check out the LR2 tests, let's look at the LEVEL2 subroutine that gets called when there is a control break. You can imagine what must happen. Just as with the PAREG program, the calculations done at L2 detail time would be need to be done in this LEVEL 1 subroutine. In other words, the STAPAY field value would be added to TOTPAY field. Additionally the functions done at L2 output time would need to be done to print the State totals from State total exception output lines (lines 242 to 245 in Figure 15-1). As you can see in the code snippet below the LEVEL 2 subroutine does exactly this.

```
120  C*   Level 2 Subroutine State break
121  C*
122  C...  LEVEL2     BEGSR
123  C...  STAPAY     ADD   TOTPAY     TOTPAY   92
124  C...             EXCPTL2OUT
125  C...             ENDSR
```

Process the Record Just Read

The last few statements in the RUNREG subroutine of PAREG2 are repeated below for convenience in referencing. These statements will be referenced in this section regarding processing the current record as well as the next section dealing with Final totals at LR time.

Staring with 105 below, you can see that the SETOF operation is turning off the L1 and L1 indicators. To repeat, we could have used any indicator number to represent the fact that it was L1 time or L2 time but since special indicators L1 and L2 are not controlled by RPG in this program, we took control of then. At the need of the L1 and L2 routines, the indicators are set off so that they can be tested properly during the next DO loop iteration.

Earlier in this chapter when we studied the first READ EMPMAST code, we examined in detail the workings of the PROCES subroutine in line 106. From this examination, we know that it moves the record just read into the current record slot (HLDMST) and it calculates gross pay and it prints the detail line. At statement 110 the major DUEQ loop for RUNREG finds its matching ENDO statement. As long as indicator 91, which indicates the end of file in the EMPMAST, is not on, the loop continues. But when 91 is on the Do loop ends and control is transferred to the first statement following the ENDDO. As you can see, this is statement 107 and the operations to be done are shown in statements 108 to 110 below.

```
105   C...        SETOF...                L1L2
106   C...        EXSR PROCES
107   C...        ENDDO
108   C...        EXSR LEVEL1
109   C...        EXSR LEVEL2
110   C...        EXCPTLROUT
111   C...        ENDSR
```

Final Total Processing

Just as a state change would force a city change (a Level 2 change would force a Level 1 change), so also does a last record change force a next highest level change (L2 in PAREG2). So, when the RUNREG DO loop is completed at statement 107, the program is all over but the totals.

Since there is no record at this point that was just read, it would not help at all to compare the current record in HLDMST with the record just read. If we were clearing this record out each time prior to reading it in, then that type of code could work. However, it is unnecessary. For when the program is just about done and it has had or it is about to have (Line 66) LR turned on, the record being processed is clearly the last and whether it is a single record or a group of records from the same city, when the next record is no record, there is a level change. In fact, all of the level fields have changed. The contents logically no longer represent the last PAYMAST read because no PAYMAST was read. Logically all of the new fields are zero or blank. SO if we were to compare the EMPSTA with the HLDSTA we would get a logical not. HLDSTA would contain the state from the last record actually read and EMPSTA would contain a logical blank.

Since we know the behavior of all LR conditions, there is no need to test the other levels. LR forces all other control breaks. So the record in HLDMST, though it has been fully processed, has not had its City and State totals for this last group written until this point in the program Line 108 gets the last City total out and 109 gets the last state total out 110 prints the final totals and then it runs into the ENDSR for RUNREG at 111. As we have noted several time sin discussing the logic of PAREG2, this forces the program back to the last detail calculation at line 66, which

turns on LR and then drifts quietly in to post detail calc oblivion as the program ends.

There are three more permutations of PAREG2. The version we just studied, which should be called PAREG2P, is the second longest because it is program described. The longest version is the RPGIV version (PAREG2P4) converted with the IBM CVTRPGPGM command already discussed. Since RPGIV does not like more than one indicator on a conditioning calculations specification and it needs space for keywords on may specifications, it is always a bigger (# of lines) program than a corresponding RPG/400 version. The shortest version PAREG2E is RPG/400 Externally described and the second shortest version (PAREG2E4) is the RPGIV version.

The externally described RPG/400 version is presented in its entirety in Figure 15-13.

Figure 15-13 Externally Described PAREG2 as PAREG2E

```
     *PAREG2P int. described PAREG - no MR, No LX totals
001 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
002 H
003 F*
004 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005 F*
006 FEMPMAST IF  E           K        DISK
007 FTIMCRD  IF  E           K        DISK
008 FSALFILE IF  E           K        DISK
009 FQPRINT  O   F      77      OF     PRINTER
010 FERROR   O   E                    PRINTER
011 IHLDMST    E DSHLDMAST                      70
012 I                                        1   70 HREC
013 I                                        4   23 HLDNM
014 C*
015 C* RPG CALCULATION SPECIFICATION FORMS
016 C*
017 C*  Run default register with no prompt input
018 C                      EXSR RUNREG
019 C                      SETON                      LR
020 C*
021 C*  Body of Code that Controls the Payroll Register
022 C            RUNREG    BEGSR
023 C*  Check to see if there is a missing master
024 C                      EXSR CHKMST
```

```
025 C*  Clear fields from CHKMST to begin fresh register
026 C                     EXSR CLR
027 C                     CLOSEEMPMAST
028 C                     OPEN EMPMAST
029 C*  First read read ahead to be able to check Levels
030 C                     READ EMPR                    91
031 C                     EXCPTHEADER
032 C                     EXSR PROCES
033 C*  Run register until EOF is hit... NU1 end program
034 C           *IN91     DOUEQ*ON
035 C*  SECOND READ -- UNTIL EOF NEED FOR LEVEL CHECK
036 C                     READ EMPR                    91
037 C   91                LEAVE
038 C*  Replaces L1 coding as in PAREG
039 C*  LEVEL 1 TEST -- See if current city is different
040 C           EMPCTY    IFNE HLDCTY
041 C           EMPSTA    ORNE HLDSTA
042 C                     SETON                    L1
043 C                     EXSR LEVEL1
044 C                     ENDIF
045 C*  Level 2 test -- See if current state is different
046 C           EMPSTA    IFNE HLDSTA
047 C                     SETON                    L2
048 C                     EXSR LEVEL2
049 C                     ENDIF
050 C                     SETOF                    L1L2
051 C                     EXSR PROCES
052 C                     ENDDO
053 C                     EXSR LEVEL1
054 C                     EXSR LEVEL2
055 C                     EXCPTLROUT
056 C                     ENDSR
057 C*
058 C*  Level 1 Subroutine - Control break on City
059 C*
060 C           LEVEL1    BEGSR
061 C           CTYPAY    ADD  STAPAY    STAPAY  92
062 C                     EXCPTL1OUT
063 C                     ENDSR
064 C*
065 C*  Level 2 Subroutine - Control break on State
066 C*
067 C           LEVEL2    BEGSR
068 C           STAPAY    ADD  TOTPAY    TOTPAY  92
069 C                     EXCPTL2OUT
070 C                     ENDSR
071 C*
072 C*  PAYCLC  Calculates Gross PAY from HRS or Salary
073 C*  Also calculates "net pay" and updates YTD files.
```

```
074 C*  Calculate pay for HELD record
075 C*  If Salaried, do not use RATE multiplier
076 C*
077 C           PAYCLC    BEGSR
078 C                     SETOF                      9298
079 C                     Z-ADD0         HLDSAL  60
080 C* REPLACES MR CYCLE WORK
081 C           HLDNO     CHAINTIMCRD                92
082 C   92                EXCPTNOTIME
083 C  N92      HLDRAT    MULT EMPHRS    HLDPAY  72
084 C  N92                Z-ADDEMPHRS    HLDHRS  92
085 C  N92      HLDPAY    ADD  CTYPAY    CTYPAY  92
086 C  N92      HLDNO     CHAINSALFILE               98
087 C  N98N92             Z-ADDSALYR     HLDSAL  60
088 C  N98N92   SALYR     DIV  52        HLDPAY
089 C  N98N92   HLDPAY    ADD  CTYPAY    CTYPAY  92
090 C  N98N92             Z-ADD0         HLDHRS
091 C                     ENDSR
092 C*
093 C* Write error msg- no master to External print file
094 C*
095 C           NOMAST    BEGSR
096 C                     WRITEHDR
097 C                     WRITEDTL
098 C                     ENDSR
099 C*
100 C*  Process line item
101 C*
102 C           PROCES    BEGSR
103 C                     EXSR MOVMS1
104 C                     EXSR PAYCLC
105 C                     EXSR PRNTLN
106 C                     ENDSR
107 C*
108 C* Print Detail Line on Register
109 C*
110 C           PRNTLN    BEGSR
111 C   OF                EXCPTHEADER
112 C   OF                SETOF                      OF
113 C                     EXCPTPRTLN1
114 C                     ENDSR
115 C*
116 C* Move Fields to Hold Area for Level Info / Comp
117 C*
118 C           MOVMS1    BEGSR
119 C                     Z-ADDEMPNO     HLDNO
120 C                     MOVELEMPNAM    HLDNAM
121 C                     Z-ADDEMPRAT    HLDRAT
122 C                     MOVELEMPCTY    HLDCTY
```

```
123 C                         MOVELEMPSTA    HLDSTA
124 C                         Z-ADDEMPZIP    HLDZIP
125 C                         MOVELEMPSCD    HLDSCD
126 C                         MOVELEMPDPT    HLDDPT
127 C                         ENDSR
128 C*
129 C* CLR Clear fields were used in missing master test
130 C*
131 C          CLR       BEGSR
132 C                         Z-ADD0         EMPNO
133 C                         MOVE *BLANKS   EMPNAM
134 C                         Z-ADD0         EMPRAT
135 C                         MOVE *BLANKS   EMPCTY
136 C                         MOVE *BLANKS   EMPSTA
137 C                         Z-ADD0         EMPZIP
138 C                         MOVE *BLANKS   EMPSCD
139 C                         MOVE *BLANKS   EMPDPT
140 C                         Z-ADD0         EMPHRS
141 C                         ENDSR
142 C*
143 C* CHKMST Read TIMCRD look for missing masters,report
144 C*
145 C          CHKMST    BEGSR
146 C          *IN93     DOUEQ*ON
147 C                    READ TIMCRD
149 C          EMPNO     CHAINEMPMAST              94
150 C   94               EXSR NOMAST
151 C                    ENDDO
152 C                    ENDSR
153 C*
154 C*
155 C*
156 C*
157 C*
158 O*
159 O* RPG OUTPUT SPECIFICATION FORMS
160 O* 161 to 198 are the same in all PAREG2 programs
```

When you become an RPG guru, terms like externally described and internally described will have become second nature to you. Figure 15-13 directly above shows the same PAREG2 program that we coded with program (internally) described data in Figure 15-1. This time, however, the program is coded using externally described files. Of course, because we do not like externally described printer files (though we use one in this program to show you how they work), we use program described exception output that is identical to that for the program described

version of the program PAREG2 in Figure 15-1. Therefore, we skipped lines 161 to 198 of the externally described program PAREG2E. Its output specs look exactly like the output lines in Figure 15-1. No change.

PAREG2E, the externally described version in Figure 15-13 checks in at 198 lines of code. This is 50 statements less than PAREG2P, the internally described version. Considering that a lot of code is swallowed up in output definitions, which are equal in both programs, this clearly shows, even when the number of fields in files is relatively small, that there can be great lines of coding savings by using externally described data.

What are the Differences in PAREG2 – Internal v External?

To know what has changed, you must contrast the major differences. Figure 15-14 shows the File description specifications of both the internal and external versions – one atop of the other.

Figure 15-14 Contrast External wit Internal File Descriptions

External Version

```
004 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005 F*
006 FEMPMAST IF  E            K        DISK
007 FTIMCRD  IF  E            K        DISK
008 FSALFILE IF  E            K        DISK
009 FQPRINT  O   F       77     OF     PRINTER
010 FERROR   O   E                     PRINTER
```

Internal Version

```
006 FEMPMAST IF  F       70  3AI     1 DISK
007 FTIMCRD  IF  F        7  3AI     1 DISK
008 FSALFILE IF  F        9  3AI     1 DISK
009 FQPRINT  O   F       77     OF     PRINTER
010 FERROR   O   E                     PRINTER
```

In Figure 15-14, it is clear to see that it takes a lot less figuring and column coding in File Descriptions to work with the External File descriptions than it does to work with the internal descriptions. Moving from left to right across the columns of the File Description specification, the first difference is in column 19. The choices are F or E. You choose F for fixed form meaning program described if you want to describe the files within the program. You choose E when you want to use the already existing descriptions within the database. As long as the field names you use in the program are the same names as those in the external description, most of the rest of the program does not matter. If the names are different, then you have lots of work in renaming fields. The both versions of AREG use the same field names in calculations so this is not an issue.

Moving from column 15 across the File Description specification, you can see that to define an internal file, you need to tell RPG the record length, the key length, whether it uses alphabetic keys, whether it is indexed, and where the key field begins in the record. For the external versions of database files, to process by key, the programmer merely needs to put a K in column 31 and that's that. RPG knows how to look at the file while the program is compiling to pick up the other information that is needed, including input and database output specifications.

You can see by comparing the input specifications in Figure 15-1 for the internally described version to the input specs in the external version in Figure 15-13 that there is substantially less coding in INPUT. In fact, in many programs, thee are no input specs and for database adds and updates, there are no output specs required.

One of the tricks we used in the external version PAREG2E is that we created a file on disk with the description of the HLDMST data structure, which is where the in-process record is stored. This is called an externally described data structure. Because so often there is a database file that mirrors almost or exactly a structure you would like to define in the program, the RPG/400 and RPGIV compilers permit you to point to a database file for the description of the data structure. This does not mean that the database file is going to put anything into the data structure. In fact, it is merely a code saving device.

There is no relationship with the description of the data structure and the database file from which the description is "stolen." However, in cases in which the programmer can save 100 lines of code by referencig an existing file to obtain the structure subfields, nobody really cares if the structure and the file have any relationship. Again, it is just a code saving mechanism, and quite clever at that.

The PAREG2E version uses one such structure. The line of code as written by the programmer is as follows:

```
FMT.IDsname....NODsExt-file++...OccrLen+
011 IHLDMST    E DSHLDMAST...         70
```

This line of code at line 11 of the PAREG2E program says the following to the RPG/400 compiler:

Hi Mr. Compiler. I would like you to define a data structure for me that is 70 positions in length. The subfield definitions for this data structure will come from an externally described file which just happens to have the data definition for the record that I would now like to define. No, Mr. compiler, after you grab the description of this to be data structure from the internals of the EMPMAST database file, please do not link inany way this structure to that database file. Thank you.

That about does it for the differences between the externally described version and internally described RPG/400 versions of PAREG2.

RPG IV Program Versions

Since none of the code conversions we are doing have anything to do with EVAL constructs, BIFS, or free form RPG. The RPGIV lines of code versions that are shown in the rest of this chapter include only those elements that are changed substantially. In other words, each statement is different because RPGIV statements are somewhat different. However, certain elements of the language have changed substantially such that DS is now on the D spec and options that are used in this program are now coded using keywords on various specification types.

RPGIV Program Described PAREG2P4

The lines of code that are the same as you would expect are not included in Figure 15-15 (Internally Described RPGIV) nor Figure 15-17 Externally Described RPGIV.

Figure 15-15 Internally Described RPGIV Version PAREG2P4

```
     * PAREG2P4 internally described PAREG RPGIV
     * - no MR, No LX totals
001 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
002 H
003 F*
004 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005 F*
006 FEMPMAST   IF   F   70      3AIDISK    KEYLOC(1)
007 FTIMCRD    IF   F    7      3AIDISK    KEYLOC(1)
008 FSALFILE   IF   F    9      3AIDISK    KEYLOC(1)
009 FQPRINT    O    F   77         PRINTER OFLIND(*INOF)
010 FERROR     O    E              PRINTER
011 D*
012 D*
013 D HLDMST          DS
014 D  HLDNO                  1      3  0
015 D  HREC                   1     70
016 D  HLDNAM                 4     33
017 D  HLDNM                  4     23
018 D  HLDRAT                34     38  2
019 D  HLDCTY                39     58
020 D  HLDSTA                59     60
021 D  HLDZIP                61     65  0
022 D  HLDSCD                66     66
023 D  HLDDPT                67     70
024 I*
025 I* RPG INPUT SPECIFICATION FORMS
026 I*
027 I*
028 I*  EMPMAST is the employee master file
029 I*  Basically same as the RPG/400 version
032 IEMPMAST   AA  01
033 I                                 1   70  EREC
034 I                                 etc
043 I*
044 I*  TIMCRD File basically the same
049 ITIMCRD    AB  02
050 I                                 1    3 0EMPNO
```

```
051 I                                 4    7 2EMPHRS
052 I*
053 I*  SALCRD basically the same
057 ISALFILE   AC  03
058 I                                 1    3 0SALENO
059 I                                 4    9 0SALYR
060 I*
061 C*
062 C* RPG CALCULATION SPECIFICATION FORMS
063 C*
064 C*  Run default register with no prompt input
065 C             EXSR      RUNREG
066 C             SETON…                       LR
067 C*
068 C*  Body of Code running of the Payroll Register
069 C     RUNREG  BEGSR
070 C*  Check to see if there is a missing master
071 C             EXSR…CHKMST
072 C*  Calculations are basically the same as PAREG2P
        One conditioning ind, SETOFF and  EXCEPT below:
…
134 C  N92HLDNO…  CHAIN     SALFILE…           98
135 C  N98
136 CANN92        Z-ADD     SALYR… HLDSAL…   6 0
137 C  N98
138 CANN92SALYR   DIV       52…    HLDPAY
139 C  N98
140 CANN92HLDPAY  ADD       CTYPAY…CTYPAY    9 2
141 C  N98
142 CANN92        Z-ADD     0      HLDHRS
…
159 C*
160 C* Print Detail Line on Register
161 C*
162 C     PRNTLN  BEGSR
163 C   OF        EXCEPT    HEADER
164 C   OF        SETOFF…                    OF
165 C             EXCEPT    PRTLN1
166 C             ENDSR
167 C*
…
210 O* RPG OUTPUT SPECIFICATION FORMS
211 O* Output specifications basically same as PAREG2P
212 OQPRINT    E…     HEADER…  2 06
213 O…                            32 'THE DOWALLOBY
```

File Description

As you check out the RPGIV converted code in Figure 15-15, the first big noticeable difference comes in File Description. Of course if we had any Control SPEC (H) entries, we would have seen those come over as keywords since the RPGIV H spec has no columnar values at all. Let's look at one H spec for an indexed file EMPMAST. It is the same as the specifications for the TIMCRD and the SALFILE so by examining it in Figure 15-16, we are in essence looking at the whole notion of fully procedural indexed files (or logical files) in RPGIV.

Figure 15-16 – Contrast RPG/400 with RPGIV File Description

```
RPGIV Database
FMT FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++
006 FEMPMAST   IF   F   70     3AIDISK    KEYLOC(1)

RPG/400 Database
FMT.FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.
006 FEMPMAST IF  F      70  3AI     1 DISK

RPGIV Printer FIle
FMT FFilename++IPEASFRlen+L…Device+.Keywords++
009 FQPRINT    O    F   77… PRINTER OFLIND(*INOF)

RPG/400 Printer File
FMT.FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.
009 FQPRINT  O   F      77     OF     PRINTER
```

For program described indexed files in RPGIV, IBM chose to keep the length of the key (3), the notion of alphabetic keys or packed keys (A) and the indexed file designation (I) as columnar. In the RPG/400 version, you can see the 3AI very plainly about eight spaces to the left of the DISK device. In RPGIV, the 3AI is right next to the DISK device. In RPG/400, the key field starting location is columnar, two spaces to the left of the DISK device in column 38 of the F form. IBM left no room for the key field starting location with RPGIV so they invented a keyword (KEYLOC) to handle this entry. As you can see in the RPGIV the key

starting location of 1 is accommodated with the Keyword entry KEYLOC(1).

In RPG/400, the programmer had the opportunity to place a value in the overflow indicator column to define the indicator for printer overflow. With RPGIV, this entry is no longer columnar. It is keyword oriented and you can see in Figure 15-16 that the completed keyword specification for overflow in the PAREG2P4 program is **OFLIND(*INOF**). It means the same. It just looks different. That sums up the PAREG2P4 File Description differences.

The "D" Spec

The next big difference starts at line 13 and goes to line 23 of the PAREG2P4 program in Figure 15-15. In the RPG/400 version, in lines 49 to 59 of the input "I" specification, the HLDMST data structure is defined. When this was converted to RPGIV, the information was transferred to the Definition "D" specification. Several of the "D" specs from PAREG2P4 are included below for closer examination:

```
013 D HLDMST                DS
014 D   HLDNO                              1      3   0
015 D   HREC                               1     70
016 D   HLDNAM                             4     33
017 D   HLDNM                              4     23
```

The "D" spec overall is very easy to relate to compared to defining a data structure using the convoluted RPG/400 "I" spec DS formats for definition and subfields. The first thing that you may have noticed is that you can indent your field names to make the more readable. There is extra space so that field names no longer need to be left justified.

Intuitively, you can see the data structure name in line 13 with the familiar DS designation. Also, intuitively, you can see the from and to positions of the subfields and the subfield names. Again intuitively, knowing that EMPNO is numeric with zero decimals it is easy to spot the zero in line 14 in the "current record" version of EMPNO named HLDNO.

Conditioning Indicators and Operation Names

In lines 134 to 142 of Figure 15-15, you can see that one indicator was not enough to condition several of the calculations in this block of code. Instead of the three spaces for conditioning indicators in RPG/400 which came in handy for RPG indicator lovers, IBM opted to leave space for just one. The implicit ANDing of three conditioning indicators was removed from the RPGIV "C" specification definition. Each of the statements selected need two indicators to condition their respective operations. Therefore, two lines of code are needed and the mission is accomplished by ANDing them together.

Table 14-1 in Chapter 14 shows the RPGIV operations that received name changes when IBM defined RPGIV. Tow of these name changes are shown in the block of code from 159 to 167 of Figure 15-15. The operation statements look very much like their counterparts in RPG/400 except for the change to the operation name themselves. The two operations in this block of code of course are SETOFF which grew on character from SETOF and EXCEPT, which also grew one character from EXCPT. Both of these changes as well as the other operation changes contribute to making RPGIV a more readable language.

That's about it for major changes in the PAREG2P4 program. Now, right before the chapter wrap-up, let's take a look at one more flavor of PAREG2 code known as PAREG2E4. This is the RPGIV externally described version of the PAREG2 program.

RPGIV Externally Described PAREG2E4

The PAREG2E4 is the RPGIV version of the PAREG2E program that we examined earlier in this chapter. It is very similar. In Figure 15-17, the code that you see is the code that either has some reference value in itself to make the program recognizable or it is code that is substantially different and therefore worthy of review for learning purposes. The full versions of all this code are available on the Lets Go Publish Web site (www.letsgopublish.com) as well as the Kelly Consulting Web site (www.kellyconsulting.com).

Figure 15-17 Externally Described RPGIV Version PAREG2E4

```
001 H* RPG HEADER (CONTROL) SPECIFICATION FORMS
002 H
003 F*
004 F* RPG FILE DESCRIPTION SPECIFICATION FORMS
005 F*
006 FEMPMAST   IF   E           K DISK
007 FTIMCRD    IF   E           K DISK
008 FSALFILE   IF   E           K DISK
009 FQPRINT    O    F   77        PRINTER OFLIND(*INOF)
010 FERROR     O    E             PRINTER
011 D HLDMST        E DS             70    EXTNAME(HLDMAST)
013 D  HREC                     1    70
015 D  HLDNM                    4    23
```

If you are an amateur humorist as I, you must now be singing. "Is that all there is?" Yes, that's about it. There are no input specs needed. The calcs are the same and the output is the same. Theoretically, we could have had no code here. File descriptions changes only in the placement of the K for the external definition of a keyed file. QPRINT changes with the OFLIND keyword but it is the same as the program described version. Additionally, the HLDMST data structure is externally described but this too is very similar to the RPG/400 version except for one thing. The external name is now a keyword, EXTNAME(HLDMAST).

All of this reaffirms a point that we have made several times in this book. With RPG/400 as the base line, RPGIV is not that much different. If you can code in RPG/400, there is no reason to not code in RPGIV – at least for base functions. Later, you can have all the fun you want with the elements of RPGIV that change coding substantially: EVAL, BIFs, and free-form RPGIV.

Chapter Summary

In this chapter we made a dramatic change in programming from the PAREG program that we had been studying throughout this book. There is no RPG program cycle in this program yet the program accomplishes the mission of matching records and control level processing, The PAREG2 program is shown in internal and external versions for

RPG/400 and RPGIV. The major differences between the internal and external versions is that there is substantially less coding in the external version than the internal version.

All of this reaffirms a point that we have made several times in this book. With RPG/400 as the base line, RPGIV is not that much different. If you can code in RPG/400, there is no reason to not code in RPGIV – at least for base functions. Later, you can have all the fun you want with the elements of RPGIV that change coding substantially: EVAL, BIFs, and free-form RPGIV.