Moving to Integrated Language Environment for RPG IV

Document Number GG24-4358-00

April 1995

International Technical Support Organization
Rochester Center

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

First Edition (April 1995)

This edition applies to Version 3 Release 1 of ILE RPG/400 Licensed Program 5763-RG1 for use with Version 3 Release 1 of the IBM Operating System/400 Licensed Program (Program 5763-SS1).

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 977 Building 663-3
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1995. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Abstract

This document is unique in its detailed coverage of the new RPG IV language and aspects of the Integrated Language Environment. The purpose of this document is to provide an introduction to the RPG IV language and guidelines for implementing applications using the Integrated Language Environment. The publication *Integrated Language Environment Concepts*, SC41-3606, should be considered a prerequisite to this publication.

This document is intended for experienced programmers, analysts, and implementers who are responsible for the creation and maintenance of application programs on the AS/400. It assumes the reader has a fairly good background in the AS/400 programming environment.

(205 pages)

© Copyright IBM Corp. 1995

Contents

Abstract . . . iii

Special Notices . . . xv

Preface . . . xvii
How This Document Is Organized . . . xvii
Related Publications . . . xviii
International Technical Support Organization Publications . . . xviii
Acknowledgments . . . xix

Chapter 1. Introduction to ILE RPG . . . 1
1.1 Integrated Language Environment (ILE) . . . 1
1.1.1 ILE Languages . . . 2
1.1.2 Application Development Environment . . . 4

Chapter 2. RPG Specification Sheets . . . 7
2.1 RPG IV Specifications Statements . . . 7
2.2 The Control (H) Specification Statement . . . 8
2.3 The File Description (F) Specification Statement . . . 9
2.4 The Definition (D) Specification Statement . . . 12
2.4.1 Examples for Declaring Data Items Using the Definition Specification 18
2.5 The Input (I) Specification Statement . . . 20
2.6 The Calculation (C) Specification Statement . . . 23
2.6.1 The New Calculation Specification Layout . . . 24
2.7 The Output (O) Specification Statement . . . 25
2.8 The File Extension (E) Specification Statement . . . 27
2.9 The Line Counter (L) Specification Statement . . . 27

Chapter 3. RPG IV Functions and Features . . . 29
3.1 Operation Codes . . . 29
3.1.1 Renamed Operation Codes . . . 29
3.1.2 New Operation Codes to Process Date and Time Data Types . . . 30
3.1.3 New Operation Code for Static Call . . . 33
3.1.4 New Operation Codes for Structured Programming . . . 34
3.2 Symbolic Names . . . 36
3.2.1 Upper/Lowercase . . . 36
3.2.2 Name Length . . . 37
3.2.3 Underscore . . . 37
3.2.4 Blank Lines . . . 37
3.2.5 Examples . . . 37
3.3 Changes in Limits . . . 37
3.4 Built-in Functions in RPG IV . . . 38
3.5 Using Date and Time Formats and Operations . . . 40
3.5.1 Initializing Date and Time Data Type Fields . . . 41
3.5.2 Example of Initializing Date and Time Data Type Fields . . . 42
3.5.3 Calculations with Date and Time Data Types . . . 43
3.5.4 Date and Time in MOVE Operations . . . 46
3.5.5 Using Date and Time APIs . . . 47
3.5.6 Timestamp . . . 49
3.6 Example Using Import/Export Data Structure . . . 51
3.7 Example Using Pointers in RPG IV . . . 53

© Copyright IBM Corp. 1995

Chapter 4. Conversion Considerations 57
4.1 CVTRPGSRC Conversion Command and Parameters 58
4.1.1 CVTRPGSRC Parameters 59
4.1.2 /COPY Considerations 61
4.1.3 Conversion Problems 62
4.1.4 Scanning Tool for Migrated Source Code 64
4.2 Source Conversion Example 64
4.3 Creation Commands 68
4.3.1 Create RPG Module 68
4.3.2 Create Program 68
4.3.3 Create Bound RPG Program 69

Chapter 5. National Language Support with RPG IV 71
5.1 Recommended Usage of Characters in RPG IV 71
5.2 Source File CCSID Considerations 71
5.3 Externalizing Constants 72
5.4 Date Fields 73
5.5 Sort Sequence 73
5.6 Case Conversion 74
5.7 DBCS Graphic Data Type 75

Chapter 6. CL and ILE CL 77
6.1 ILE CL Functions 77
6.1.1 Changes to Existing Interfaces 77
6.1.2 CL Considerations with RPG IV in Compatibility Mode 78
6.1.3 ILE CL Considerations 78
6.1.4 The Call Bound Procedure Command 79
6.2 Changing Source Type from CL to CLLE 81
6.3 Should I Move CL to ILE CL? 81

Chapter 7. ILE Design Considerations 83
7.1 Overview of ILE Concepts 83
7.2 ILE Compile and Bind Commands 87
7.2.1 OPM Compatibility Mode 87
7.2.2 Comparison of Compile/Bind Commands 88
7.3 Activation Groups 89
7.3.1 Default activation group 89
7.3.2 User-Named Activation Group 90
7.3.3 Activation Group of Caller 90
7.3.4 System-Named Activation Group (*NEW) 90
7.3.5 Activation Group Recommendations 91
7.4 Differences Between Default and Non-Default Activation Groups 91
7.5 The Call Stack 92
7.6 Control Boundary 94
7.6.1 Control Boundary Example 95
7.7 ILE Static Call Syntax 96
7.8 Binding Considerations 96
7.8.1 Exports and Imports 97
7.8.2 RPG Initialization Considerations for an ILE *PGM or *SRVPGM . . . 99
7.8.3 Unresolved References 103
7.8.4 Service Program Signature 104
7.8.5 Service Program Recommendations 105
7.8.6 Updating Programs without Re-binding 105
7.9 Resource Scoping 106
7.9.1 Overrides and File Opens 106

7.9.2 Override Example . . . 108
7.10 Transparency . . . 110
7.11 Ending an ILE Program . . . 112
7.12 Ending an Application . . . 113
7.12.1 OPM RPG Application Example . . . 113
7.12.2 ILE RPG/400 Application Example . . . 114
7.12.3 Ways of Ending an ILE Application . . . 115
7.12.4 Use of RCLRSC . . . 121

Chapter 8. Development Environment . . . 123
8.1 Application Development Manager/400 . . . 123
8.1.2 Naming Conventions . . . 125
8.1.3 Relationships . . . 126
8.2 Introduction of the Walk-Through Scenarios . . . 127
8.2.2 Setup of the Application Development Manager/400 Environment . 129
8.2.3 Enhance the Mailing Application (Service Programs) . . . 132
8.2.4 Enhance a Service Program (Signature Implications) . . . 138
8.2.5 Import/Export Variables in ILE . . . 140
8.3 Use Binding Directories in Application Development Manager/400 . . . 147
8.4 How to Manage Without Application Development Manager/400 . . . 148
8.5 Copyright Your Software . . . 148

Chapter 9. Performance . . . 151
9.1 Compile Time . . . 151
9.1.1 Compile Options . . . 152
9.2 Program Object Size Comparisons . . . 153
9.2.1 Object Size Conversion Project . . . 154
9.3 Runtime Performance . . . 155
9.3.1 Working Memory Size for Runtime . . . 155
9.3.2 Choice of Tools . . . 155
9.3.3 Considerations . . . 156
9.4 Performance Benefits of ILE . . . 159

Chapter 10. Exception Handling . . . 161
10.1 What Is An Exception/Error? . . . 161
10.1.1 File Exceptions . . . 161
10.1.2 Program Exceptions . . . 162
10.2 Exception Handling Architecture . . . 162
10.2.1 Job Message Queues and Call Stacks . . . 163
10.2.2 Terminology . . . 164
10.2.3 Exception Messages . . . 167
10.2.4 Types of Exception Handlers . . . 167
10.2.5 Exception Handler Priority . . . 168
10.2.6 Default Actions for Unhandled Exceptions . . . 168
10.2.7 Handling an Exception . . . 169
10.2.8 Percolating an Exception . . . 169
10.2.9 Promoting an Exception . . . 170
10.3 Steps in Exception Handling . . . 170
10.3.1 Exception Handling Flow . . . 171
10.4 Comparing OPM and ILE Exception Handling . . . 174
10.4.1 Performance Impact . . . 175
10.4.2 ILE Condition Handler . . . 175

Appendix A. Diskette Install Instructions . . . 177

Appendix B. RPG IV Coding Examples 179
B.1.1 Using Pointers in RPG IV 179

Appendix C. Migration Information 185

Appendix D. Development environment example code 189
D.1.1 ADM Setup 189
D.1.2 Copy Build Options 191
D.1.3 Check out PARTL parts 193
D.2 Mailing List Application Description 194
D.3 Functional Scenario 194
D.3.1 Inquire into the Mailing List Master File 196
D.3.2 Maintain Mailing List Master File 197
D.3.3 Submit Mailing by Account Number 199
D.3.4 Submit Special Analysis Report 199
D.3.5 Query Mailing List File 200
D.4 Parts Structure 200

Index 201

Figures

1. H Specification Coded in RPG IV . . . 8
2. H Specification Coded in RPG/400 . . . 8
3. F Specifications Coded in RPG IV . . . 9
4. F Specifications Coded in RPG/400 . . . 10
5. Layout of RPG IV Definition (D) Specifications . . . 13
6. Example Using EXPORT and IMPORT Keywords . . . 17
7. Examples Using the Definition Statements - Field Definitions . . . 19
8. Examples Using the Definition Statements - Arrays . . . 19
9. Examples Using the Definition Statements - Data Structures . . . 20
10. RPG IV I Specification for Externally Described Files . . . 20
11. RPG IV I Specification: Record Layout . . . 21
12. RPG IV I Specification: Program Described File . . . 21
13. C Specification Coded in RPG IV . . . 23
14. C Specification Coded in RPG/400 . . . 24
15. O Specification Coded in RPG IV . . . 26
16. O Specification Coded in RPG/400 . . . 26
17. Example for ADDDUR Operation Code . . . 31
18. Example for SUBDUR Operation Code . . . 32
19. Example for EXTRCT Operation Code . . . 32
20. Example for TEST Operation Code . . . 33
21. Bound Procedure Call with Long Procedure Name . . . 33
22. Structured Programming in RPG/400 . . . 34
23. Structured Programming in RPG IV . . . 34
24. Comparing EVAL to RPG/400 Coding Style . . . 35
25. EVAL Operation Code . . . 36
26. The %ADDR Built-in Function . . . 38
27. The %ELEM Built-in Function . . . 39
28. The %SUBST Built-in Function . . . 39
29. The %SIZE Built-in Function . . . 39
30. The %TRIML Built-in Function . . . 40
31. Initializing Date and Time Data Type Fields . . . 42
32. Initializing a Date with Today′s Date . . . 43
33. Calculating with Date and Time Data Types . . . 44
34. Calculating with Date and Time Data Types . . . 44
35. Calculating with Date and Time Data Types . . . 45
36. ADDDUR, SUBDUR, EXTRCT and TEST Examples . . . 45
37. Date and Time Data Types in MOVE Operations . . . 46
38. Using Date and Time APIs . . . 48
39. Using Timestamp Data Type . . . 50
40. Using Timestamp Data Type . . . 51
41. Imported and Exported Data Structure . . . 52
42. Example for EXPORT/IMPORT: Exporting Procedure . . . 52
43. Example for EXPORT/IMPORT: Importing Procedure . . . 53
44. Pointers: Definition of Pointers in D-specs . . . 54
45. Pointers: Receive a Pointer . . . 54
46. Pointers: Define Module Information Array . . . 55
47. Pointers: Write Module List to a Database File . . . 55
48. Command Convert RPG Source. . . . 59
49. Conversion Example: Before . . . 65
50. Conversion Example: COPY Member . . . 66
51. Conversion Example: After . . . 67

© Copyright IBM Corp. 1995

52. CALLPRC Command Syntax 79
53. Bind by Copy and by Reference 84
54. DSPJOB - Activation Groups 85
55. Example - ILE RPG/400 Application 89
56. DSPJOB - Call Stack: Initial Screen 92
57. DSPJOB - Call Stack: Activation Groups 93
58. DSPJOB - Call Stack: Modules 94
59. Control Boundary Example 95
60. Export and Import Relationship Example 101
61. Override Example 109
62. Transparency Example 110
63. DSPJOB - Display File Overrides 111
64. Example - Ending an OPM RPG Application 114
65. Example - Ending an ILE RPG/400 Application 115
66. Example - Ending an ILE Application 120
67. Example of a Build Option for Documentation 126
68. Create Relations 126
69. Trigger Relations 127
70. Part Naming Rules 127
71. Mailing Application Menu 128
72. Project Structure 129
73. Building the Initial Application 130
74. Build Report Initial Application 130
75. Initial Program Structure 131
76. Program Structure Scenario-2 133
77. Working with Application Parts 133
78. Create a new Part Command 134
79. Subsetting Parts List 135
80. Example of Changing a BLDOPT of the CRTPGM Command 135
81. Display Call Stack Detail 136
82. Defining a User Option for Promote Part 137
83. Promoting and Archiving Parts 137
84. Program Structure Scenario-3 138
85. BNDSRC and BLDOPT Updates for Scenario-3 138
86. Relation Between Import/Export Variables and the Binding Language 141
87. Source Changes for MLGINQR 143
88. Source Changes for MLGLBLR2 143
89. Source Changes for MLGNAMR 144
90. Binding Source Changes for MLGSRV01 144
91. All Created Parts Using the PARTL 145
92. Promoting Parts 145
93. Content of the Promote Part List after the Promote 146
94. Error Handling Components for OPM and ILE 163
95. Job Message Queue/Call Stack example 164
96. Control Boundaries Due to Changing Activation Groups 165
97. Control Boundaries Within OPM Default Activation Group 166
98. Exception Flow in Call Stack 171
99. Exception Flow in Call Stack 171
100. Exception Flow in Call Stack 172
101. Exception Flow in Call Stack 173
102. Exception Flow in Call Stack 173
103. Mailing List Menu 196
104. Mailing List Inquiry Panel 196
105. Account Number Inquiry 196
106. Maintain Mailing List Master Panel 197

107. Change Mailing List Master Panel 197
108. Display GE Value 198
109. Name Search 198
110. Result JONES Search 198
111. Return with Account Number 199
112. Example Report 199
113. Example Report Zip Code 55920 199

Tables

1. H Specifications - Function Changed to Keywords 9
2. F Specifications - Functions Changed to Keywords 10
3. Keywords Supported in Definition Specification 14
4. Comparison for Externally Described Record Layout 21
5. Comparison for Externally Described Field Layout 21
6. Comparison for Program Described Record Layout 22
7. Comparison for Program Described Field Descriptions 22
8. Comparison of RPG IV and RPG/400 C Spec Layout 24
9. Renamed Operation Codes 29
10. Unary Operators 34
11. Binary Operators 35
12. Changes in Limits 37
13. External Formats for Date Data Type 41
14. External Formats for Time Data Type 41
15. Date Formats with Control Specification 42
16. Usable Combinations with DATEDIT for MOVE of *DATE and UDATE . . 47
17. Invariant Character Set 71
18. Comparison of ILE Compile and Bind Commands 88
19. Dynamic and Static Call Syntax 96
20. ILE Program Termination 112
21. Effect on Resources, Depending on the Way of Ending a Procedure at a Stack Level Closest to a Hard- or Soft Control Boundary 116
22. Storage Requirements for ILE programs 154
23. Size Ratio ILE to OPM programs 155

© Copyright IBM Corp. 1995

Special Notices

This publication is intended to help application fabricators to take advantage of the RPG language evolution and the Integrated Language Environment, an integral part of the Operating System/400 V3R1. The information in this publication is not intended as the specification of any programming interfaces that are provided by Operating System/400 or ILE RPG/400. See the PUBLICATIONS section of the IBM Programming Announcement for OS/400 V3R1 and ILE RPG/400 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

Application System/400	AS/400
C/400	COBOL/400
DATABASE 2 OS/400	DB2/400
Distributed Relational Database Architecture	DRDA
IBM	ILE
Integrated Language Environment	Operating System/400
OS/400	RPG/400

The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other trademarks are trademarks of their respective companies.

Preface

This document discusses implementation topics for users already familiar with AS/400 application development.

It contains an introduction to the RPG IV language and ILE CL, as well as a discussion of advanced topics related to the Integrated Language Environment.

This document is intended for experienced programmers, analysts, and implementers responsible for the creation and maintenance of AS/400 applications.

How This Document Is Organized

The document is organized as follows:

- Chapter 1, "Introduction to ILE RPG"

  Describes the benefits of ILE, lists the available high-level programming languages and application development tools.

- Chapter 2, "RPG Specification Sheets"

  Compares the new RPG language definition with it′s previous version, RPG/400.

- Chapter 3, "RPG IV Functions and Features"

  Describes the functional enhancements for the RPG IV language.

- Chapter 4, "Conversion Considerations"

  Discusses the aspects for migrating RPG/400 to RPG IV.

- Chapter 5, "National Language Support with RPG IV"

  Discusses the features available in RPG IV to internationalize your application.

- Chapter 6, "CL and ILE CL"

  Describes the extensions to the OS/400 command language support for ILE.

- Chapter 7, "ILE Design Considerations"

  Defines and describes the new concepts available to run ILE applications.

- Chapter 8, "Development Environment"

  Organizing application development projects has become increasingly complex. This chapter attempts to show our approach for an ILE AD project.

- Chapter 9, "Performance"

  Often, performance and its impact are secondary issues in a development project. This chapter addresses performance aspects during the build process and execution.

- Chapter 10, "Exception Handling"

  Robust applications need to handle unexpected conditions. This chapter describes the methods available with RPG IV and ILE.

© Copyright IBM Corp. 1995

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *Integrated Language Environment Concepts*, SC41-3606
- *ILE Application Development Example*, SC41-3602
- *ILE RPG/400 Reference*, SC09-1526
- *ILE RPG/400 Programmer's Guide*, SC09-1525
- *CL Programmer's Guide*, SC41-3721
- *CL Reference Guide*, SC41-3722
- *Data Management*, SC41-3710
- *Work Management*, SC41-3306
- *Application Development Manager/400 User's Guide*, SC09-1808
- *Application Development Manager Introduction and Planning Guide*, GC09-1807
- *Application Development Manager API Reference*, SC09-1809
- *International Application Development*, SC41-3603
- *System API Reference*, SC41-3801
- *ILE COBOL/400 Reference*, SC09-1523
- *ILE COBOL/400 Programmers' Guide*, SC09-1522
- *V31 Performance Capabilities Reference*, ZC41-8166

International Technical Support Organization Publications

- *System/36 to AS/400 Application Migration*, GG24-3250
- *AD/Cycle Code/400, ADM/400 and ADS/400*, GG24-3928
- *AS/400 ILE: A Practical Approach*, GG24-4148

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get listings of ITSO technical bulletins (Redbooks) online, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Technical Bulletins (Redbooks)

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

You may order individual books, CD-ROM collections, or customized sets, called GBOFs, which relate to specific functions of interest to you.

Acknowledgments

The advisor for this project was:

Klaus Subtil
International Technical Support Organization, Rochester Center

The authors of this document are:

Debbie Hatt
IBM UK

Jens Gert Andersen
IBM Denmark

Abraham Mizan
ISM South Africa

Ed van Weeren
IBM Netherlands

This publication is the result of a residency conducted at the International Technical Support Organization, Rochester Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Dick Bains
Carmen Hanson
Walt Madden
Joe Parks
Paul Remtema
Lee Walkky
Jing Wang
Scott Youngman
IBM Rochester Development Laboratory - USA

Susan Gantner
IBM AS/400 Competency Center, Rochester - USA

Phil Coulthard
Ted Fines
Dave Knott
Hans Koert

Jon Paris
Pramod Patel
IBM Toronto Development Laboratory - Canada

Michele Chilanti
Jim Cook
Ted Zonderland
International Technical Support Organization, Rochester Center

Chapter 1. Introduction to ILE RPG

The Integrated Language Environment introduced with OS/400 Version 2 Release 3 and the new RPG IV language definition in Version 3 Release 1 writes the next chapter on RPG and AS/400 programming.

RPG IV, as realized through the ILE RPG/400 compiler, joins the family of Integrated Language Environment (ILE) languages. ILE provides the ability to efficiently modularize applications. Applications are now built by using a collection of smaller modules. The benefits of ILE RPG/400 are realized with the ability to build, change and maintain code in smaller modules. When you build modular applications in ILE RPG/400, you compile and bind smaller functions. The increased compile time is offset by the fact that the modules being created are smaller.

1.1 Integrated Language Environment (ILE)

ILE is an architectural change to language compilers and the runtime characteristics of AS/400 programs. It is an extension to the architecture which means that your existing programs continue to run without changing and recompiling. ILE is available with Version 2 Release 3 of OS/400.

Integrated Language Environment is tightly integrated into the Operating System/400. The key benefits for the new ILE environment are:

- ***Language Integration***. Application programs are developed using the language mix best suited to perform each required function.
- ***Reusability***. Code from supported languages is divided into smaller, reusable, more logical modules that compile faster and require less maintenance over their life.
- ***Performance***. Capability is provided to optimize code in compute-intensive applications and to reduce the time to perform inter-program calls.

Integrated Language Environment increases developer productivity by providing the capability to divide code into smaller, more logical units that compile faster. The system binder combines the compiled modules to create the application program. In addition, the separation of compilation and bind steps provides more flexibility packaging the application.

The new source level debug tool that supports the ILE languages provides enhanced capability over the system debugger with the new feature to debug at the source or listing level of the program. Step, breakpoint, and conditional breakpoint functions have been provided. Expressions are entered and evaluated using the syntax of the programming language being debugged. The current system debug facility remains unchanged for programs developed outside ILE.

© Copyright IBM Corp. 1995

1.1.1 ILE Languages

1.1.1.1 Why ILE RPG/400 and RPG IV

Since IBM introduced the RPG language more than 20 years ago, the language constructs have been constantly enhanced and modernized. With the announcement of the Integrated Language Environment supporting the new RPG compiler, the language makes another evolutionary step.

The changes and enhancements are not intended to create a new language but to enhance the existing capabilities, to implement most of our customer requirements, and to position RPG for evolving programming techniques. For our customers and RPG programmers, this latest enhancement means an assurance for the future and that their investment in skills and application continues to be lucrative by giving them the capability to mix programming languages through the ILE system support.

The designers had to take into consideration the existence of thousands of RPG programmers that still need to be familiar and feel comfortable with the language. The big advantage to this approach is that experienced programmers can immediately start using the new functions changing existing programs and writing new ones.

The RPG IV language:

- Re-formats and simplifies the RPG specifications forms and at the same time adds new functions, most of them to answer customer requests.
- Removes some of the column-orientation limitations and introduces the capability of free-format arithmetic expressions and built-in functions.
- Lifts the RPG file, field and array name length restrictions.
- Expands and removes some of the language limitations, such as the number of files, and the number and size of arrays.
- Allows usage of uppercase and lowercase in symbolic names.
- Offers better support for date and time operations.
- Adds new functions and improves some of the existing functions.
- Prepares the languages for future enhancements and growth.
- Allows participation in the new Integrated Language Environment (ILE).

The AS/400 system offers a large number of RPG compilers in Version 3 Release 1. Integrated Language Environment RPG/400, more commonly known as RPG IV, is a new member in the RPG compiler family. RPG IV is the only one we can use to integrate RPG programs in the new ILE architecture. A separate chapter in this book describes how you can migrate RPG/400 programs to RPG IV.

When you order the RPG product, Integrated Language Environment RPG/400, you get a variety of different RPG compilers supporting all levels of the language.

- ILE RPG/400
  - for the RPG IV language definition
- RPG/400
  - for the RPG III definition

- S/36 Compatible RPG
  - for the RPG II definition
- Two compilers for previous releases
  - for S/36 and RPG/400

In addition, support for the S/38 environment is provided.

Note that there is no *PRV version for RPG IV

1.1.1.2 ILE COBOL/400

With V3R1, IBM also announced the ILE COBOL/400 programming language. Through ANSI-85 High functions of ILE COBOL/400, it is easier to port code to the AS/400 system from other platforms. Programmer productivity is increased with ILE COBOL/400, through its extensive database and workstation support, inter-language calls, interactive syntax checking, debug facilities, and a full complement of compile-time error diagnostics.

The ILE COBOL/400 product consists of the following COBOL components:

- ILE COBOL/400
- COBOL/400
- IBM System/36-Compatible COBOL
- IBM System/38-Compatible COBOL
- COBOL/400 Previous Compiler
- System/36-Compatible COBOL Previous Compiler

A number of new functions have been incorporated into ILE COBOL/400 that help increase user productivity. These new functions include:

- Variable Length Record support (RECORD IS VARYING Clause)
- EXTERNAL data items
- EXTERNAL files available to every program in the run unit
- Nested Source Programs
- INITIAL Clause to initialize on every call
- REPLACE statement to replace source program text
- DISPLAY WITH NO ADVANCING statement for cursor control
- ACCEPT FROM DAY-OF-WEEK statement
- Support for Nested Copy statements
- Enhancements to Extended ACCEPT and DISPLAY statements
- Procedure-pointer support

Additionally, new syntax has been added to COBOL on the CALL statement to differentiate between static and dynamic calls. Your investment in IBM SAA COBOL applications is protected by maintaining near-upward source compatibility in ILE COBOL/400. Furthermore, greater conformance to additional ANSI-85 High functions makes it easier for other IBM and non-IBM platform users to move their COBOL code to the AS/400 system. Differences are documented in the *ILE COBOL/400 Programmer's Guide* SC09-1522.

1.1.1.3 ILE C/400

C is a very popular language on most vendors′ systems and with most software engineers. The key to the continued success of the AS/400 system is the availability of new and state-of-the-art application solutions. Many application vendors have been reluctant to use C on the AS/400 system because of its performance history and capabilities. ILE C/400 overcomes these difficulties and opens the AS/400 system to the power and flexibility of the C language.

ILE C/400 high-performance, high-function, conformance to ANSI standards, and support of modular design helps software writers migrate their applications to the AS/400 system.

ILE C/400 addresses the requirements of application vendors and individual customers alike. The common runtime environment, improved inter-language communication, and binding support allow improved packaging of solutions. They reduce development effort through code reuse, enhance application maintenance and support, and improve integration, while allowing the solution to be written in the language of choice. Major areas of opportunity are:

- Non-C AS/400 customers
  - C/400 complements RPG/400 and COBOL/400 by providing better support for string and bit manipulation, numerical computation, floating-point data, dynamic memory allocation, and system programming functions.
- Current IBM C customers (non-AS/400)
  - Customers who have application software written in C on other platforms can migrate their application to the AS/400 system. ILE C/400 conforms to SAA C Level 2, and can now statically bind modules without any performance penalties or design restriction.
- Current C/400 Customers
  - The ILE C/400 compiler and its associated runtime form the basis for continued C application support for the AS/400 system and its successors. The C/400 compiler continues to be available in its present form; however, new customer requirements and enhancements in system support are implemented in ILE C/400.

Most third-party tools are written in C on competitive platforms. With the availability of ILE C/400, new opportunities are open for application vendors on the AS/400 system. Of particular importance to application vendors is the opportunity to sell *black box* routines for incorporation into a customer′s own programs.

1.1.2 Application Development Environment

Besides the ILE programming language compilers and the ILE system support, IBM offers a variety of tools that help you create your application, shorten the development cycle, maintain applications as an never-ending process and reduce future development cost.

Application Development Manager/400 (ADM) is such a tool developed by IBM for the AS/400 system as the change management and version control facility. Once you get familiar with ILE, you will ask the question ″How do I manage my application with all of these additional objects?″ To give you an idea how important it is to control development and maintenance, we have chosen to use ADM to build a scenario.

1.1.2.1 Impact Management

We also should look at the effects that the use of ILE has on the maintenance effort to use impact analysis methods and tools.

A product that is very integrated with ADM is *Application Dictionary Services/400* (ADS). It maintains a database of the relationships between parts of your applications, including programs, display files and database files. With the dictionary database in place, you can do impact analysis for planned application changes quickly. You can investigate a field in a database file to see how many other files and programs that reference that particular field need to be recompiled.

In Version 3 Release 1, ADS and ADM are packaged with the Application Development ToolSet product.

This product also provides the AS/400 host support for the two workstation-based development tools, CODE/400 and VRPG Client/2.

1.1.2.2 CODE/400 and VRPG Client/2

CODE/400 offers an interface from the workstation to the AS/400 system for listing, retrieving and replacing source, for launching host compiles, and for debugging applications. It provides productive, easy-to-use tools for edit, compile, debug, and screen and report painting. It fully supports all new ILE functions for CL and RPG IV including the RPG IV verifier that guarantees after using this ″local compile verification″, the first compile at the host always compiles without errors. It also supports a mixed debug environment for ILE and OPM programs with one debugger.

A companion product to CODE/400 is *VRPG Client/2* the latest offering for creating workstation applications that access AS/400 data and AS/400 programs using the RPG IV language definition. It allows easy transition from a character based to a graphical user interface using existing RPG skills.

Chapter 2. RPG Specification Sheets

Chapter Overview: This chapter describes the specifications and the changes that were made to enhance the RPG IV language definition.

2.1 RPG IV Specifications Statements

The first version of RPG was introduced by IBM almost 30 years ago based on the 80-column punch cards used at this time. Evolving over the years, the column orientation and limitations have been preserved, and RPG programmers have become familiar with it. Functions added over the years had to fit into the restrictive layout and sometimes led to misuse of certain functional specifications.

With RPG IV, new and often requested functions made a major redesign of the specification layouts necessary and opened the opportunity for modification. The intention of this chapter is to help you, as an experienced RPG programmer, quickly identify the changes and enhancements that have been made to this new language definition.

To allow a better program structure, the number of specification statements has been reduced to six. The layout for all of them has been modified to accommodate the increased length, and free-format logical and arithmetic expressions. Functional keywords replace column-specific special values, allowing easier language enhancements in future. For programmers maintaining code not written by themselves, keywords make the source easier, readable, and less difficult to maintain. A new (D) spec layout has been added. Extension specifications (E), and line counter specifications (L) statements have been removed from the new RPG.

RPG IV supports the following specifications in the listed order:

- Control (H) Specifications
- File Description (F) Specifications
- Definition (D) Specifications
- Input (I) Specifications
- Calculation (C) Specifications
- Output (O) Specification

Before we discuss the specification sheets in detail, you should be aware of the following modifications:

The source entry utility (SEU) has been enhanced to capture and check the new RPG IV Specification layouts. The SEU prompt facility (P?) allows you to display a list of the prompt formats.

The default name of the source file for RPG IV is QRPGLESRC. The length for the source statements is now 100 characters. When you create a source file for RPG IV using the command CRTSRCPF, the record length should be 112 characters; 12 for source sequence number and date fields, and 100 for the source statement field.

© Copyright IBM Corp. 1995

A new source type RPGLE is added for the RPG IV source member.

2.2 The Control (H) Specification Statement

The control specification provides the compiler with information about the program and the system. Other types of information you can include are:

- Name of the program
- Date and time formats for fields used in the program
- Use of alternate collating sequence or file translation

In the control specifications statement, keywords and values are used in a free format to specify the desired information. The positions used are from 7 to 80. Notice in the example in Figure 1, the enhanced H specification allows you to code multiple keywords on a single line, and you can use multiple H specification formats in the same source. Additionally, you can use data area RPGLEHSPEC in your library list to provide the same information.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H ALTSEQ(*EXT)  1  CURSYM('$')  2  DATEDIT(*MDY)  3
H DEBUG(*YES)   4
H TIMFMT(*ISO)  5  DATFMT(*MDY)  6
```

Figure 1. H Specification Coded in RPG IV

Figure 2 shows you equivalent source coded for RPG/400. The differences between the two coding styles are shown in the following example.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+.....
H.......1..CDYI....S.............1.F.............................Pgm-Id
H       1  $M/     S
```

Figure 2. H Specification Coded in RPG/400

The reference keys help you to identify the RPG IV keywords while you have to refer to the column position in the RPG/400 example.

1 The alternate collating sequence entry for database files in position 26 is now specified using the ALTSEQ keyword.

2 The currency symbol in position 18 is now represented by the CURSYM keyword.

3 The date edit function in column 20 has been altered to the DATEDIT keyword.

4 The debug entry in column 15 of the original H specification has been replaced by the DEBUG keyword.

5 The time format is an enhancement of RPG IV, and was not available in the prior RPG language.

6 The date format function moved from position 19 to the DATFMT keyword.

Table 1 summarizes the keywords supported in the RPG IV control specification and shows the equivalent position for RPG/400.

Table 1. H Specifications - Function Changed to Keywords

Keywords{(Value)}	**RPG/400 equivalent**	**Description**
ALTSEQ	Pos. 26	Alternate collating sequence
CURSYM	Pos. 18	Currency symbol
DATEDIT	Pos. 20	Date edit code
DATFMT	Pos. 19	Date format
DEBUG	Pos. 15	Debug option
DECEDIT	Pos. 21	Decimal notation
DFTNAME	Pos. 75-80	Program or module identification
FORMSALIGN	Pos. 41	Forms alignment
FTRANS	Pos. 43	File translation
TIMFMT	No equivalent	Time format for time data type

Since date and time external formats relate closely to data types, DATE and TIME and the duration operation codes, refer to 3.5, "Using Date and Time Formats and Operations" on page 40 for a contextual discussion and examples.

2.3 The File Description (F) Specification Statement

Each file used by your program, such as database, printer, or display file to name only the most common ones, and its attributes is identified by a corresponding file description specification. For RPG IV, the former limitation of 50 files per user program is raised, and now there is almost no limit.

The layout of this statement has been changed. A continuation line no longer exists. Keywords replace the continuation line. The Extension (E) and line (L) specifications forms have been removed. The Record Address File moved from E spec and is now a keyword in F spec. Forms length from L spec is also a keyword in F spec. Array definitions have been moved to the new definition (D) specification.

The differences between RPG IV and RPG/400 format layout and coding style are shown in Figure 3 and Figure 4 on page 10. In these examples, note that through the implementation of keywords and source text in mixed case, the code is less difficult to read and maintain.

To fully understand the examples, assume that FILE1 contains 4 record formats, FMT1 through FMT4, and that FMT2 contains 4 fields: CUSTNUM, ORDNUMB, ORDDATE, and TOTAMOUNT.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+.....
FFilename++IPEASFRlen+LKlen+AIDevice+.Functions++++++++++++++++++++++++++
FFile1     UF   E             Disk    Include(Fmt2)      1
F.....................................Functions++++++++++++++++++++++++++
F                                     COMMIT(OneCharFld) 2
```

Figure 3. F Specifications Coded in RPG IV

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+.....
FFilenameIPEAF....RlenLKlAIOvKlocEDevice+......KOptionEntry+A....U1......
FFILE1   UF  E                    DISK

F            FMT1                              KIGNORE 1
F            FMT3                              KIGNORE 1
F            FMT4                              KIGNORE 1
F                                              KCOMMIT 2
  :                        :                :
  :                        :                :
I              CUSTNUM                         CUSTNO 3
I              ORDNUMB                         ORDNO  3
I              ORDDATE                         ORDDAT 3
I              TOTAMOUNT                       TOTAMT 3
```

Figure 4. F Specifications Coded in RPG/400

1 Instead of using the IGNORE option multiple times in the RPG/400 example, you can now exploit the INCLUDE keyword to select the record format used in your program. The new COMMIT keyword **2** gives the capability to condition commitment control, and to set the parameter before opening the file.

3 These lines are eliminated in RPG IV since the increased length for field names does not require renaming.

If additional keywords are required, you can enter them on subsequent F specs starting from the Functions position.

Note:

Two different prompt layouts exist in SEU, one for *Program Described Files* (F), and one for *External Described Files* (FX).

Table 2 is a list of the keywords we can use in the File Description Specification statement.

Table 2 (Page 1 of 2). F Specifications - Functions Changed to Keywords

Keywords{(Value)}	RPG/400 equivalent	Description
COMMIT{(rpg_name)}	COMIT option on continuation line	Optional commitment control **Enhanced function**
DATFMT(format{separator})	n/a	Date format and optional separator **New function**
DEVID(fieldname)	ID option on continuation line	Name of the program device for last processed record
EXTIND(U1-U8)	U1-U8 in pos. 71-72	External file condition indicator
FORMLEN(number)	Pos. 18-19 on L-spec	Form length for PRINTER file
FORMOFL(number)	Pos. 20-22 on L-spec	The line number specified in the overflow line
IGNORE(recformat{:recformat...})	IGNORE option on continuation line	Ignores record format from externally described file
INCLUDE(recformat{:recformat...})	n/a	Opposite of IGNORE **New function**
INFDS(DSname)	INFDS option on continuation line	Name of file information data structure

Table 2 (Page 2 of 2). F Specifications - Functions Changed to Keywords		
Keywords{(Value)}	**RPG/400 equivalent**	**Description**
INFSR(SUBRname)	INFSR option on continuation line	File exception/error subroutine
KEYLOC(number)	Pos. 35-38	Key field starting location for program-described files
MAXDEV(*ONLY/*FILE)	NUM option	Maximum number of devices
OFLIND(indicator)	Pos. 33-34	Overflow indicator
PASS(*NOIND)	PASS option on continuation line	User controlled indicator area for program described WORKSTN file
PGMNAME(program-name)	Pos. 54-59	Program to handle the support of SPECIAL I/O devices
PLIST(Plist name)	PLIST option on continuation line	Parm list to be passed to the program specified in PGMNAME keyword
PREFIX(prefix_name)	n/a	The prefix_name is attached to all the fields of the external described file **New function**
PRTCTL(data_struct{:*COMPAT})	PRTCTL option on continuation line	Dynamic printer control **Enhanced function**
RAFDATA(filename)	Pos. 11-18 on E-spec	Record address file (RAF) name
RECNO(fieldname)	RECNO option on continuation line	For DISK files processed by relative-record number
RENAME(Ext_format:Int_format)	RENAME option on continuation line	To rename record formats of externally described files
SAVEDS(DS_name)	SAVDS option on continuation line	Names data structure to be saved and restored for each device
SAVEIND(number)	IND option on continuation line	Restores and saves indicators
SFILE(recfmt_name:RRN_fld)	SFILE option on continuation line	The first parameter is the name of the subfile record format. The second parameter is a field name that contains the relative record number of the subfile.
SLN(number)	SLN option on continuation line	The start line number determines where a record format is written to a display file.
TIMFMT(format{separator})	n/a	Default time format and optional time separator **New Function**
USROPN	UC in Pos 71-72	User controlled open of a file

The following sections describe the new and enhanced functions indicated in Table 2 on page 10 in more detail. For a detailed description of existing functions, please refer to publication *ILE RPG/400 Reference*.

COMMIT(Rpg_Name)

The enhancement for the commitment control option allows you to optionally activate commitment control for a file described in your program. If the one character Rpg_Name field contains a value of 1, the file is opened for commitment control. If Rpg_name is set to a different value, commitment control is not activated. The content of the field is explicitly set, or it is passed as a parameter to your program. This keyword has no effect on shared opened files. The parameter must be set prior to opening the file. Conditional commitment control allows applications to open files with

commitment control active under certain conditions, for example, random auditing or on customer request.

DATFMT(format{separator})

Specifies the default date format and an optional default date separator. On the file level, this keyword is typically used for program-described files.

TIMFMT(format{separator})

Specifies the default time format and an optional default time separator. On the file level, this keyword is typically used for program-described files.

PREFIX(prefix_name)

All field names in all record formats of the externally described file are prefixed with the value specified in the keyword argument. Using the fields in your program requires that those field names are coded with the prefix.

The total length of the name must not exceed the maximum length of an RPG field name.

PRTCTL(data_struct{:COMPAT})

The dynamic print control function has been moved from the F-spec continuation and the data_structure argument for the PRTCTL keyword has a new layout. The parameter *COMPAT indicates that you want to use the old style of the print control data structure. If *COMPAT is not used, the structure of the PRTCTL data structure is:

1-3 pos A character field that contains the space-before value

4-6 pos A character field that contains the space-after value

7-9 pos A character field that contains the skip-before value

10-12 pos A character field that contains the skip-after value

13-15 pos A numeric field that contains the current line value

INCLUDE

This keyword has the opposite effect of the IGNORE option. You can now specify the names of the record format of a file to be included in your program. All of the not listed record formats are ignored.

2.4 The Definition (D) Specification Statement

The definition specification format has no equivalent in the language definition prior to RPG IV. The definition specification consolidates and simplifies the definition of program variables. The declaration of data structures has been moved from the input specification; array definition from the eliminated extension (E) specification is now located in the D specification. Additional declaration facilities are for stand-alone fields, arrays within data structures, and dynamic arrays. It is located between the file description (F), and the input (I) specification.

In Figure 5 on page 13 the layout of the definition specification is shown.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
DField1           S            10  0 INZ
D  Field2         S20    INZ('September 06, 1994')
```

Figure 5. Layout of RPG IV Definition (D) Specifications

The following list is a brief discussion of the specification entries:

Name The name of a data item. This is a field, a data structure, a named constant, a data structure subfield, or an array. Please note the 15 character-wide space provided to allow indentation for structuring and better readability supported by mixed case source text.

E An 'E' in this position indicates that the data structure or the data structure subfield is externally described.

T 'S' means program status data structure, 'U' means data-area data structure.

Ds Positions 24-25 identify a data structure or the type of the variable. 'DS' describes a data structure, 'C' defines a constant and 'S' a stand-alone field or an array. Blank means none of the listed data items.

From From position. If this entry is blank, the To/L+++ field contains the length. The entry is used only for subfields in a data structure.

To/L This is the length of the field if the From position is blank. If the From position is not blank, this entry contains the To position value.

I Internal data type. Defines the characteristics of a field.

- **A** Character
- **G** Graphic character
- **T** Time
- **D** Date
- **Z** Timestamp
- **P** Packed decimal
- **B** Fixed binary
- **S** Zone
- ***** Pointer.

D Decimal positions. Up to 30 decimal positions.

Functions Keywords are used to define data and attributes.

The RPG IV language in V3R1 supports a set of keywords that is categorized as follows:

- Array/Table
- Data
- Data structure
- Date/time
- Initialization

- Named constants
- Pointers
- Storage

Please refer to Table 3 for a complete list of the supported keywords and the RPG/400 equivalent declarations.

Table 3 (Page 1 of 2). Keywords Supported in Definition Specification

RPG IV Keyword	RPG/400 equivalent	Description
ALT(array_name)	Pos. 46-57 on E-spec	Array/table in alternating format
ASCEND	A in pos. 45 on E-spec	Data in an array/table in ascending sequence
DESCEND	D in pos. 45 on E-spec	Data in an array/table in descending sequence
BASED(ptr)	n/a	The basing pointer (ptr) holds the storage location of the based data structure or stand-alone field. **New function**
CONST(literal)	Named constant on I-spec	Value for a named constant
CTDATA	Pos. 33-35 on E-spec	Array or table data is loaded at compile time.
DATFMT(format{separator})	n/a	Specifies the date format for a date data type field and optional separator. **New function**
DIM(numeric_constant)	Pos. 36-39 on E-spec	Maximum number of elements in an array or table
DTAARA(dtaaraname)	Pos. 21-30 on I-spec	External data area associated to the field, data-structure, data-structure subfield or data-area data-structure
EXPORT	n/a	The storage for the field is allocated in this module but may be used in other modules in this program. **New function**
EXTFLD(fldname)	Pos. 21-30 on I-spec	External name of a field in an externally described DS that is to be renamed.
EXTFMT(code)	Pos. 43 on E-spec	External data type for arrays **Enhanced function**
EXTNAME(file_name{:format_name})	Pos. 21-30 on I-spec	File name containing the definition of an externally described data structure
FROMFILE(file_name)	Pos. 11-18 on E-spec	Required for a pre-runtime array or table
IMPORT	n/a	The storage for the field is allocated in another module, but may be accessed in this module. **New function**
INZ{(constant)}	on I-specs	The field is initialized to the default values of the data type or to the specified constant.
LIKE(fld_name)	*LIKE DEFN	The attributes of the data being defined are taken from the variable fld_name.
NOOPT	n/a	No optimization performed for stand-alone field or data structure **New function**

Table 3 (Page 2 of 2). Keywords Supported in Definition Specification		
RPG IV Keyword	**RPG/400 equivalent**	**Description**
OCCURS(numeric_constant)	Pos. 44-47 on I-spec	Number of occurrences in a multiple occurrence data structure
OVERLAY(name{:pos})	On I-spec	A data structure subfield overlays the storage of the subfield specified in the argument of the keyword. **Enhanced function**
PACKEVEN		Zeros out the high order digit of an even packed subfield in a data structure
PERRCD(numeric_constant)	Pos. 33-35 on E-spec	Number of elements per record for a compile-time or pre-runtime array or table
PREFIX(prefix_name)	n/a	The name is prefixed to all subfields in an externally described data structure. **New function**
PROCPTR	n/a	Defines a pointer as a procedure pointer. Only allowed with data type * (pointer). **New function**
TIMFMT	n/a	Specifies the time format for a time data type field **New function**
TOFILE(file_name)	Pos. 19-26 on E-spec	Target file for pre-runtime and compile-time array or table

The following sections describe the new and enhanced functions indicated in Table 3 on page 14 in more detail. For a detailed description of existing function, please refer to publication *ILE RPG/400 Reference*.

BASED(PTR)

PTR is the name of a field that contains the address of a space in storage. The keyword is used for a field, a data structure or a runtime array. There is no need to define the PTR field. The function is used to access dynamically allocated storage. To set the point into the PTR field, we use the %ADDR built-in function. An example of how to use the BASED keyword is given in 3.7, “Example Using Pointers in RPG IV” on page 53 where we explain the built-in functions.

DATFMT(format{separator})

Use this keyword to define the format of a D (date) data type field and optionally the date separator character. See an example in Figure 7 on page 19. If you also initialize the field, the constant must be in the format you defined in the H specifications. The default format in H specs is *ISO. See Table 13 on page 41 for a list of date formats.

TIMFMT(format)

Use this keyword to define the format of a T (time) data type field and optionally the date separator character. See an example in Figure 7 on page 19. If you also initialize the field, the constant must be in the format you defined in the H specifications. The default format in H specs is *ISO. See Table 14 on page 41 for a list of time formats.

EXPORT/IMPORT

The keyword is used when binding multiple compilation units into an executable program. EXPORT allows you to make a data item, for example, variables or constants, known outside of the compilation unit (module) where it is defined. Through the IMPORT feature, other compilation units can access this data item without allocating the storage.

When binding multiple compilation units into a program, the exported field name must be unique within the program. Multiple modules can import an exported field. Program creation fails if no matching exported field name is found for an import. Characteristics of exported and imported fields should not be altered within the program. The program is created even if the characteristics are different, but you might have to deal with unpredictable results.

EXPORT or IMPORT keywords cannot be used for unnamed data structures.

An IMPORT field may not be initialized. Modules with EXPORT fields must be called first to allocate and set up the exported fields.

Compared to passing parameters, EXPORT/IMPORT has the advantage of accessing the data from more than one module within the same program.

Compared to Local Data Area (*LDA), EXPORT/IMPORT is faster.

Figure 6 on page 17 shows you an example for the EXPORT and import keywords. Imagine that both code fragments **1** and **2** are compiled separately and bound in to an executable program. The call bound (CALLB) **3** operation code is used to transfer control from *procedure* RPG1 to procedure SUBMOD. Field Custname1 is exported from module RPG1 and imported by module SUBMOD.

```
F*  This is the calling module  RPG1 1
F
FORDERFILE IF   E           K DISK
F
DCustname1        S             30    EXPORT
D            :               :
D            :               :
C                   DOW       (*IN01 = *OFF)
C                   READ      RF1                                    01
C                   IF        *IN01 = *OFF
C                   Eval      Custname1 = CUSTNAME
C         3 CALLB     'SUBMOD'
C                   ENDIF
C                   ENDDO
C                   EVAL      *INLR = *ON


D* This is the called module  SUBMOD 2
D
DCustname1        S             30    IMPORT
D            :               :
D            :               :
C                   MOVE      *BLANKS       Rpl_y            10
C
C     Rpl_y         DSPLY     '*EXT'        Custname1
C
C                   EVAL      *INLR = *ON
```

Figure 6. Example Using EXPORT and IMPORT Keywords

EXTFMT(code) The external data type definition for arrays has been enhanced to support date and time data types.

INZ{(CONSTANT)}

Use this keyword to initialize a stand-alone field, data structure, data structure subfield, or array. Without a constant, the data item is initialized to the default value for its data type. Using a constant initializes to the specified constant.

When initializing date or time type fields, the format of the literal must be according to the format specified in control specifications for date and time. If no format is declared in the control specifications, it defaults to *ISO, yyyy-mm-dd for date and hh.mm.ss for time. The literal should be defined as following:

- INZ(D'date value') for date
- INZ(T'time value') for time
- INZ(Z'timestamp value') for timestamp

A field, array, or data structure with keyword INZ cannot be used as a parameter in an *ENTRY PLIST.

OVERLAY(name{:pos})

The keyword is only allowed for data structure subfields. It overlays the storage of an already defined subfield on the same data structure. In parenthesis, you specify the subfield you want to overlay. Optionally, you can specify the starting position. If the starting position is not specified, the overlay starts from the first position. The length of the field is defined in To/l+++.

If the subfield is an array, OVERLAY applies to each element of the array.

PACKEVEN

The PACKEVEN keyword indicates 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. For a field or array element of length N, if the PACKEVEN keyword is not specified, the number of digits is 2N - 1; if the PACKEVEN keyword is specified, the number of digits is 2(N-1).

The keyword can improve performance in arithmetic operations replacing the null left-most digit with zero.

PREFIX(prefix_name)

Field names for externally described files are prefixed with the prefix_name for all fields in all records of the file.

Fields that are explicitly renamed on the Input specifications are not affected by this keyword.

The total length of the name after applying the prefix must not exceed the maximum length of an RPG field name.

PROCPTR

Defines a procedure pointer for a variable with * in position 40 for the internal data type.

NOOPT

No optimization is to be performed on the stand-alone field or data structure when this keyword is specified. This insures 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.

2.4.1 Examples for Declaring Data Items Using the Definition Specification

The source code fragment shown in Figure 7 on page 19 gives you some self-explanatory examples for the definition of stand-alone and constant fields.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++

D* Define a named constant whose value is the lower case alphabet
DLower           C                   CONST('abcdefghijklmnop-
D                                          qrstuvwxyz')

D* A named constant can also be defined without the keyword CONST
DUPPER           C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

D* Define a numeric field size 20 with 12 decimal positions:
DMonthTotal      S             20 12

D* Define the same field initialized to 0
DMonthTotal      S             20 12 INZ

D* Define a date field and initialize it to 19th of June 1994
DDelv_date       S               D   INZ(D'1994-06-19')
D                                    DATFMT(*YMD)

D* Define a time field
DDelv_time       S               T   INZ(T'12-00-00')

D* Define a field LIKE an existing one
DJanuary         S              5  2
DFebruary        S                   LIKE(January)
DTot_months      S             +2    LIKE(January)
DArmonths        S                   LIKE(January) DIM(12)
```

Figure 7. Examples Using the Definition Statements - Field Definitions

Figure 8 illustrates the coding for arrays. More examples are found in the publication *ILE RPG/400 Reference*.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D* Myarray has 20 elements, each is 5 digits long
DMyarray         S              5  0 DIM(20)
D
D
D* Array2 is a compile time array with 6 elements, each element
D*   10 char long, in ascending order, 3 elements per record
DArray2          S             10    DIM(6) CTDATA PERRCD(3)
D                                    ASCEND

         :                  :                 :
         :                  :                 :

*The specifications for the compile-time array Array2 are:

**CTDATA(Array2)
AAAAAAAAAABBBBBBBBBBCCCCCCCCCC
DDDDDDDDDDEEEEEEEEEEFFFFFFFFFF
```

Figure 8. Examples Using the Definition Statements - Arrays

Figure 9 on page 20 describes the coding for data structures. To define data structures, you can either use From and To/L, or you can only specify the length in the To/L. field. To split a data structure subfield, you can either use the OVERLAY keyword, or entries in From and To/L.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D* Define a data structure and initialize it  to zero and blanks
DItemNumber      DS                 INZ
D  Group                       5
D  Color                       6
D  Size                        3  0
D  Number                      6  0
D
D
D* Define a data structure with splitting subfields
D* State is part of CityState field
DAddress         DS
D  street                     15
D  Number                     10
D  CityState                  30
D  State                26     45
D
D
D* Define a data structure with array
DTotalyear       DS
D  Total                       7  2
D  Permonth                    5  2 DIM(12)
D
D
D* Define a data structure with OVERLAY
DWorkerDet       DS                 OCCUR(100)
D Name                        30
D  FirstName             1     15
D  LastName             16     30
D Phone_No                    10
D  Area_Code                   3    OVERLAY(Phone_No)
D  Local_No                    7    OVERLAY(Phone_No:4)
D Status                       1    inz('1')
```

Figure 9. Examples Using the Definition Statements - Data Structures

2.5 The Input (I) Specification Statement

The layout for the input specifications has been redesigned drastically and is now used for its original purpose: To describe files, record layouts, and fields. The definition for data structures, data structure subfields, and named constants has been moved to the new definition specification. After converting your source to RPG IV, this is where your declaration is found. Please refer to 2.4, “The Definition (D) Specification Statement” on page 12 for the definition of those data items. The following sections discuss externally and program-described file description.

2.5.1.1 Externally Described Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
 *Externally described record layout
IRcdname+++....Ri...................................................
I.............Ext-field+.................Field+++++++++L1M1..PlMnZr....
```

Figure 10. RPG IV I Specification for Externally Described Files

Layout changes for externally described files are required for the increased name length for record and field names. The record identification entry now allows you to enter 10 characters. Although the length of field names is now 10 characters, the field description entry provides space for 14 characters. This allows you to indent entries for better readability.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
 *Externally described record layout
IRcdname+++....Ri..................................................
I.............Ext-field+.................Field+++++++++L1M1..PlMnZr....
```

Figure 11. RPG IV I Specification: Record Layout

Table 4 and Table 5 summarize the layout changes of record and field description for externally described files:

Table 4. Comparison for Externally Described Record Layout

RPG IV	**RPG/400 equivalent**	**Description**
Pos. 7-16	Pos. 7-14	Record name
Pos. 17-20		Reserved
Pos. 21-22	Pos. 18-20	Record identifying indicator
Pos. 23-80		Reserved

Table 5. Comparison for Externally Described Field Layout

RPG IV	**RPG/400 equivalent**	**Description**
Pos. 7-20	Pos. 7-20	Reserved
Pos. 21-30	Pos. 21-30	External field name
Pos. 31-48		Reserved
Pos. 49-62	Pos. 53-58	RPG/400 field name, (optional in RPG IV)
Pos. 63-64	Pos. 59-60	Control level
Pos. 65-66	Pos. 61-62	Matching fields
Pos. 67-68		Reserved
69-74	65-70	Field indicators
75-80		Reserved

2.5.1.2 Program Described Files

Program described files are shown here only for completeness. We assume that most applications designed today would use externally described files. So you might want to skip this section.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
 *Program described record layout
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................

 *Program described field layout
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
```

Figure 12. RPG IV I Specification: Program Described File

For program described files, you can now define date, time, or timestamp type of fields, and specify the formats of them. See Table 13 on page 41 and Table 14 on page 41 for a list of external formats for date and time data types.

- Record identification entries
  - 10 positions for file name
  - 5 characters for record position identification code
- Field description entries
  - Date/time format. This entry is used to specify the external format for a date or time field. It is only used for program described files. See Table 13 on page 41 and Table 14 on page 41 for valid date and time formats.
  - Separator. This entry is used to specify the separator for a date or time field. It is also used for program described files.
  - Data format. New data type fields are added, (D) for date, (T) for time, (Z) for timestamp, (A) for character, (S) for zoned decimal, and (G) for graphic.

    You can still use blank for numeric and character fields.
  - 5 positions for FROM and TO entries
  - 2 positions for decimal positions
  - 14 positions for the field name. If the name is a normal input field, or the name of an entire array, it must follow the rules for a symbolic name. To refer to an element of an array, specify the name of the array, and enclosed the index in parenthesis.

Table 6 and Table 7 summarize the layout changes of program described records and fields.

Table 6. Comparison for Program Described Record Layout

RPG IV	RPG/400 equivalent	Description
Pos. 7-16	Pos. 7-14	File name as in file description specification
Pos. 16-18	Pos. 14-18	Logical relationship
Pos. 17-18	Pos. 15-16	Sequence
Pos. 19	Pos. 17	Number
Pos. 20	Pos. 18	Option
Pos. 21-22	Pos. 19-20	Record identifying indicator or **
Pos. 23-46	Pos. 14-18	Up to three record identification codes are entered. See the following:
23-27, 31-35, 39-43	21-24, 28-31, 35-38	Position
28, 36, 44	25, 32, 39	Not
29, 37, 45	26, 33, 40	Code part
30, 38, 46	27, 34, 41	Character

Table 7 (Page 1 of 2). Comparison for Program Described Field Descriptions

RPG IV	RPG/400 equivalent	Description
Pos. 7-30	Pos. 7-42	Reserved

Table 7 (Page 2 of 2). Comparison for Program Described Field Descriptions		
RPG IV	**RPG/400 equivalent**	**Description**
Pos. 31-34	n/a	Date/Time external format
Pos. 35	n/a	Date/Time separator
Pos. 36	Pos. 43	Data format
Pos. 37-46	Pos. 44-51	Field location, equivalent to From and To
Pos. 47-48	Pos. 52	Decimal position
Pos. 49-62	Pos. 53-58	Field name
63-64	59-60	Control level
65-66	61-62	Matching fields
67-68	63-64	Field record relation
69-74	65-70	Field indicator

2.6 The Calculation (C) Specification Statement

Most of the major enhancements in the RPG IV language definition take effect in the calculation specifications. The following list is a summary of the most important modifications:

- The space for factor 1, factor 2, and result field has been increased to 14 positions.
- Extended factor 2 has been designed to allow free format, and arithmetic and logical operations. The space provided by the extended factor 2 is 45 positions.
- There is only one conditioning indicator instead of three.
- The space for operation codes has been increased to 10 positions.
- Free format and built-in functions have been added.
- New operation codes and functions have been added.

Figure 13 and Figure 14 on page 24 compare the original with the enhanced RPG IV format and give a first example.

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C     MySalary      ADD       YourSalary    MySalary         10 2
C
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++..
C
C                   IF        *IN01 = *ON and Quantity > 0
```

Figure 13. C Specification Coded in RPG IV

```
CLON01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++....
C                    MOVE ARRY,2    ARRY,3
```

Figure 14. C Specification Coded in RPG/400

2.6.1 The New Calculation Specification Layout

The position of almost all of the columns has changed. Before the new layout for the C-specs is discussed in detail, Table 8 briefly summarizes and compares the different formats.

Table 8. Comparison of RPG IV and RPG/400 C Spec Layout

RPG/400 Position	RPG IV Position	Function
6	6	Format
7-8	7-8	Control level
9-17	9-11	Condition indicators are reduced to a single one. Multiple indicators are converted to multiple lines.
18-27	12-25	Length of factor 1 increased from 10 to 14 characters
28-32	26-35	Operation and extender length increased from 5 to 10 characters
33-42	36-49	Length of factor 2 increased from 10 to 14 characters
43-48	50-63	Length of the Result field increased from 6 to 14 characters
49-51	64-68	Field length field increased from 3 to 5 characters
52	69-70	Decimal position
53	n/a	Moved to the operation and extender column (26-35)
54-59	71-76	Result indicators
60-74	81-100	Comments

- **Conditioning indicator.** There is only one conditioning indicator instead of three. When you convert RPG/400 source members with more than one indicator on the same line, the conversion command creates additional line statements, one for every indicator.
- **Factor one.** The space for factor 1 is now 14 positions.
- **Operation code.** The space for operation codes has been increased to 10 positions. Some of the existing operation codes are renamed to improve readability and better understanding of the program code. All of the renamed operation codes are listed in Table 9 on page 29.

  Conversion from RPG/400 code to RPG IV renames these operation codes to the new names. The RPG IV compiler and SEU do not support the previous syntax when the source member is RPGLE.
- **Operation Extender.** The operation extenders provide additional attributes to some of the operations. The extenders are specified in parenthesis following the operation code, for example, MULT(H) or MULT (H). Valid extenders are:

  Entry **Explanation**

Blank No operation extension

(H) Half adjust. It indicates that the result of an arithmetic operation is to be rounded. It is used with arithmetic operations, but not with MVR (move remainder), or DIV followed by MVR.

(N) Record is read but not locked. It is used for database files defined as read-for-update. Using the extender (N) in read operations such as READ, READE, READP, READPE, and CHAIN does not lock the record. If the extender is not used, the record is locked until update, or next read from the same file.

(P) Pad operation. A (P) defined in operations such as CAT, SUBST, MOVEA, MOVEL, or XLATE indicates that the result of these operations is padded with blanks on the right of the string if the result field is longer than the result of the operation. For MOVE operations, padding is done from the left.

(D), (T), (Z) These extenders are used to test a character or numeric field for valid date, time, or timestamp values. It is used with the new TEST operation code. The field we want to test is not a field defined as date, time, or timestamp type.

- **Factor 2.** The space for factor 2 is now 14 positions.
- **Extended Factor 2.** The extended factor 2 is used with the new operation codes DOU (Do until), DOW (Do while), EVAL (Evaluation), IF (If/Then), and WHEN (When true then select). The space for extended factor 2 is 45 characters, and allows free format coding and expressions such as:
  - EVAL Amount = Quantity * Price
  - DOW *IN90 = *ON AND Quantity > 0

  If there is not enough space for the expression in one extended factor 2, you can continue on the extended factor 2 on the next line. In case you split the name of the field, you can terminate the first line using an hyphen (-) or plus (+).

  You can find further details on the new operation codes and examples in chapter 3.1, "Operation Codes" on page 29.
- **Result Field.** The space for the Result field is now 14 positions.
- **Field length.** The size of this entry has been increased to 5 characters to allow the new field length of 32,767 for character fields.
- **Decimal position.** The size of this entry has been increased to 2 characters to allow up to 30 decimal positions in a numeric field.

2.7 The Output (O) Specification Statement

The output specification layout has been changed. The positions of some of the column specifications have been increased to include the new name size and the new field length. Compare Figure 15 on page 26, and Figure 16 on page 26 for the differences between RPG/400 and RPG IV:

```
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
OQSYSPRT   T    LR                     1  1
O                       INSTAL              21
O                                           24 'OF'
```

Figure 15. O Specification Coded in RPG IV

```
OName++++DFBASbSaN01N02N03Excnam.......................................
OQSYSPRT T  1 1   LR
O                         INSTAL    21
O                                   24 'OF'
```

Figure 16. O Specification Coded in RPG/400

The following sections describe the changes in the output specification statement.

2.7.1.1 Record Identification and Control Entries

- 10 positions for file name entry.
- 10 positions for EXCEPT name entry.
- 20 positions for comments. This a new entry added in RPG IV.
- Space lines before and after printing a line. The entries have been increased from one position to three. The maximum spaces you can now specify to advance before and after printing a line is from 0 to 255.
- Skip to line before and after printing a line. The entries have been increased from two positions to three. The maximum line number you can now specify to skip before and after printing a line is from 1 to 255.

2.7.1.2 Record Identification for Addition/Deletion

- 10 positions for file name entry.
- 10 positions for EXCEPT name entry.
- 20 positions for comments. This is a new entry added in RPG IV.

2.7.1.3 Field Description and Control Entries

- 14 positions for field name entry.
- The end position entry has been increased to 5 in order to capture the end position of fields in records with up to 32767 characters.
- Data format.

  New data type of fields are added. (D) for date field, (T) for time field, (Z) for timestamp field, (A) for character field. (S) for zone decimal field, and (G) for graphic field. Remember, the entry in this position specifies the format of the data in the record in the file. The entry has no effect on the format used for internal processing of the specified field in the program.

- 28 positions for constant/edit word entry.

  In addition to constants, edit words, and format names, we can now specify the external format for date and time fields.

- 20 positions for comments. This a new entry added in RPG IV.

2.8 The File Extension (E) Specification Statement

The extension specification has been eliminated in RPG IV. Prior to RPG IV it was used to describe:

- Record address files

  This function is now implemented with keyword RAFDATA in the file specifications.

- Arrays and tables

  Those data items are now described in the definition specifications.

2.9 The Line Counter (L) Specification Statement

The line counter specification has been eliminated and the function is now implemented with keywords in the file specification.

Chapter 3. RPG IV Functions and Features

Chapter Overview: While the previous chapter focused primarily on the layout changes for RPG specification sheets, this chapter discusses the functional enhancements introduced with the RPG IV language definition that include:

- Expanded naming capabilities
- Mixed-case source code (not case-sensitive)
- Built-in functions
- Format-free arithmetic and logical expressions
- New and renamed operation codes
- Support for new data types
- Raised and removed limits

Most of the enhancements are illustrated with comprehensive examples. Some features, for example, the new date and time support, are discussed in more detail than other more obvious and easy-to-use functions. For more thorough information, please refer to the publications *ILE RPG/400 Reference*, SC09-1526, and the *ILE RPG/400 Programmer's Guide*, SC09-1525.

3.1 Operation Codes

You can classify the changes to operation codes in four different categories. These operation codes are:

- Renamed
- New and process date and time data types
- New and perform a static call
- New and support your efforts for structured programming

3.1.1 Renamed Operation Codes

Due to the expanded space for operation codes, some of the existing codes have been renamed to increase readability. RPG IV does not support the previous syntax of these operation codes. The functionality of these operation codes remains unchanged. The conversion command CVTRPGSRC converts them automatically. Table 9 gives you a complete list of the operation codes that are renamed:

Table 9 (Page 1 of 2). Renamed Operation Codes

RPG/400 Operation Code	**RPG IV Operation Code**
BITOF	BITOFF
CHEKR	CHECKR
COMIT	COMMIT
DEFN	DEFINE
DELET	DELETE
EXCPT	EXCEPT
LOKUP	LOOKUP

© Copyright IBM Corp. 1995

Table 9 (Page 2 of 2). Renamed Operation Codes	
RPG/400 Operation Code	**RPG IV Operation Code**
OCUR	OCCUR
REDPE	READPE
RETRN	RETURN
SELEC	SELECT
SETOF	SETOFF
UNLCK	UNLOCK
UPDAT	UPDATE
WHxx	WHENxx

The characteristics of the renamed operation codes remain unchanged.

3.1.2 New Operation Codes to Process Date and Time Data Types

Duration codes are used in operations such as: add duration (ADDDUR), subtract duration (SUBDUR), and extract (EXTRCT) to specify from/to which part of the date, time, or timestamp field a value is added, subtracted, or extracted. They are specified in the factor 2 and result field. See examples in Figure 36 on page 45.

You can express time periods by using the duration codes

- *YEARS or *Y
- *MONTHS or *M
- *DAYS or *D
- *HOURS or *H
- *MINUTES or *MN
- *SECONDS or *S
- *MSECONDS or *MS

in combination with the following operations codes:

ADDDUR The ADDDUR operation adds the duration specified in factor 2 to the field or constant specified in factor 1 and places the resulting date, time, or timestamp in the Result field.

Factor 1 is optional and may contain a date, time, or timestamp field. It can also contain a field, array element, or data structure subfield. In this case, the data type of the field must be the same data type as the field specified in the result field. If factor 1 is not specified, the duration is added to the field specified in the result field.

Factor 2 is required and consists of two parts. The first part is a numeric field or constant with zero decimal positions. If the value is negative, the duration is subtracted. The second part, separated by a colon, is a duration code consistent with the result field data type.

The result field must be a date, time, or timestamp field, array, or array element.

```
CLON01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C     OrderDate     ADDDUR    23:*Days      ShipDate
C                   ADDDUR    12:*Hours     StartTime
```

Figure 17. Example for ADDDUR Operation Code

SUBDUR The SUBDUR provides two operations:

1. Subtract a duration to establish a new date, time, or timestamp
2. Calculate a duration

Subtract a duration: The SUBDUR subtracts a duration specified in factor 2 from a field or constant specified in factor 1 and places the resulting date, time, or timestamp in a field specified in the result field.

Factor 1 is optional and may contain a date, time, or timestamp field. It can also contain a field, array element, or constant. In this case, the data type of the field must be the same data type as the field specified in the result field. If factor 1 is not specified, the duration is subtracted from the field specified in the result field.

Factor 2 is required and consists of two parts. The first part is a numeric field or constant with zero decimal positions. If the value is negative, the duration is added. The second part, separated by colon, is a duration code consistent with the result field data type.

The result field must be a date, time, or timestamp field, array, or array element.

Calculate duration: The SUBDUR can also be used to calculate a duration between:

1. Two dates
2. A date and a timestamp
3. Two times
4. A time and a timestamp
5. Two timestamps

Both factor 1 and 2 are required and must contain date, time, or timestamp field types, according to the rules specified above.

The result field contains two sub-factors. The first part is the name of a numeric field, array, or array element with zero decimal positions. The second part, separated by a colon, is a duration code denoting the type of the duration.

Please note that calculating a duration, using the SUBDUR operation, the result is in years, days, months, hours, minutes, seconds, or microseconds and not in date, time, or timestamp.

```
CLON01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C*Determine the OrderDate which was XX years, YY months, ZZ days
C*prior to DelDate
C
C     DelDate       SUBDUR    xx:*Years     OrderDate
C                   SUBDUR    yy:*Months    OrderDate
C                   SUBDUR    zz:*Days      OrderDate
C
C*Calculate the number of days between OrderDate and DelDate
C
C     DelDate       SUBDUR    OrderDate     Num_days:*D
```

Figure 18. Example for SUBDUR Operation Code

Note:

EXTRCT Extracts year, month, day, hours, minutes, seconds, or microseconds of a date, time, or timestamp data type field and places it into a field specified in Result. This facilitates the isolation of the year, month, day, hours, minutes, seconds or microseconds portion of a field.

Factor 2 contains two sub-factors. The first part is a date, time, or timestamp field, array, or array element. The second part, separated by a colon, is a duration code. The duration code must be consistent with the field type in the first sub-factor.

The result field is a numeric or character field, array, or array element.

```
CLON01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C*Extract the year from a date field
C
C                   EXTRCT    BrtDate:*Y    BrtYear
C
C*Extract the hour from a timestamp field
C
C                   EXTRCT    LogonTime:*H  Loghour
```

Figure 19. Example for EXTRCT Operation Code

TEST The TEST checks if the content of a field is a valid date, time, or timestamp.

Factor 1 contains a date or time format if the field to test in the result field is a character or numeric field. If not specified, the format is taken from the format (DATFMT) specified on the Control specification. You should also specify an operation extender (D), (T), or (Z).

If the field to test on the result field is a date, time, or timestamp type, no operation extender is used. Factor 1 must be blank.

An indicator is required in positions 73-74 and is set to on if the value in the result field is not valid.

```
CLON01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C*  DATE1 is a numeric field and contains '911014'
C*  Indicator 40 is set ON since DATE1 is not *DMY
C
C     *DMY          TEST(D)                DATE1                 40
C
```

Figure 20. Example for TEST Operation Code

3.1.3 New Operation Code for Static Call

CALLB CALLB is a new operation code in RPG IV. It has been added to perform within the ILE environment. CALLB calls a procedure bound in an ILE program. The operation code is used in an ILE RPG module to call a procedure (module) bound on the same ILE program. The difference between CALL and CALLB is that the CALL calls another program, and the CALLB calls a module within a program. The procedure to call can be written in any ILE language. Factor 2 is required, and must be a literal or a named constant containing the name of the procedure or a procedure pointer containing the address of a procedure to be called. The name of the procedure can be up to 255 characters if this is not the name of a procedure of an RPG module. The result field is optional and may contain the name of a PLIST.

You can specify an indicator in 75-76. This indicator is set to on if the call ends with LR set on.

As all the modules of a program are bound together, calling a procedure of a module is faster than to call a program using the CALL operation command.

Note:

In order to use the bound procedure call with other ILE languages allowing procedure names longer than 10 characters such as C/400, you might want to consider the example illustrated in Figure 21.

```
D*ame+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
D Longname        C                   CONST('Long_C_Procedure_Name')
...
C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C                   CallB     LongName

C                   CallB     'Medium_Name'
```

Figure 21. Bound Procedure Call with Long Procedure Name

3.1.4 New Operation Codes for Structured Programming

The redesigned calculation specification for extended factor 2 allows a syntax change for the operation codes for structuring your program. The arithmetic expressions in extended factor 2 gives you a more intuitive way to code *do loops* and *case* statements as illustrated in Figure 23 compared to the previous coding shown in Figure 22:

```
CLON01N02N03Factor1+++OpcdeFactor2+++ResultLenDEHiLoEqComments+++++++...
C           *IN03     DOWEQ*OFF
C           STATUS    IFEQ 'S'
C           QTY       ANDLE1000
...
C                     ENDIF
C                     ENDDO
```

Figure 22. Structured Programming in RPG/400

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C
C                   DOW       *IN03 = *OFF
C                   IF        (Status = 'S') AND Quantity < 1000
...
C                   ENDIF
C                   ENDDO
```

Figure 23. Structured Programming in RPG IV

The new operation codes in this category are:

- DOU
- DOW
- IF
- WHEN
- EVAL result = expression

The operation codes DOU (Do until), DOW (Do while), IF(If/Then), and WHEN (when true then select), are similar to DOUxx, DOWxx, IFxx, and WHENxx. Instead of using factor 1 and factor 2, you can now use the extended factor 2 to describe the logical expression. The expression can contain more than one condition concatenated by relational operators. Factor 1 is not used.

The unary and binary operators supported in arithmetic and logical expressions are:

Table 10. Unary Operators

Description	Operator
Positive value	+
Negative value	-
Negation of indicators	NOT

Table 11. Binary Operators

Description	Operator
Addition, String Concatenation	+
Subtraction	-
Multiplication	*
Exponentiation	**
Division	/
Remainder	//
Equal	=
Greater than or equal	> =
Greater than	>
Less than or equal	< =
Less than	<
Not equal	< >
Logical and	AND
Logical or	OR

3.1.4.1 EVAL Operation Code

EVAL is a new operation code also used in extended factor 2 to code expressions such as: Result = Expression. The expression is evaluated, and the result placed in result. Therefore the result cannot be a constant and must be the name of field, array, array element or data structure subfield. The expression must match the result field type and is numeric, character, relational, or logical.

Figure 24 and Figure 25 on page 36 compare two code fragments written in RPG/400 and RPG IV. The RPG IV example is using the new EVAL operation code while the RPG/400 is coded in the traditional way. As you see, the code in RPG IV is more readable.

```
CLON01N02N03Factor1+++OpcdeFactor2+++ResultLenDEHiLoEqComments+++++++...
C           STATUS    IFEQ 'S'
C           QTY       ANDLE1000
C           QTY       MULT PRICE     AMOUNT  72H
C                     ELSE
C           PRICE     MULT 0.10      PRICED  52H
C           QTY       MULT PRICED    AMOUNT    H
C                     ENDIF
```

Figure 24. Comparing EVAL to RPG/400 Coding Style

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++
C
C                   IF        (Status = 'S') AND Quantity < 1000
C                   EVAL(H)   Amount = Quantity * Price
C                   ELSE
C                   EVAL(H)   Amount = (Price * 0.10) * Quantity
C                   ENDIF
```

Figure 25. EVAL Operation Code

3.2 Symbolic Names

Names for data items used in your program such as variables, constants, and data structures are called *symbolic names*. They identify uniquely a specific data item.

3.2.1 Upper/Lowercase

You can now mix lower and uppercase characters in the names of files, fields, record formats, arrays, data structures, labels, and in any symbolic names used to identify and access specific data in the program. This is also true for operation codes and reserved words. The SEU and RPG IV compiler accepts everything in upper or lowercase. The compiler translates all the characters to uppercase. In the list produced by the compiler, all the characters appear as they were written by the programmer (in upper and lowercase). In the cross reference list, all the names are in uppercase.

The compiler does not translate the following:

- Literals
- Comments
- Compile-time array/table data
- Currency symbol
- Date/time edit values on H spec
- Date/time separator on I and O specs
- Decimal edit value on H spec
- Comparison characters on Record ID entries of I spec

Note: Although the characters $, #, and @ are allowed for names, you should avoid using them. The reason for this is that in some countries, these characters are shown as uppercase national characters. For example, in Denmark and Norway, the $ is shown as the character Å (its lower case equivalent is å), # is Æ (lower case æ), and @ is Ø (lower case ø). In Spain, the # is Ñ, which has ñ as the lower case equivalent.

A Danish programmer would expect that all names could be entered in lowercase Danish. A name such as *SÆLGER* is a valid field or file name for RPG IV, but it cannot be entered as *sælger*. The compiler only translates the letters a-z to uppercase.

3.2.2 Name Length

The new RPG IV supports symbolic names from 1 to 10 characters long. That includes file, field, record format, data structure and array names. The programmer does not need to rename record formats to 8, or field names to 6 characters just because the definition in DDS is longer. The factor 1, factor 2, and results field have also been increased to 14 characters. This allows the usage of built-in functions and arrays with names of up to 10 characters and enough space for the index.

3.2.3 Underscore

You can use underscore symbol (_) on the symbolic name except as the first character.

3.2.4 Blank Lines

Totally blank lines are now allowed between statements. This makes the source more readable and allows it to indicate logical program portions.

3.2.5 Examples

The following examples show some of the new functions.

- CusNumber = CUSNUMBER = cusnumber
- Ord_Number = ORD_NUMBER = ord_number
- MyArray,idx
- MySalary <> Mysalar

In the last example, the field names refer to two different variables because they do not contain the same number of characters.

3.3 Changes in Limits

One of the reasons for redefining the RPG language lay in the limitations built into the previous compiler. With the ILE system support, restrictions and limitations could be removed. Table 12 summarizes the raised limits.

Table 12 (Page 1 of 2). Changes in Limits

Description	RPG/400	RPG IV
Field/Array Name	6	10
Data Structure Name	6	10
Record Format Name	8	10
File Name	8	10
Number of Files	50	None
Character Field Size	256	32767
Constant Size	256	1024
Data Structure Size	9999	32767
Number of Decimal Places	9	30
Number of Array Elements	9999	32767

Table 12 (Page 2 of 2). Changes in Limits		
Description	**RPG/400**	**RPG IV**
Named Constant	256	1024
Number of Subroutines	256	None
Size of Program	Varies	None
Note: The length for the file name in the INFDS (INFormation DataStructure) remains 8 characters.		

3.4 Built-in Functions in RPG IV

%ADDR(variable) Places the address of an item into a variable. The item is a field, an element of array, or an expression.

%ADDR built-in function is also used in the Definition specification with an * data type field (pointer). The size of the * data type field is always 16, and you do not have to specify it. When you use a field in calculation to contain the address of an item, you do not have to define the size of this field.

Figure 26 shows an example of using %ADDR built-in function.

:

```
D Pointer1        S              *   INZ(%ADDR(Field1))  1
D Field1          S             10   INZ('ABCD')
D Field2          S             10   BASED(Pointer2)     2
D Result          S             10
              :                 :               :
              :                 :               :
C                   EVAL      Pointer2 = %ADDR(Field1)   3
C
C                   IF        Field1 = Field2
C                   MOVE      'Yes'         Result
C                   ELSE
C                   MOVE      'No'          Result
C                   ENDIF
C                   DSPLY     '*EXT'        Result
C                   MOVE      *ON           *INLR
```

Figure 26. The %ADDR Built-in Function

The following explanations refer to the previous example:

- **1** Pointer1 is defined as a pointer field (*) and initialized with the address of the Field1 field.
- **2** Field2 is a 10 character field and its address is based on the content of the field Pointer2.
- **3** Using the EVAL operation, we specify that Pointer2 contains the address of Field1. But Field2 is based on the address in Pointer2. This means that Field1 and Field2 have the same address, and as a result, the same content.

%ELEM(array_name) or

%ELEM(multiple_occurrence_data_structure_name Places into a variable the number of elements of an array, or the number of occurrences of a multi-occurrence data structure.

Figure 27 is an example of using %ELEM in an array.

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C                   dou       Index = %ELEM(Name_Array)
C                    :           :
C                   eval      INDEX = Index + 1
C                   endDO
```

Figure 27. The %ELEM Built-in Function

%SUBST(string:start{:length}) Extracts a portion of a string. Start represents the starting position of the substring. The length is optional and defines the length of the substring. If not specified, the length of the substring is from the start position to the end of the string. %SUBST can also be used in a definition of a field in Definition specification.

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C                   EVAL      Comment = 'It''s a great language'
C                   EVAL      Fcomment = %SUBST(Comment:8)
C*   Fcomment = 'great language'
C
```

Figure 28. The %SUBST Built-in Function

%SIZE(name{:*ALL}) Places into a numeric variable the size of a field, literal, array, data structure, or named constant. If the name is an array or a multiple occurrence data structure, the variable contains the size of one element or one occurrence. If *ALL is specified, the size received is the size of all the elements or occurrences. %SIZE can also be used in Definition specification.

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C                   EVAL      NumFld = %SIZE(Field1)
C                   EVAL      NumFld = %SIZE(1994)
C                   EVAL      NumFld = %size(Array1:*ALL)
C
```

Figure 29. The %SIZE Built-in Function

%TRIM(string) Returns the string cleared of leading and trailing blanks

%TRIML(string)

Returns the string cleared of leading blanks

%TRIMR(string) Returns the string cleared of trailing blanks

```
CLON01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C                   EVAL      Astring = '     leading blanks'
C                   EVAL      Fstring = %TRIML(Astring) + ' removed'
C*   Fstring = 'leading blanks removed'
C
```

Figure 30. The %TRIML Built-in Function

In this example, the built-in %TRIML function moves the string to the left, clearing all the leading blanks. EVAL statement adds the literal 'removed' to the result of %TRIML and the content of Fstring is: 'leading blanks removed'. Astring and Fstring fields should be defined in D or C specifications.

When %TRIM, %TRIML, or %TRIMR are used in Definition specification, the string parameter must be a literal or a named constant.

Further Reading

See the *ILE RPG/400 Reference manual*, SC09-1526, for more information about the calculation specification layouts, the new operation codes, duration codes, operation extenders, and the new built-in functions.

3.5 Using Date and Time Formats and Operations

RPG IV now fully supports the date and time data types that were introduced in OS/400 V2R1.1 together with Distributed Relational Database Architecture* (DRDA). In addition, operation codes as described in 3.1.2, "New Operation Codes to Process Date and Time Data Types" on page 30 are available for date and time calculation using the date and time data types as well as testing for valid content.

The new operation codes for date and time calculations are very powerful, and make date and time calculations much easier than previous RPG support. In current applications, subroutines or separate programs may be used to handle calculation of dates, such as adding or subtracting a number of days to or from a date, or validating the content.

Many of these routines or programs could depend on a specific method for defining and storing fields containing dates. The format used may contain two digits for year only, it may contain four digits for the year, or maybe a century digit, where a zero defines 20th century and a one defines 21th century.

The new support in RPG IV allows you to define three new data types:

- Date data type - field type D
- Time data type - field type T
- Timestamp data type - field type Z

In DB2 for OS/400, these fields types could be defined since V2R1.1, and note that the date data type is a type L field in DDS. Prior to RPG IV, character fields had to be used to read and write those data types, and it was the programmer's responsibility to ensure correct content.

The new RPG IV field types are not numeric or character fields. They are date fields, time fields, and timestamp fields, and you can only use certain operation codes with date, time, and timestamp fields.

The compiler tests to see if the content of a date or time field is valid. If it is not valid, an error message is generated.

Table 13 and Table 14 describe the different date and time formats we can use in RPG IV. The same reserved words are used in definition (D), input (I), calculation (C), and output (O) specifications for date and time type fields. We discuss later in this chapter the rules and overrides for date and time definitions.

Table 13. External Formats for Date Data Type

Name	Description	Format	Sep.	Length	Example
*MDY	Month/Day/Year	mm/dd/yy	/-.,&	8	01/15/91
*DMY	Day/Month/Year	dd/mm/yy	/-.,&	8	15/01/91
*YMD	Year/Month/Day	yy/mm/dd	/-.,&	8	91/01/15
*JUL	Julian	yy/ddd	/-.,&	6	91/015
*ISO	International Standards Org.	yyyy-mm-dd	-	10	1991-01-15
*USA	IBM USA Standard	mm/dd/yyyy	/	10	01/15/1991
*EUR	IBM European Standard	dd.mm.yyyy	.	10	15.01.1991
*JIS	Japanese Industrial Standard	yyyy-mm-dd	-	10	1991-01-15

Table 14. External Formats for Time Data Type

Name	Description	Format	Sep.	Length	Example
*HMS	Hours:Minutes:Seconds	hh:mm:ss	:.,&	8	14:00:00
*ISO	International Standards Org.	hh.mm.ss	.	8	14.00.00
*USA	IBM USA Standard	hh:mm AM	:	8	02:00 PM
*EUR	IBM European Standard	hh.mm.ss	.	8	14.00.00
*JIS	Japanese Industrial Standard	hh:mm:ss	8	:	14:00:00

For data types D (date), T (time), and Z (timestamp), it is not required to specify length, but you are allowed to do that. In case you specify it, the length must be according to the format of the date, time, and timestamp field. Optionally, you can also define a date or time format. The default is *ISO. If the field is initialized, the constant must be in the format specified in H specs. The literal should be defined in the following way:

- For date - D′date value′
- For time - T′time value′
- For timestamp - Z′timestamp value′

3.5.1 Initializing Date and Time Data Type Fields

The control specification defines the default format for internally defined date fields and literals used in the program. The keyword is DATFMT and if not specified, the default is *ISO.

The DATFMT is not to be confused with the DATEDIT keyword. The DATEDIT keyword defines the default format of numeric fields containing a date in the program and the delimiter used for editing. This includes the RPG reserved

words UDATE and *DATE, as they are numeric fields containing a date. The default format for DATEDIT is *MDY and / as delimiter.

The control specification also defines the format of internally defined time fields and literals. The default is *ISO.

Table 15. Date Formats with Control Specification

Keyword	Default	Description
DATFMT	*ISO	Default format for internally defined date fields and literals. Possible values are *MDY *DMY *YMD *JUL *ISO *USA *EUR *JIS.
DATEDIT	*MDY /	Format and delimiter for numeric fields containing a date including UDATE and *DATE Possible values for format are *MDY *DMY *YMD, and the separator character can be any character.
TIMFMT	*ISO	Default format for internally defined time fields and literals Possible values are *HMS *ISO *USA *EUR *JIS.

In the other RPG IV specifications, you can override the defaults for the date and time data types.

Note: For formats with a two-digit year (*YMD, *MDY, *DMY, *JUL), the AS/400 system treats any years ranging from 40 through 99 as 1940 through 1999, and 00 through 39 as 2000 through 2039.

Valid dates for *ISO, *USA, *EUR, and *JIS are from year 0001, month 01, day 01 through year 9999, month 12, day 31.

The calendar system used is the Gregorian calendar. If you want to use other calendar systems, you have to use the date and time APIs. With these, you can convert a Gregorian date to a Lilian date, and then use the Lilian date to find the day of the week, or have it converted into the Japanese or Republic of China calendar, based on era. Refer to 3.5.5, “Using Date and Time APIs” on page 47 for more information about these APIs.

3.5.2 Example of Initializing Date and Time Data Type Fields

Figure 31 shows an example of how the date and time data type fields are initialized:

```
1 H DATMFT(*YMD&) TIMFMT(*HMS&)

2 D Date1           S               D   DATFMT(*MDY) INZ(D'94 04 01')
3 D Time1           S               T   TIMFMT(*HMS) INZ(T'13 00 00')
4 D NumDate         S              6  0 INZ(910401)
5 D DateHire        S               D   DATFMT(*USA)
6 D UpdDate         S               D   DATFMT(*ISO)
7 D UpdTime         S               T   TIMFMT(*USA)
8 D EndTime         S               T   TIMFMT(*ISO)
```

Figure 31. Initializing Date and Time Data Type Fields

The following list gives an explanation of the source statements:

1 The control (H) specification defines the default format for internally defined date data types and literals to be *YMD with & (blank) as the delimiter with the DATFMT keyword.

The TIMFMT keyword defines the default format for time data types and literals to be *HMS with & (blank) as the delimiter.

2 This line defines a date data-type field with the external format of *MDY (mm/dd/yy). It is initialized with the date April 1 1994, and the format of this constant must be in the format defined by the control specification, and therefore it is year, month, and date separated by a blank. The length of the field is implicitly defined. A length can be specified, and must match the defined format.

3 The field Time1 is a time data type field with external format *HMS (hh:mm:ss). The constant used to initialize the field must follow the definition in the control specification.

4 NumDate is a numeric field - it is not a date data type field.

5 DateHire is a date data type field with the format *USA (mm/dd/yyyy)

6 UpdDate is a date data type field, format is *ISO (yyyy-mm-dd).

7 UpdTime is a time data type field, format is *USA (hh:mm xM).

8 EndTime is a time data type field, format is *ISO (hh.mm.ss).

To Initialize a date data-type field with today's date within a program, you can use the following method to be independent of the date format in the control (H) specification.

```
 * Define data structure with ISO date and subfields for yyyy/mm/dd
D TodayDS         DS
D Today                           D   DATFMT(*ISO) Inz
D  TodayY                        4    OVERLAY(Today:1)
D  TodayM                        2    OVERLAY(Today:6)
D  TodayD                        2    OVERLAY(Today:9)
 * Initialize the subfields with year, month, and day
C                   Move      *Year         TodayY
C                   Move      *Month        TodayM
C                   Move      *Day          TodayD
```

Figure 32. Initializing a Date with Today's Date

This example is using *ISO format, but you can also use similar coding for *YMD date format if you only want to have the two-digit year. The data structure contains the field to be used for today's date, Today. It is initialized to 0001-01-01. The year, month and day fields are overlaying portions of the date field and then used to initialize the date.

3.5.3 Calculations with Date and Time Data Types

Working with business dates is straightforward. Using the duration operation codes, you can add or subtract a number of years, months, and days to or from a date. Leap years are taken into account.

A leap year is a year that is a multiple of four, but not a multiple of one hundred. However, a year that is a multiple of 400 is a leap year. For example, year 2000 is a leap year, but year 1900 is not.

For time, you can work with hours, minutes, and seconds. It is not possible to mix the durations for data types. For example, you cannot find the duration in minutes between two dates.

To find the very last day of a month, you can use the example in Figure 33 The date wanted is the last day in the month, two months from today, June 6, 1994.

```
D PmdayDS         DS
D Pmday                          D   DATFMT(*ISO) Inz(D'1994-06-16')
D  PmdayY                       4    OVERLAY(Pmday:1)
D  PmdayM                       2    OVERLAY(Pmday:6)
D  PmdayD                       2    OVERLAY(Pmday:9)
 * Set date to the first day of the month
C                   Move      '01'          PmdayD
 * Add the number of months wanted, plus 1
C                   AddDur    3:*M          Pmday
 * Subtract one day from the date
C                   Subdur    1:*D          Pmday
 * Pmday now = '1994-08-31'
```

Figure 33. Calculating with Date and Time Data Types

A year is not a whole number of days, rather about 365.242 days, or an approximation of 365.25 days. You could have found the number of days between the two dates, divided this figure with the 365.25 approximation to get the age. This method works fine within a century, but is inaccurate when used over more centuries.

The solution is to define both fields with subfields for month and day. After subtracting the two dates to find the number of years between them, an additional test is made. If the person's birthday is after the day tested for, then one year is subtracted from the number of years found as shown in Figure 34.

```
 * A person is born August 24, 1951
D BornDS          DS
D Born                           D   DATFMT(*ISO) Inz(D'1951-08-24')
D  BornM                        2    OVERLAY(Born:6)
D  BornD                        2    OVERLAY(Born:9)
 * Today is June 10, 1994
D TodayDS         DS
D Today                          D   DATFMT(*ISO) Inz(D'1994-06-10')
D  TodayM                       2    OVERLAY(Today:6)
D  TodayD                       2    OVERLAY(Today:9)
 * Field to hold age of the person
D Age             S             3  0

C     Today         Subdur    Born          Age:*Y
 * The result of this operation is 43
 * Then test to see if the month and day of birth is after today's date
 * and adjust, if it is
C                   If        (BornM >= TodayM) and (BornD > TodayD)
C                   Sub       1             Age
C                   End
 * The result in this example is 42
```

Figure 34. Calculating with Date and Time Data Types

When comparing dates, they do not have to be in the same format. If different formats are used, the compiler ensures that they are compared in compatible formats. The example in Figure 35 on page 45 illustrates this:

```
D Pmday                          D   DATFMT(*ISO)
D Today                          D   DATFMT(*DMY)
 * The following test is valid
C                   If        (Pmday >= Today)
```

Figure 35. Calculating with Date and Time Data Types

The following list contains some important notes to remember when using date data type fields in calculations:

- Ensure a date is correct before moving it into a date data type field.
- When using TEST(D) or TEST(T) against a character field, the character field must have delimiters.
- *DATE and UDATE are not date data types and cannot be used with any comparisons with a date data type or with duration operations.
- Date data types with a two digit year cannot go beyond a century, and years 00 through 39 are between years 2000 and 2039.
- SUBDUR cannot be used to subtract two dates and have a result field that is a date data type field.

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C
C 1
C     OrderDate     ADDDUR    23:*Days      ShipDate
C 2
C     ShipDate      ADDDUR    NoYrs:*Y      WarntyDate
C 3
C     StartTime     ADDDUR    8:*hours      EndTime
C
C 4
C     Duedate       SUBDUR    XX:*Y         Loan_Date
C     Loan_Date     SUBDUR    DueDate       NumDays::D
C     EndTime       SUBDUR    8::hours      StartTime
C
C 5
C                   EXTRCT    BirthDate:*Y  Year
C                   EXTRCT    StartTime*colon.*M  Month
C
C 6
C                   TEST                    DateField             50
C     *DMY          TEST(D)                 Delv_Date             53
```

Figure 36. ADDDUR, SUBDUR, EXTRCT and TEST Examples

The following list contains explanations for the previous example:

- **1** Add 23 days to OrderDate and and put the result in ShipDate.
- **2** Add the value in NoYrs to the years portion of ShipDate and put the result in WarntyDate.
- **3** Add 8 hours to StartTime and put it in EndTime.

- **4** Similar examples with subtract. In the second example the operation subtracts DueDate from Loan_Date, converts it in dates, and puts it into the field NumDays.
- **5** Extract the year portion of BirthDate and put it into Year field.
- **6** Test the content of Delv_Date field according to date format *DMY. If the date is not valid, the indicator 53 is ON. In this example, the extender (D) is used since the field is a character or numeric field and not a date field.

3.5.4 Date and Time in MOVE Operations

Before moving a numeric or character field into a date data type field, ensure that the date is valid. If the date is not valid, an error is generated. You can use the TEST(D) (test date) operation code to check if a character or numeric field contains a valid date.

```
  H DATFMT(*YMD&) TIMFMT(*HMS&)
                :                :              :
                :                :              :
  D Date1           S             D   DATFMT(*MDY) INZ(D'94 04 01')
  D Time1           S             T   TIMFMT(*HMS) INZ(T'13 00 00')
  D NumDate         S            6  0 INZ(910401)
  D DateHire        S             D   DATFMT(*USA)
  D UpdDate         S             D   DATFMT(*ISO)
  D UpdTime         S             T   TIMFMT(*USA)
               :                :              :
               :                :              :
A C                   Move      Date1         UpdDate
  C* UpdDate now = '1994-04-01

B C                   MOVE      TIME1         UpdTime
  C* UpdTime now = '01:00 PM

C C     *YMD          Move      NumDate       DateHire
  C* DateHire now = '04/01/1991'

D C     *USA          MOVE      *DATE         UpdDate
E C     *USA          MOVE      '07:00 AM'    EndTime
```

Figure 37. Date and Time Data Types in MOVE Operations

The control and definition specifications for the example in Figure 37 are discussed in Figure 31 on page 42. The following list explains the MOVE operations:

A This instruction moves a date data type field to another date data type field. Since the two fields have different formats, a format change is made from *MDY to *ISO, and converted from two digits to four digits according to the previously described rules.

B The initialization of Time1 in definition specifications must be according to the format in H specs. RPG converts it to *HMS because of the *HMS specified in D specs. The UpdTime field is defined with *USA format. Moving Time1 to UpdTime, RPG converts it to *USA format because of the UpdTime definition.

Factor 1 in both examples is not used. Factor 1 is optional and is used only if we need to specify the format of a character or numeric field if it is the source or target of the operation.

C The numeric field NumDate is moved to DateHire. Since the format of NumDate is not known by the program, factor 1 is used to specify the correct format. The numeric field is converted to the format of the DateHire date field with delimiters inserted. If DateHire is not a valid date, an error is generated. You can handle an error using PSSR routine.

D The numeric field *DATE (the RPG reserved word for representing the job date with a four-digit year) is moved into the date field UpdDate. The control (H) specification did not specify DATEDIT, therefore the format for *DATE is mm/dd/yyyy.

Note: Since the format of *DATE and UDATE is derived at compile time, this program compiles with DATEDIT different from *MDY, but the program fails at runtime.

E The character literal must be in the format defined in factor 1 and include the correct delimiters.

The following table shows the usable combinations with DATEDIT in the control specification, and factor 1 on a MOVE statement with *DATE and UDATE in factor 2, and a date data type field in the result field.

*Table 16. Usable Combinations with DATEDIT for MOVE of *DATE and UDATE*

DATEDIT in H-spec	Factor 1 for *DATE	Factor 1 for UDATE
*DMY	*EUR	*DMY
*MDY (or blank)	*USA	*MDY
*YMD	*ISO (or *JIS)	*YMD

3.5.5 Using Date and Time APIs

There are a number of APIs available for calculation or representation in other calendar systems. These are standard APIs and are also used in the UNIX world. The calendar used for date and time calculations in these APIs is the Lilian calendar.

The Lilian calendar starts on October 15, 1582, which defines Lilian day 1. Therefore, only dates from October 15, 1582, through December 31, 9999, are valid for Lilian date calculations.

The date and time APIs are bindable APIs. This means that your RPG IV program cannot be created using default activation group with *YES. You must use *NO and give the activation group name when creating the program. Please refer to Chapter 7, “ILE Design Considerations” on page 83 for details on activation group. You might have to use the operation extender D on the call bound procedure. It indicates that operational descriptors are included and required by the bindable APIs.

The following examples show only a small part of the possibilities of the APIs. For a detailed description, please refer to the *System API Reference*.

Code fragment number **1** in Figure 38 on page 48 illustrates how to convert a date into a Lilian date. The CEEDAYS API requires a character field (DateChar) with a valid date as input parameter and the format of the date.

The statements marked with **2** use the CEEDYWK API with the Lilian date as input parameter to determine the number for the day of the week.

Since the APIs support character and integer data types and not date or time data types, code fragment **3** converts a date field to it's character representation before calling the API CEEDAYS.

The statements marked with **4** convert a Lilian date into a *Japanese era* date. The Japanese era calendar starts counting from day 1 for each emperor.

After converting a date into the corresponding Lilian date, you can convert it into the Japanese era date. For correct results, the CEEDATE API requires an output date format to be passed as second parameter of the bound call. Since the output character string contains DBCS characters between shift-out, shift-in control characters, the result can only be shown on DBCS capable devices. However, you can write the result into a data base file to look at it's hexadecimal representation.

Using this facility, Japanese companies can run their business applications using, for example *ISO, or *JIS date formats in their database. When needed, they can use these APIs to convert to Japanese era dates.

Similar function is available for Republic of China era dates.

The last example marked with **5** shows how to get a character representation of a Lilian date with the CEEDATE API. Again, the correct output format is passed to the API. Notice that it only returns English words.

```
 * Program to show the use of bindable date APIs.
 * Note that it cannot be compiled with default activation group,
 * since the CEE APIs are bindable APIs.
 *
 *  Define the date field with date of birth
DDateBorn        S               D   DATFMT(*ISO) INZ(D'1951-08-24')
 *  This field is holding the Lilian date
D LilDate        S             9B 0 INZ
 *  Field to receive the day of week
D DayOfWeek      S             9B 0 INZ
 *  Defines the format of the character field communicating with APIs
D Fmt            S            35    INZ('YYYY-MM-DD')
 *  Character field sending/receiving date
D DateChar       S            35    INZ('1994-08-24')
 *  Defines the format I want back for Japanese date
DFmtJpn          C                  '<JJJJ> YY.MM.DD'
 *  Field to hold Japanese date
D DateJpn        S            20
 *  Defines the format I want back for date with text
DFmtTxt          C                  'Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY'
```

Figure 38 (Part 1 of 2). Using Date and Time APIs

```
  1
 *  Lilian date returned is 150429.
C                   CallB(D)  'CEEDAYS'
C                   PARM                   DateChar
C                   Parm                   Fmt
C                   PARM                   LilDate

  2
 *  The number returned is 4, which is a Wednesday.
C                   CallB(D)  'CEEDYWK'
C                   PARM                   LilDate
C                   Parm                   DayOfWeek

  3 to find the Lilian date for the date field DateBorn.
 *  The Lilian date returned is 134723.
C     *ISO          Movel(P)  DateBorn     DateChar
C                   CallB(D)  'CEEDAYS'
C                   PARM                   DateChar
C                   Parm                   Fmt
C                   PARM                   LilDate
  4
 *  The date is era "Showa" year 26, month 8, day 24.
 *  The field returned contains the following (hex only for DBCS part):
 *                 26.08.24
 *          04B4704FF4FF4FF
 *          E535AF026B08B24
C                   Movel(P)  FmtJpn       Fmt
C                   CallB(D)  'CEEDATE'
C                   PARM                   LilDate
C                   Parm                   Fmt
C                   PARM                   DateJpn

  5
 *  The result returned is Friday, August 24, 1951.
C                   Movel(P)  FmtTxt       Fmt
C                   CallB(D)  'CEEDATE'
C                   PARM                   LilDate
C                   Parm                   Fmt
C                   PARM                   DateChar
C                   SETON                                        LR
```

Figure 38 (Part 2 of 2). Using Date and Time APIs

Note: The bindable API names are case-sensitive, and for the date and time API, you must use all uppercase.

3.5.6 Timestamp

A timestamp has only one format, yyyy-mm-dd-hh.mm.ss.msmsms. The timestamp data field is read from a database record, and you can use extract and duration operation codes with a timestamp.

To get a timestamp into your program without using a database file, you can use the QWCCVTDT API, which is not a bindable API.

```
H DATFMT(*YMD&) TIMFMT(*HMS&) DATEDIT(*DMY)
 * Define DS to hold result from QWCCVTDT with subfields for each element
 * The format is CYYMMDDHHMMSSMis
DSystimeDS        DS
D Systime                       16
D SystimeD                       6S 0 Overlay(Systime:2)
D SystimeH                       6S 0 Overlay(Systime:8)
D SystimeM                       3S 0 Overlay(Systime:14)
 * Define DS with timestamp field and subfields to receive values
DTimeDS           DS
D  TimeSt                         Z   Inz
D   TimeStDT                      D   overlay(TimeSt:1) DatFmt(*ISO)
D   TimeStHM                      T   overlay(TimeSt:12) Timfmt(*ISO)
D   TimeStMS                     3S 0 overlay(TimeSt:21)
 * Call QWCCVTDT to get current machine date and time
C                   Call      'QWCCVTDT'
C                   PARM      '*CURRENT  '  Parm1            10
C                   PARM                    Dumm2            10
C                   PARM      '*YMD      '  Parm3            10
C                   PARM                    Systime          16
C                   PARM      X'0000'       error             4
 * Move fields from current machine date and time to timestamp fields
C     *YMD          Move      SystimeD      TimeStDT
C     *HMS          Move      SystimeH      TimeStHM
C                   Move      SystimeM      TimeStMS
 * Only milliseconds are set in the microsecond part of timestamp
```

Figure 39. Using Timestamp Data Type

Another method for getting a timestamp is to use the bindable API for Get Current Local Time, CEELOCT.

The program is only slightly different from the previous example, as this API returns 17 bytes for the timestamp. The difference is that the CEELOCT returns a full four-digit year, where the QWCCVTDT has a century digit as the first byte.

```
 * Define DS to hold result from CEELOCT with subfields for each element
 * Format is YYYYMMDDHHMMSSMis
DSystimeDS        DS
D Systime                       17
D SystimeD                       8S 0 Overlay(Systime:1)
D SystimeH                       6S 0 Overlay(Systime:9)
D SystimeM                       3S 0 Overlay(Systime:15)
 * Define DS with timestamp field and subfields to receive values
DTimeDS           DS
D  TimeSt                         Z   Inz
D   TimeStDT                      D   Overlay(TimeSt:1) DatFmt(*ISO)
D   TimeStHM                      T   Overlay(TimeSt:12) Timfmt(*ISO)
D   TimeStMS                     3S 0 Overlay(TimeSt:21)
 *  This field is holding the Lilian date
D LilDate         S              9B 0 INZ
 * Call CEELOCT to get current machine date and time - it also returns
 * Lilian date as well as the Lilian number of seconds
C                   CallB(D)  'CEELOCT'
C                   PARM                    LilDate
C                   Parm                    FLOAT8            8
C                   PARM                    SysTime
 * Move fields from current machine date and time to timestamp fields
C     *ISO          Move      SystimeD      TimeStDT
C     *HMS          Move      SystimeH      TimeStHM
C                   Move      SystimeM      TimeStMS
 * Only milliseconds are set in the microsecond part of timestamp
```

Figure 40. Using Timestamp Data Type

If you just want to get the time with an accuracy in seconds, you can, of course, use the TIME operation code. With the TIME operation code, you get the date as well, but remember that the format of the date returned is dependent on the DATFMT job attribute.

When the timestamp data type field is initialized within your program or read from a database, it is used for duration calculations.

3.6 Example Using Import/Export Data Structure

In this example, we have two RPG modules bound together into a program. Procedure RPG1 shown in Figure 42 on page 52 performs a static call to the procedure RPG2 shown in Figure 43 on page 53.

1. Module RPG1 displays a panel and the user enters the customer number. The user wants to see details about the customer entered. See Figure 41 on page 52.

```
13:12:32                                                      9/06/94

              Customer Information

 Enter Customer Number:  123456


   Customer Name:  Peter Smith

   Status:         A              (A=Active  N=Inactive)

   Discount:       10,00

   Debit Limit:    100.000,00
```

Figure 41. Imported and Exported Data Structure

2. CUSTNUM is a field received from the display format and contains the customer number.

3. Both modules have defined an external defined data structure with the fields. We want to retrieve the content of them from the data base file and display it to the screen. In RPG1, the data structure is defined with keyword EXPORT, and in RPG2 with keyword IMPORT. That means that the storage area for both data structures is the same. Remember the name and characteristics of export/import items must be the same in both modules.

```
+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+..
 FDSPFILE   CF   E              WORKSTN

 DCUSTDETAIL     E DS                   EXTNAME(CUSTOMER)
 D                                      INZ
 D                                      EXPORT
 DCUSTEXIST        S              1     EXPORT

 C                   WRITE     KEYLINE
 C                   EXFMT     DSPF1
 C                   DOW       *IN03 = *OFF
 C                   EVAL      CUSTEXIST = *BLANKS
 C                   CALLB     'RPG2'
 C                   EVAL      *IN10 = CUSTEXIST
 C                   WRITE     DSPF2
 C                   CLEAR     CUSTDETAIL
 C                   EXFMT     DSPF1
 C                   ENDDO
 C                   EVAL      *INLR = *ON
```

Figure 42. Example for EXPORT/IMPORT: Exporting Procedure

4. Module RPG1 uses a bound procedure call to module RPG2. CUSTNUM is a field received from the display format and contains the customer number entered by the user. The same field is also a subfield of the export/import data structure.

5. Using the CUSTNUM as a key field RPG2 chains the data base file.

6. If a customer is not found, RPG2 moves the value of indicator 10 to the field CUSTEXIST. CUSTEXIST is also an EXPORT/IMPORT field in both modules.

CUSTEXIST is used to send a message to the user that customer was not found.

7. If a customer is found, RPG1 displays a record format with the customer details.

```
+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+..
 Fcustomerf if   e           k disk

 DCUSTDETAIL     E DS                  EXTNAME(CUSTOMER)
 D                                     import
 DCUSTEXIST        S              1    import

 C     custnum       chain     rf1                                10
 C                   EVAL      CUSTEXIST = *in10
 C                   EVAL      *INLR = *ON
```

Figure 43. Example for EXPORT/IMPORT: Importing Procedure

Note:

This code is deliberately written in uppercase and lowercase.

3.7 Example Using Pointers in RPG IV

Pointer indicates the address of a space and not the space itself. We already use pointers in RPG. The OCCUR in multiple occurrence data structure, the index in arrays and the parameters passed to a called program are pointers indicating where the data is. The new EXPORT/IMPORT keyword definitions are also pointers. We also use pointers to indicate specific address in a user space. Later in this chapter, after we explain the BASED keyword and the %ADDR built in function, we have an example of how to use pointer (*).

The examples shown in Figure 44 on page 54 through Figure 47 on page 55 include a command, an ILE CL program, and an RPG IV program. The command, using the CL and RPG programs, creates an outfile containing information about the modules that are used in an ILE program. This information is also available through the DSPPGM command but only for display or print. The complete source examples are shown in Appendix B, “RPG IV Coding Examples” on page 179.

A space of 32KB is assigned overlaying the user space that contains the list of all modules in a specific ILE program. The space starts at pointer value PTR.

Since the user space that is returned contains not only the list of modules but also a header, a more precise layout with offsets is specified. This is done by overlaying the SPACE with an array ARR of 32767 separate fields of 1 byte.

What we are only interested in to begin with is:

- The field that contains the address of the begin of the module list
- The field that contains the number of modules in the list

For both fields, we specify the overlay. OFFSET starts at 125 bytes from the beginning of the user space, and SIZE starts at 133 bytes from the beginning.

```
DPTR             S                *                            Pointer field
DSPACE           DS                   BASED(PTR)               Assign start of
 *                                                              the userspace
D SP1                       32767                              First subfield
 *                                                              in userspace
 * ARR is used with offset to set the pointer to array         Re-align Usrspc
DARR                            1    OVERLAY(SP1)               with 1 byte
                                     DIM(32767)                 array
 * Offset is pointing to start of array                        Offset value for
DOFFSET                         9B 0 OVERLAY(SP1:125)           start of the list
 *                                                              data section in
 *                                                              the user space
 * Size has number of module names retrieved                   Number of list
DSIZE                           9B 0 OVERLAY(SP1:133)           entries
```

Figure 44. Pointers: Definition of Pointers in D-specs

The next step is to get the address of the beginning of the module list:

1. Field OFFSET contains the offset value of the start of the module list data section in the array, for example, X'104'. So, the byte we want the address of is (104 + 1).
2. The pointer value of that field is moved into pointer field MODPTR.
3. Automatically the start of the array MODARR is re-aligned to this pointer value (see the definition of MODARR).

```
C                   EVAL      MODPTR = %ADDR(ARR(OFFSET + 1))
```

Figure 45. Pointers: Receive a Pointer

The next step is to overlay a new array with an element length of 508 (as described in the API manual) and with a number of 500 elements (this number is arbitrary).

We also need to specify a data structure with the specific layout of the information of one module in the 508 bytes of an array element.

```
DMODPTR          S                *                            Pointer field
DMOD_ENTRYS      C                    CONST(500)               Initial Array
                                                                length
DMODARR          S             508    BASED(MODPTR)            Re-align the
                                      DIM(MOD_ENTRYS)           start of list
 *                                                              entries in the
 *                                                              user space
DX               S              11  0                          Array index val
DMOD_INFO        DS                                            Re-define of
 *                                                              ONE list
                            entry
D PGM_NAME                      10
D PGM_LIB                       10
D MOD_NAME                      10
D MOD_LIB                       10
D SRC_FILE                      10
D SRC_LIB                       10
D MEMBER                        10
D MOD_ATTR                      10
D                              428                                           Filler
```

Figure 46. Pointers: Define Module Information Array

Now we process all of the module list entries and write them into a database file.

```
 *
C                  DO        SIZE                         Do as many times as
 *                                                         list entries avail.
C                  ADD       1              X             Increase index nbr
C                  EVAL      MOD_INFO = MODARR(X)         Move array entry to
 *                                                         data structure
C                  WRITE     MODLST         MOD_INFO      Write record
C                  ENDDO
```

Figure 47. Pointers: Write Module List to a Database File

You will notice that the space of 32K has been addressed multiple times:

- After receiving the USRSPC pointer, it is aligned with SPACE.
- The subfield SP1 starts at the same address.
- And finally, the module list MODARR starts at an address X'105' bytes from the beginning of SP1.

This is an example of a program using an overlay technique for user spaces. There are many more APIs that produce information in a user space. Search in the API manual for more information about the content of the user space regarding the header and list layout, and use this example as a base for your own pointer programs.

Chapter 4. Conversion Considerations

To take advantage of the enhancements provided with the RPG IV language definition,you might consider converting your existing RPG application. As you can appreciate, RPG IV gives you more flexibility and better readability, and as a result of these advantages, you can improve your development environment and productivity.

An important point is that the new design of RPG IV opens the door for future enhancements and functions that were impossible to implement on the previous compiler. The reason was the restrictions in the layout on the different specifications statements. Increasing the length for some of the entries, introducing keywords, and free format operations in the extended calculation specifications lifted these limitations.

IBM offers you a simple Control Language command to convert your RPG/400 or RPG III source members to RPG IV source members.

Before you start migrating your RPG source members, you might want to create a new source file to store the RPG IV sources. The default name for the RPG IV source file is QRPGLESRC. It is recommended that you use this name because all the CL commands use this name as the default when they refer to the RPG IV source file.

When you are creating the target source file using the command CRTSRCPF for RPG IV members, *specify 112 for record length.* Twelve characters are for sequence numbers and date fields. The additional 100 characters receive your source code. The length of the statement field has been increased to include the new size of the entries in the specifications layouts. If you leave the default size, you lose the commenting text entries.

If you are using variant characters in your program, make sure that the CCSID of the source file is specified correctly. The CCSID of a new source file created on V3R1 is taken from the default CCDID job attribute, unless a CCSID is specified on the CRTSRCPF command.

There is also a new source type, RPGLE. The SEU and PDM recognize the new type as an RPG IV source. If the source type is RPGLE, SEU performs the appropriate syntax check for RPG IV statements.

In the Work with Members Using PDM panel, there is a new option (15=Create module). Using this option, if the source type is RPGLE, PDM submits to job queue to run the command CRTRPGMOD to create an ILE RPG module. Using option 14, if the source type is RPGLE, PDM submits to job queue the command CRTBNDRPG to create a bind RPG program with only one module. More about these commands appears later in this chapter.

RPGLE is also the attribute type for an ILE RPG program and for module objects.

Note: The conversion command from RPG/400 to RPG IV source members renames automatically the source type to RPGLE.

The following list describes different approaches for your conversion to the RPG IV language and their advantages and disadvantages:

© Copyright IBM Corp. 1995

- Migrate program by program. An application is converted step by step. OS/400 allows an application to run in a mixed environment with ILE and non-ILE programs.

  Advantages: We do not have to stop running the application.

  Disadvantages: We do not gain any of the ILE functions.

  We do not have any performance improvement.

  We have to deal with two types of source members and compilers.

- Migrate an entire application, converting RPG/400 to RPG IV without using any ILE functions and facilities.

  Advantages: Only RPG IV source members and compiler to deal with.

  Easy to start implementing ILE functions step by step.

  Disadvantages: We have to stop running the application until we migrate all the programs including the CL programs.

  We do not gain any of the ILE functions.

  We do not have any performance improvement.

- Migrate an entire application, taking advantage of the ILE functions and facilities.

  Advantages: Only RPG IV and ILE CLL source members and compilers to deal with.

  ILE functions and application control implementation.

  Performance improvement and development productivity.

  Disadvantages: Migrating an application and performing changes using ILE functions needs a very good knowledge of ILE.

  The migration needs more resources and time.

The migration of every program requires two steps. The first step is to convert the source members from RPG/400 to RPG IV. The second step is to create the program object. We will see later that to create an ILE program can also be a two-step procedure.

4.1 CVTRPGSRC Conversion Command and Parameters

The CVTRPGSRC command converts RPG/400 to RPG IV source code. A conversion report is printed by default. The command maps the old layouts to the new ones, converts specific functions to keywords, moves data structures from I specs and arrays from the E specs to the new D specifications. The CVTRPGSRC command is part of ILE RPG licensed program product.

Please note that the conversion tool does not support conversion from RPG IV back to RPG/400 layout.

The source program is assumed to be error free. Using CVTRPGSRC does not remove the unsupported operation codes FREE and DEBUG.

Source File CCSID

If your source contains variant characters, the from source file has a CCSID different from the to source file, and the job CCSID is not 65535, then the source may be converted into an undesired source file CCSID, and may cause compilation failures or unexpected results. See 5.2, "Source File CCSID Considerations" on page 71 for more details.

See Figure 48 for a description of parameters in the CVTRPGSRC command.

```
                          Convert RPG Source (CVTRPGSRC)

Type choices, press Enter.

From file  . . . . . . . . . . .  __________     Name
  Library  . . . . . . . . . . .    *LIBL        Name, *LIBL, *CURLIB
From member  . . . . . . . . . .  __________     Name, generic*, *ALL
To file  . . . . . . . . . . . .  QRPGLESRC      Name, QRPGLESRC, *NONE
  Library  . . . . . . . . . . .    *LIBL        Name, *LIBL, *CURLIB
To member  . . . . . . . . . . .  *FROMMBR       Name, *FROMMBR

                              Additional Parameters

Expand copy member . . . . . . .  *NO            *NO, *YES
Print conversion report  . . . .  *YES           *YES, *NO
Include second level text  . . .  *NO            *NO, *YES
Insert specification template  .  *NO            *NO, *YES
Log file . . . . . . . . . . . .  QRNCVTLG       Name, QRNCVTLG
  Library  . . . . . . . . . . .    *LIBL        Name, *LIBL, *CURLIB
Log file member  . . . . . . . .  *FIRST         Name, *FIRST,*LAST
```

Figure 48. Command Convert RPG Source.

4.1.1 CVTRPGSRC Parameters

The following list gives an explanation of the parameters in CVTRPGSRC command:

From file
: The source file name that contains the RPG/400 source member(s) to be converted. Normally this is the QRPGSRC source file. This is a required parameter.

From member
: Specifies the name of the source member to be converted. Valid types of source members to convert are RPG, RPT, RPG38, RPT38, and SQLRPG. The conversion command does not support conversion of RPG36 and RPT36 source members. The conversion tool automatically expands first the RPT and RPT38 source members to RPG and then converts them to RPG IV. This is necessary since ILE RPG/400 does not support auto-report source members.

If you specify *ALL, you can convert all of the source members in a source file. You can also specify a generic name. This is a required parameter.

To file This is the name of the source file to receive the converted source member or members. The source member type is changed to RPGLE. For SQLRPG source members, the new type is SQLRPGLE. The default source file name is QRPGLESRC. The source file must exist and should have a record length of 112 characters. Option *NONE allows you to test whether there are any problems in the conversion. To create a conversion report parameter, CVTPRT must be *YES.

To member The default is *FROMMBR. This is valid only if FROM and TO files are not the same.

Expand copy member This parameter allows you to decide if you want to include /COPY members in the conversion result. The default *NO means do not expand the /COPY member or members into the converted source. *YES means expand the /COPY member or members permanently. See 4.1.2, "/COPY Considerations" on page 61 for more considerations about /COPY.

Print conversion report The default is *YES creates a conversion report. The warning messages issued during conversion are found in Appendix C, "Migration Information" on page 185. Before converting all your source, it is recommended that you create this report using *NONE for *To file* and *ALL for *From member*. The report contains messages such as:

- `CALL operation code found`. This is useful in case you want to change the CALL to CALLB operation code.
- `DEBUG operation code is not supported in RPG IV.`
- `FREE operation code is not supported in RPG IV.`
- `/COPY compiler directive found`. This is also very useful in case the /COPY refers to data structures.

For further analyzing, you can use the tool described in 4.1.4, "Scanning Tool for Migrated Source Code" on page 64.

Insert specification template If you specify *YES, specification templates are inserted in the converted source member. The templates are inserted at the beginning of the appropriate specification section. The default is *NO.

Log file The log file is used to track the conversion information. The file must exist before you start using the CVTRPGSRC command. A default file exists in library QRPGLE. The name of the file is QARNCVTLG. The default name in the parameter is QRNCVTLG. You can create your own file by copying the file QARNCVTLG to another library using the command CRTDUPOBJ. The name of the file is the default name QRNCVTLG or any other name. The

log file is accessed using QUERY or a user-written program. You can find the DDS definition for this file in *ILE RPG/400 Programmer's Guide* , SC09-1525.

Note:

The conversion command also converts source from a database file. If the records of the database file contain sequence numbers, you should remove them before you start with the conversion. The recommended record length for a database file with RPG statements to convert is 100 characters.

4.1.2 /COPY Considerations

In case you use /COPY statements in RPG/400 source members, you might discover errors during compilation and not during conversion. The following list contains a description of what kinds of problems may occur and recommendations to resolve them:

- All source members that are copied into programs (using /COPY) during compilation *must* be converted *before* you create the ILE module or ILE program.
- The conversion tool includes data structures in copy members *following the Input specifications.* This problem occurs only if the source member with /COPY data structures also contains Input specifications. The compilation fails and gives this message: `form type out of sequence`. This happens since data structures are now defined in Definition (D) specifications and D specs in RPG IV precede input specifications.

  There are two techniques to resolve this type of problem:

  1. Using the parameter EXPCPY (Expand copy member) with value *YES includes all /COPY members in the target source. This seems to be the easiest work around, but you loose flexibility in maintaining your applications.
  2. To keep your /COPY members for data structures separate, convert your sources and parse them with compiler option *NOGEN. If you get the message: `Form type out of sequence`, you have to manually move the data structures to the definition specifications.

- You might experience similar problems using /COPY for definitions in Line Counter specification and Record Address File in Extension specification. In RPG IV these definitions are replaced by keywords in File Definition specifications. As a result of that, when you convert the content of /COPY members and the file is not part of the member, the conversion aid cannot determine where to insert the keywords.
- During the conversion, the tool merges stand-alone arrays with data structure subfields with the same name. If the data structure and the array are not in the same source member (one of them is in a /COPY member), the conversion aid cannot merge them and, as a result of that, the compilation fails with a compile-time error.
- If a /COPY member contains just definitions in Input specification, the conversion tool cannot determine how to convert them correctly without the surrounding specifications on the primary source member. In the following example the /COPY member contains two fields:

```
I                                        1   7 FLD1
I                                        8  15 FLD2
```

Without the context to the primary source member, the conversion tool cannot determine if these fields are:

- Data structure subfields to be placed in the definition specifications
- Fields of a program described file to remain in the input specifications
- Externally described subfields in the I specs, or
- Renamed fields of an externally described file.

In those cases, the tool assumes that they are part of a data structure and moves them to the definition specifications. The original definition is a comment included in the target source:

The previous example is converted to:

```
D FLD1                           1    7
D FLD2                           8   15
I*                                       1   7 FLD1
I*                                       8  15 FLD2
```

If this assumption turns out to be wrong, you have to manually correct the resulting source.

4.1.3 Conversion Problems

Two of the known RPG IV to RPG II conversion problems that you may have to correct manually are described below.

4.1.3.1 Combining Arrays with An Externally Described DS Subfield

When combining an array definition with an externally described data structure subfield, the following situation can occur:

```
A            R RECFMT
A              CUSNUM    10
A              SUM       10 1
```

The externally described data structure TOTAL describes two fields, CUSNUM and SUM **1**. In the I specifications of your RPG/400 source code, the data structure is declared as externally described **2**.

```
...
E                      SUM      10  1 3
IDSNAME      E DSTOTAL                2
...
C             CUSNUM   DSPLY
C                      MOVE  *ON      *INLR
...
```

In addition, your RPG/400 source code contains an array declaration for SUM **3** in the E specifications. When merging the externally described data structure with the source code the conversion results in the following source statements:

```
...
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++
DSUM           S             1   DIM(10)         4
D DSNAME      E DS                EXTNAME(TOTAL)  4
...
C     CUSNUM      DSPLY
C                 MOVE  *ON      *INLR
...
```

The conversion tool is not able to look into the externally described data structure and treats the array and data structure as separate entities **4**. Although this piece of converted code is syntactically correct, the compile fails since it is not allowed to define a stand-alone array and a data structure subfield with the same name. To solve this problem, use the compile listing and correct your source:

```
...
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++
D DSNAME      E DS                EXTNAME(TOTAL)
D  SUM        E                   DIM(10)  5
...
C     CUSNUM      DSPLY
C                 MOVE  *ON      *INLR
...
```

Now the array **5** is defined as a subfield to the data structure.

4.1.3.2 Renaming and Initializing an Externally Described DS Subfield

The following example illustrates another incompatibility that cannot be detected by the conversion aid:

```
IDsname....NODsExt-file++.............OccrLen+......
ITOTAL     E DSCUSTOMER                                1
I              SUMFIELD                          SUM  2
I I           '9999999'                          SUM  3
C           SUM       DSPLY
C                     MOVE *ON       *INLR
```

Externally described data structure CUSTOMER **1** defines the subfield SUMFIELD that needs to be renamed **2** because of the length restrictions and initialized **3**. The conversion results in the following code and, as you can see, generates two definitions for the same subfield **4**:

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++
D TOTAL         E DS                  EXTNAME(CUSTOMER)
D  SUM          E                     EXTFLD(SUMFIELD)   4
D  SUM          E                     INZ('9999999')     4
C     SUM           DSPLY
C                   MOVE      *ON           *INLR
```

As a result of that, a compile-time error is produced since the field is defined twice. In order to correct it, you should manually combine the two definition lines in one field definition with two keywords **5**.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++
D TOTAL         E DS                EXTNAME(CUSTOMER)
D  SUM          E                   EXTFLD(SUMFIELD) INZ('9999999')  5
C     SUM           DSPLY
C                   MOVE      *ON           *INLR
```

4.1.4 Scanning Tool for Migrated Source Code

The CVTRPGSRC command provides a conversion report that should be very useful in order to verify the status of your converted code. However, this is only spool output. In the GG244358 library, there is a command for scanning a source file that you can use after the conversion. This library is supplied on a diskette in the back of this publication. Instructions on how to install the library on your system are found in Appendix A, "Diskette Install Instructions" on page 177.

The command allows you to specify a search argument, for example, CALL, that is used to scan the source file and write all the source records that contain the string in an outfile. The command is called:

```
                             Scan Source - TAA (SCNSRC)

 Type choices, press Enter.

 File name  . . . . . . . . . . . > yoursource    Name, *CBL, *CL, *CLP...
   Library name . . . . . . . . . >   yourlib     Name, *LIBL, *CURLIB
 Argument to scan for . . . . . . > CALL
 Member name  . . . . . . . . . .   *ALL          Name, generic*, *ALL
 Translate source to upper case     *NO           *YES, *NO
 Print following statement  . . . > *NO           *YES, *NO
 Library for SCNSRCP file . . . .   GG244358      Name, *NONE, *LIBL, *CURLIB
 Member to receive output . . . .   SCNSRCP       Name
 Replace data in member . . . . .   *YES          *YES, *NO
```

In the source file GG244358/QATTINFO, there is a description of the scan source command.

Besides using the outfile in your own programs, you can use Query as well. For that reason, two initial queries are also provided in the library that you might need to change depending on your situation.

4.2 Source Conversion Example

In this example, we convert an RPG/400 to RPG IV source member. The program contains only sample code to show how some of the specifications are changed when converting to RPG IV. It is not intended to be a program that compiles and runs.

You should enter the CVTRPGSRC command with the following parameters:

```
CVTRPGSRC FROMFILE(yourlib/QRPGSRC)  FROMMBR(PGM1) +
          TOFILE(yourlib/QRPGLESRC) INSRTPL(*YES)  +
          EXPCPY(*YES)
```

Figure 49 on page 65 shows the source of the RPG/400 version.

```
  FPAYROLL IF  E                   DISK
  FQPRINT  O   F     132     OF    LPRINTER
3 LQPRINT   51FL 48OL
  E                    ARR        15  1               Define Array
7  * Define a data structure with subfields
4 I            DS
  I                                       1   6 YMD
  I                                       1   2 YY
  I                                       3   4 MM
  I                                       5   6 DD
4 I            DS
  I                                       1  15 ARR
  C                     EXCPTHDG                        Prt HDG
  C* Read a record
  C                     SETOF                     010203
  C                     SETOF                     040506
  C           *IN20     DOWEQ'0'
  C* Do calculations
6 C   01 02 03PAY       SUB  200       NET     72       Net pay
6 C   01 02 03          EXCPTDETAIL                     Prt detail
6 C   03 04
  COR 05 06             READ PAYREC                 20 EOF
  C*
6 C  N03N02             EXSR PAYC                       Calc pay
  C                     ENDDO
  C***
  C                     SETON                     LR    Set LR
  C                     RETRN                           Return
  C                     CALL 'DUMMY'                    Call dummy
8 C                     DEBUG                            debug
8 C                     FREE 'DUMMY'                    Free dummy
  C* Last record calculations
6 CLRN03N04             EXCPTTOTAL                      Prt Total
  C/COPY PAYTAX
  OQPRINT  E  206           HDG
  O                                   25 'Heading'
  O        E  1             DETAIL
  O                                   25 'Detail'
  O        E  1             TOTAL
  O                                   25 'Total'
```

Figure 49. Conversion Example: Before

Figure 50 on page 66 shows the PAYTAX /COPY member.

```
C           PAYC      BEGSR
C           *LIKE     DEFN RATE      PAY   + 2       Define Pay
C           *LIKE     DEFN HOURS     OTIME           Define Otime
C           *LIKE     DEFN RATE      OTPAY + 2       Define OTPay
C* Calculate payment - hours over 35 get additional 75 %
C*
C           HOURS     IFLE 35                        Hours <= 35
C           HOURS     MULT RATE      PAY             Total pay
C                     ELSE                            -else-
C           RATE      MULT 35        PAY             Full 35 hrs
C           HOURS     SUB  35        OTIME           Overtime hours
C           RATE      MULT 1.75      OTRATE  94      Find rate
C           OTRATE    MULT OTIME     OTPAY     H     Overtime pay
C                     ADD  OTPAY     PAY             Total pay
C                     END
C                     ENDSR
```

Figure 50. Conversion Example: COPY Member

Besides the new layout for all specifications, note the following about the result of the conversion:

1 Parameter INSRTPL(*YES) inserts in the converted source an appropriate template at the beginning of every specification section.

2 Parameter EXPCPY(*YES) expands into the converted source member the /COPY member or members.

3 Some of the file definitions are converted to keyword definitions.

4 Data structures from input specification have been moved to definition specification. Note that data structure definitions precede input specifications.

5 The array definition has been moved from Extension (E) specification to Definition specification. Arrays that are subfields in a data structure are now defined together with the subfield using the keyword DIM.

6 Calculation statements with more than one condition indicator have been split to up to three lines. Each line contains only one condition indicator. AN is added to the second and third line before the indicator.

7 Comments preceding moved statements are also moved.

8 On the Print Conversion report, you receive warning messages because the program contains the Op-Codes FREE, DEBUG, and CALL. You have to remove the Op-Codes FREE and DEBUG in order to compile the program. You might want to change the Op-Code CALL to CALLB in order to call a bind procedure.

Figure 51 on page 67 shows the PGM1 source member after the conversion.

```
1 F*ilename++IPEASFRlen+LKlen+AIDevice+.Functions+++++...++++++++Comments+
  F*
  FPAYROLL   IF   E             DISK
  FQPRINT    O    F  132        PRINTER OFLIND(*INOF)  3
  F                                     FORMLEN(51)    3
  F                                     FORMOFL(48)    3
7  * Define a data structure with subfields
1 D*ame+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++
4 D                 DS
  D  YMD                   1      6
  D  YY                    1      2
  D  MM                    3      4
  D  DD                    5      6
4 D                 DS
5 D  ARR                   1     15
  D                                     DIM(15)                      Define Array
1 CLON01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
  C                   EXCEPT    HDG                                  Prt HDG
  C* Read a record
  C                   SETOFF                                010203
  C                   SETOFF                                040506
  C     *IN20         DOWEQ     '0'
  C* Do calculations
6 C   01
  CAN 02
  CAN 03PAY           SUB       200           NET              7 2          Net pay
6 C   01
  CAN 02
  CAN 03              EXCEPT    DETAIL                               Prt detail
6 C   03
  CAN 04
  COR 05
  CAN 06              READ      PAYREC                          20     EOF
  C*
6 C  N03
  CANN02              EXSR      PAYC                                      Calc pay
  C                   ENDDO
  C***
  C                   SETON                                 LR       Set LR
  C                   RETURN                                         Return
  C                   CALL      'DUMMY'                              Call dummy
  C                   DEBUG                                           debug
  C                   FREE      'DUMMY'                              Free dummy
  C* Last record calculations
6 CLRN03
  CANN04              EXCEPT    TOTAL                                Prt Total
2 C*/COPY PAYTAX
  C     PAYC          BEGSR
  C     *LIKE         DEFINE    RATE          PAY            + 2          Define Pay
  C     *LIKE         DEFINE    HOURS         OTIME                       Define Otime
  C     *LIKE         DEFINE    RATE          OTPAY          + 2          Define OTPay
  C* Calculate payment - hours over 35 get additional 75 %
  C*
  C     HOURS         IFLE      35                                   Hours <= 35
  C     HOURS         MULT      RATE          PAY                    Total pay
  C                   ELSE                                            -else-
  C     RATE          MULT      35            PAY                    Full 35 hrs
  C     HOURS         SUB       35            OTIME                  Overtime hours
  C     RATE          MULT      1.75          OTRATE           9 4        Find rate
  C     OTRATE        MULT(H)   OTIME         OTPAY                  Overtime pay
  C                   ADD       OTPAY         PAY                    Total pay
  C                   END
  C                   ENDSR
1 OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+...........................
  OQPRINT    E           HDG           2 06
  O                                         25 'Heading'
  O          E           DETAIL        1
  O                                         25 'Detail'
  O          E           TOTAL         1
  O                                         25 'Total'
```

Figure 51. Conversion Example: After

4.3 Creation Commands

For the user, the program creation for ILE languages is basically a two step process. Step one creates an intermediate result that is called a module. Step two takes one or multiple modules and turns them into a running program.

Many features of the Application Development Tools Set licensed program product have been enhanced to support RPG IV and ILE. Program Development Manager (PDM) option 15 allows you to create modules. Option 14 in conjunction with source type RPGLE invokes CRTBNDRPG and option 26 prompts you for the CRTPGM command to bind multiple modules into a program. All of these options are available from the Work with Objects using PDM panel. For a discussion of the new terms modules, ILE programs, and procedures, refer to Chapter 7, “ILE Design Considerations” on page 83.

4.3.1 Create RPG Module

The command CRTRPGMOD creates an object of type *MODULE containing non-running code. An object of type *MODULE can hold one or more procedures that need to be bound into a program object. User-written procedures are run using the new syntax for *call bound procedure*, for example, the CALLB operation code as described in 3.1.3, “New Operation Code for Static Call” on page 33 for RPG, or the CALLPRC command for ILE CL.

An interesting parameter not available in the RPG/400 compiler is the FIXNBR (Fix numeric). The parameter allows you to specify whether zoned decimal data, which is not valid, is fixed by the compiler upon conversion to packed data. The values for this parameter are:

***NONE** This is the default value. *NONE means that invalid zoned decimal data is not fixed by the compiler on the conversion to packed data and results in decimal data error.

***ZONED** Invalid zoned decimal data is fixed by the compiler on the conversion to packed data. Blanks in numeric fields are treated as zeros. Every invalid digit is replaced by zeros. If the sign is not valid, it is forced to a positive sign of F.

4.3.2 Create Program

The CRTPGM command binds one or more modules and creates a bound program object (*PGM). All of the participated modules are bound in one single program. The modules must be created and exist when you use the command CRTPGM. The CRTPGM is an OS/400 and not an RPG command. You can bind together modules created from different languages.

One of the modules must be defined as the Program Entry Procedure Module (PEP). This module contains the procedure that is called first, using a dynamic call to this program. The CL command used to call a program is CALL, and the opcode used in RPG is also CALL.

4.3.3 Create Bound RPG Program

The CRTBNDRPG command creates first a temporary module from the RPG IV source member and then creates the program. The source must be an RPG IV code. The program contains a single module. You call a program using either the CL CALL command, or the RPG CALL opcode. If you are in PDM and the source type is RPGLE, you can use option 14 instead of CRTBNDRPG.

Further reading

See the *CL Programming manual*, (SC41-3721), or the online help, for more information about the different parameters in the create commands.

Chapter 5. National Language Support with RPG IV

When businesses are expanding into new markets, there is a requirement to meet the cultural expectations of the end users in the countries. For programmers, it is important to be able to support all requirements for all countries without having to change the programs for each new country or language.

This chapter discusses the recommended techniques to be used for RPG IV programs. For more details related to other application parts, such as database, display, printer, and message files, please refer to *International Application Development*.

5.1 Recommended Usage of Characters in RPG IV

Since the alphabetic characters differ from country to country, and they do not have the same meaning (hexadecimal value) for all countries, it is strongly recommended that you only use characters in the invariant character set for all naming and constants within your programs.

Table 17. Invariant Character Set

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9			
+	<	=	>	%	&	*	"	'	(	)	.	,
_	-	/	:	;	?							

Characters outside this character set are called variant characters. Note that the characters $, #, and @ are not part of the invariant character set. In some countries, they represent uppercase characters. For example, in Denmark and Norway, the # represents the uppercase character Æ, and its lowercase equivalent is æ.

Since the compiler translates only the letters a-z to uppercase, the effect of the conversion could create the following situation: A Danish programmer would expect that all names could be entered in lowercase Danish. The name *SÆLGER* is a valid field or file name for RPG IV, but it cannot be entered as *sælger*.

5.2 Source File CCSID Considerations

If you only use invariant characters in the program source, you do not have to worry about CCSIDs for your source files.

However, if you are using variant characters (except for comments), you have to be aware of the CCSID tagging of your source files, if not already done.

© Copyright IBM Corp. 1995

When creating a source file prior to V3R1, the CCSID of the source file is by default taken from the job performing the create. If the CCSID of the job is 65535, the source file gets CCSID 65535, which means no conversion. This is called implicitly tagging.

There is also a keyword on the CRTSRCPF for CCSID. With this you can specify the CCSID you need for the source file. This is called explicitly tagging.

With V3R1, the default has changed if the job is running with a CCSID of 65535. In this case, the CCSID of the source file is the default CCSID, which is a new job attribute in V3R1. The default CCSID in the job is found based upon the language ID job attribute.

Important

When a source file that was implicitly tagged with CCSID 65535 is residing on a system installing V3R1 or restored onto a V3R1 system, the CCSID of that source file is set to the system value default CCSID. This value is based upon the system value for language ID. Make sure that your source files have the CCSID corresponding to the content.

The CCSID of a physical file is changed if it has no logical files created over it.

The RPG IV compiler uses the CCSID of the primary source file for compilations. Any /COPY source members are converted to the CCSID of the primary source file.

5.3 Externalizing Constants

To make your programs more flexible for translation, it is important to have all language-sensitive constants external to the programs.

It is also important to have the formats of cultural dependent data items, such as dates, externalized and selectable. This is accomplished by providing the options in an options file where the customer can select the required values at install or runtime.

Constants are either stored in message files, database files, or any other object that can hold the necessary information. The programs can then retrieve the information when needed.

One method to become independent of what the end user wants to see on displays and printed output is to use decimal numbers within the program to be used for testing. You supply the relation between the numeric and external value in a table.

For example, you want to provide a yes or no option with one character. In a message description, you define the internal value for no as 0, and the yes value is 1. The program accepts the character value for input, checks it for validity, and moves the corresponding decimal value to the field used. For output, the program takes the numeric value and moves it to the output field.

5.4 Date Fields

Applications should have only one method for storing dates. The format has to be consistent throughout the application. Otherwise, the end users are confused.

The most convenient format to use for storing is probably year/month/day. This format allows comparison and sorting directly with the field. Application programs can then show the format required by the end user.

Today, many applications use many different methods for storing and working with dates. If your application is not capable of handling dates from year 2000 and forwards, you should change the application to use the new date data types over the next few years. New applications are recommended to use the new date data types. The recommended format for storing in the database is *ISO.

5.5 Sort Sequence

The sequence used by the system and compilers for comparing and ordering is by default the EBCDIC hexadecimal value in the code page.

Using character data according to their hexadecimal values does not necessarily provide the correct sequence from a linguistic point of view.

Typical problems are:

- All lowercase characters precede any uppercase character.
- The national characters not included in the English alphabet are not in the correct position between English characters where they usually fit.

Prior to V2R3, the various components, such as database, query products, and compilers had different methods for solving the problem for ordering and comparing data according to cultural needs. Even some of the products, for example, SQL, did not provide any possibility for influencing the ordering and comparison of data.

With V2R3, the sort sequence was introduced. The sort sequence support provides a consistent interface to reference the tables for ordering and comparing.

RPG IV has the same support. To make the program independent of user requirements for sort sequences, and to only have one set of programs for all languages, it is recommended that you use the *JOBRUN parameter for sort sequence and language ID when creating programs or modules. The program then uses the job attributes for sort sequence when it is run.

To compile a program or module with the external reference to sort sequence, you have to specify the ALTSEQ keyword in the control specification:

```
H Altseq(*EXT)
```

The *EXT parameter tells the compiler to use the parameters in the create command for the sort sequence. You would then specify *JOBRUN (default is *HEX) for the sort sequence. The default is *JOBRUN for the language identifier.

```
Sort sequence . . . . . . . . . SRTSEQ        *JOBRUN
  Library . . . . . . . . . . .
Language identifier . . . . . . LANGID        *JOBRUN
```

Note: When a program reads a database file and performs functions such as level break, matching record, or other comparisons using key fields that have a sort sequence applied, the sort sequence used by the program must be compatible with the sort sequence of the database file. Otherwise, unexpected results may occur.

5.6 Case Conversion

When using data, it is sometimes necessary to ensure that character data is in all uppercase. To be able to perform correct casing of the data, you need to know it is encoded, meaning which CCSID was used for entering the data.

With V3R1 a new convert case API is available. This API has two interfaces, one for OPM calls and one for ILE bound calls:

- QLGCNVCS for OPM calls
- QlgConvertCase for ILE bound calls

The parameter list used is the same for both. The following example shows how to use the bound call.

```
  * Program to show the use of the bindable API for case conversion.
  * Note that it cannot be compiled with default activation group.
  *
  *  Define the Request Control Block
 D ReqCtlDS        DS
 D  Request                       9B 0 INZ(1)
 D  CCSID                         9B 0 INZ(277)
 D  Case                          9B 0 INZ(0)
 D  Reserved                     10    INZ(X'00000000000000000000')
  *  Defines the input data I want to convert
 DInpData          S             10    INZ('aczæøå')
  *  Defines the output data field
 DOutData          S             10
  *  Field to hold the length of data
 DLgtdata          S              9B 0 INZ(%Size(InpData))
  *  Error code - not used in this program
 DErrCode          S              9B 0 INZ(0)
  *  Name of convert API - it is case sensitive.
 DCvtCase          C                   'QlgConvertCase'

  *  Convert the data InpData field to upper case using CCSID 277.
  *  Program name is too long to have in factor 2 - use named constant
 C                   CallB(D)  CvtCase
 C                   PARM                    ReqCtlDS
 C                   Parm                    InpData
 C                   Parm                    OutData
 C                   Parm                    LgtData
 C                   Parm                    ErrCode
  *  OutData contains:  ACZÆØÅ
```

The API can also perform uppercase to lowercase conversion using CCSIDs. Using a different format of the request control block, it is possible to use conversion tables instead of CCSIDs. This support is similar to the support in QDCXLATE API.

For more information on the case conversion API, please refer to the *System API Reference*, SC41-3801.

5.7 DBCS Graphic Data Type

RPG IV now supports the graphic data type. The field type is G.

It is important to notice that when defining a graphic data type field, you specify the number of characters, not the number of bytes.

```
DField3           S              10G
```

Field3 is a graphic data type field with 10 DBCS characters. The space occupied by this field is 20 bytes.

Chapter 6. CL and ILE CL

ILE CL was added to the V3R1 OS/400 system support to allow application developers to create ILE CL programs and procedures to eliminate the need to understand the intricacies of ILE when running CL in the Integrated Language Environment.

The purpose of this chapter is to discuss the aspects of using CL programs with RPG IV programs in compatibility mode as well as considerations for moving CL to ILE CL programs.

6.1 ILE CL Functions

Several new terms used in this section are explained in Chapter 7, "ILE Design Considerations" on page 83.

The following three new CL commands have been added:

- CRTCLMOD
  - Creating a ILE CL module that is then used to create other programs and service programs.
- CRTBNDCL
  - Create a bound ILE CL program. This is similar to creating a CL module and then specifying only that module on the CRTPGM. This creates a CL program in one step.
- CALLPRC
  - A static call to another module or procedure. This is a call to an RPG, COBOL, C, or even another CL procedure.

6.1.1 Changes to Existing Interfaces

Commands that display or otherwise use program or module attributes have minor changes to include the attribute for CL modules and bound ILE programs.

The CL compiler listing for ILE programs and modules is similar to the compiler listing for the OPM CL compiler. However, the intermediate code generated by the compiler is not listed.

ILE CL procedures are able to issue static procedure calls to other ILE procedures in the same program object or in a service program; this is achieved by using the new CALLPRC command.

A new source type (CLLE) has been added to PDM, SEU, and Programmer's menu as well as a new program attribute. For example, if CLP is specified for the source type, CRTCLPGM is used to create a CL program from the Programming Development Manager (PDM). If CLLE is used, CRTBNDCL is used to create an ILE CL program.

Two new special values for *where allowed to run* have been added to CL commands. These values *IMOD and *BMOD are used to specify that a command can run only in an interactive ILE CL program or a batch ILE CL

© Copyright IBM Corp. 1995

program. *IPGM and *BPGM indicate that the command is allowed to run in a program, ILE or OPM.

System/38 environment is not supported for ILE CL. For instance, if a user was running in S/38 environment, and attempted to compile source in S/38 syntax, the program does not compile for ILE CL. System/38 environment commands are allowed if they are specified in native syntax. System/38 environment commands that are compiler directives, for example, DCLDTAARA, SNDDTAARA, RCVDTAARA, 38 version of DCLF, are not supported in ILE CL.

Although we code procedures or modules the source for a CL procedure is denoted with a PGM...ENDPGM such as a CL program is today. There is only one occurrence of a PGM and ENDPGM in the source for a CL module so therefore, a CL module is made up of only one procedure. If several CL modules are bound together, the module specified on the ENTMOD parameter is used as the main entry point. The name of the CL procedure is the same as the module name specified on the CRTCLMOD command.

6.1.2 CL Considerations with RPG IV in Compatibility Mode

Since the compatibility mode scenario does not take advantage of the ILE architecture with activation group, there are only a few items to look for.

Most important is the use of MONMSG for RPG messages. Even with RPG IV programs created in compatibility mode, the escape messages issued have changed.

If you monitor for RPG9001 today, you have to exchange this message for CEE9901, if you still want to keep the CL program monitoring for this message. You should consider changing the RPG program to handle error situations instead. For error handling with RPG IV programs, please refer to Chapter 10, “Exception Handling” on page 161.

Notice that the new command CALLPRC is not valid in a OPM CL program.

6.1.3 ILE CL Considerations

Almost all CL commands are valid for use in ILE CL programs. Following are the considerations if you are moving your CL programs to ILE CL.

TFRCTL The transfer control command is not valid within ILE CL, and an ILE CL source containing the TFRCTL command will fail to compile.

You can use the PRTCMDUSG command to get a listing of CL programs using the TFRCTL command, or you can use the PDM scan function to perform the search within the CL program source.

RCLRSC The RCLRSC command has no function related to non-default activation groups. RCLACTGRP is a new command to free resources associated with activation groups.

RTVCLSRC The RTVCLSRC command is not allowed against an ILE CL module or program object to retrieve the CLLE source. However, the RTVCLSRC is used within a CLLE module or program to retrieve the source from an OPM CL program.

PRTCMDUSG PRTCMDUSG cannot be used against a CLLE module or program, but is used within CLLE against a CL program object.

SNDPGMMSG The receiving point of SNDPGMMSG *PRV is changed within ILE. The program message is sent to the program stack entry that has the next higher control entry procedure.

MONMSG Some of the messages have been changed, and if you monitor for these messages, you need to change them. The RPG9001 error message is replaced by CEE9901.

CALLPRC This new command is only used for calling ILE procedures, and only if the program is created using DFTACTGRP(*NO).

Note: There is no previous release support for ILE CL in V3R1M0.

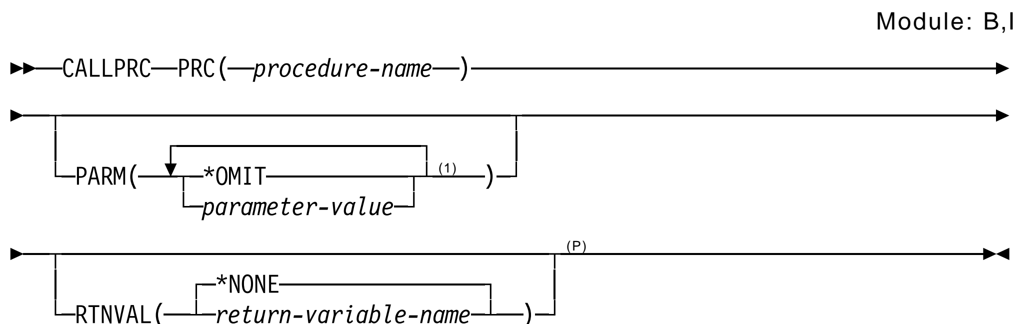
There are a few differences between the create commands for CL and ILE CL that you should be aware of:

- The Allow RTVCLSRC keyword on the CRTCLPGM command does not exist for ILE CL module or program creation.
- The GENOPT keyword on the CRTCLPGM command does not exist for ILE CL module or program creation, and thus no intermediate representation of a program (IRP) listing is generated.
- The default activation group (DFTACTGRP) keyword must be *NO, and you must specify an activation group name or *CALLER in the ACTGRP keyword when you want your program to run outside the default activation group.
- The compiler option *GEN/*NOGEN is supported for CRTCLMOD only. CRTBNDCL always creates a program object.

6.1.4 The Call Bound Procedure Command

The Call Bound Procedure (CALLPRC) command calls a bound procedure named on the command, and passes control to it. Optionally, the procedure issuing the CALLPRC command can pass parameters to the called procedure. The CALLPRC command is used in compiled ILE CL programs and modules. Upon return, the return code from the called procedure is placed in the RTNVAL parameter if specified.

Module: B,I



Notes:

[1] A maximum of 300 repetitions

[P] All parameters preceding this point are specified in positional form.

Figure 52. CALLPRC Command Syntax

Each parameter value passed to the called procedure is a character string constant, a numeric constant, a logical constant, a floating-point constant, or a CL program variable. If a floating-point constant is specified, the value is converted to double-precision format and passed to the called program. If parameters are passed, the value of the constant or variable is available to the program that is called. Parameters cannot be passed in any of the following forms: lists of values, qualified names, expressions, or keyword parameters. Up to 300 parameters are passed to the called procedure.

Note: Although the CALLPRC command allows up to 300 parameters to be passed, the number that the called procedure can accept depends on the language of the called procedure. For example, a CL procedure cannot accept more than 40 parameters.

If parameters are passed to a procedure using the CALLPRC command, the values of the parameters are passed in the order that they appear on the CALLPRC command; this order must match the order in which they appear in the parameter list in the calling procedure.

Parameters in a called procedure are used in place of its variables. However, no storage in the called procedure is associated with the variables it receives. Instead, if a variable is passed, the storage for the variable is in the procedure where it was originally declared. If a constant is passed, a copy of the constant is made in the calling procedure, and that copy is passed to the called procedure.

The result is that if a variable is passed, the called procedure can change its value and the change is reflected in the calling procedure. If a constant is passed, and its value is changed by the called procedure, the changed value is not known to the calling procedure. Therefore, if the calling procedure calls the same procedure again, the values of constants are set to their original values, but the variables do not change.

6.1.4.1 Parameter Passing

One of the differences between ILE CL and OPM CL is in the way the situation is handled when a caller passes either more or fewer parameters than the called program expects.

In OPM, the caller gets escape message CPF0001 with diagnostic message CPD0172 in the job log.

This no longer occurs in ILE. If the caller of an ILE CL program specifies more than the number of parameters expected by the CL procedure, the program entry procedure (PEP) passes along the expected number of parameters; parameters after the expected number is ignored. If the caller of an ILE CL program specifies fewer than the number of parameters specified by the caller, it also passes enough null pointers so that the CL procedure gets the number of parameters that it expects.

The CALLPRC command always passes the number of parameters that are specified on the command. If the called procedure expects fewer than are passed, the extra ones are ignored. If the called procedure expects more than are passed, the results are unpredictable; it depends on what the called procedure tries to do with the missing parameters.

As a result, callers of ILE CL procedures and programs do not get a parameter mismatch message if they specify the wrong number of parameters. However, CL programmers are able to write ILE CL programs and procedures with variable length parameter lists.

6.2 Changing Source Type from CL to CLLE

Instead of manually changing the source type, you can create a PDM option to do the change of source type.

From the Work with Members Using PDM screen, use F16=User options to create user-defined PDM options. The new option is, for example, called CT for change type. The command to use is CHFPFM, and you can enter this:

```
                        Create User-Defined Option

/Type option and command, press Enter.

   Option  . . . . . . . . .   CT   Option to create

   Command . . . . . . . . .   CHGPFM FILE(&L/&F) MBR(&N) SRCTYPE(CLLE)
```

If you have a source file containing different member types, or you want to change the type using a generic name, use the Subset Member List panel.

```
                              Subset Member List

Type choices, press Enter.

  Member . . . . . . .   *ALL          *ALL, name, *generic*

  Member type  . . . .   CLP           *ALL, type, *generic*, *BLANK

  From date  . . . . .   01/01/00      Earliest date to include

  To date  . . . . . .   12/31/99      Latest date to include

  Text . . . . . . . .   *ALL
```

When you have the subset list you want, type option CT on the first entry in the list, use F13=Repeat, and press the Enter key to change the CL programs to CLLE. You do not see the changed member type until you press F5=Refresh.

6.3 Should I Move CL to ILE CL?

The answer to this question depends on the choice of programming language, and how you want to exploit the Integrated Language Environment in your application:

- If you are running ILE programs in compatibility mode, then it is recommended you keep your CL programs and not move them to CLLE.

  You may have to modify your CL programs if you monitor for RPG messages within CL programs.

- If you want to take advantage of the application isolation in ILE, you have to move your CL programs to ILE CL.

  It is not advisable to mix the use of compatibility mode and non-default activation groups within an application.

With these recommendations, the next sections are looking at the considerations for using CL with RPG IV programs, and moving CL programs to ILE CL.

Chapter 7. ILE Design Considerations

This chapter concentrates on aspects of the ILE architecture that *must* be addressed from a design perspective. The contents should be read by anyone involved in either designing migrations from Original Program Model (OPM) to ILE or designing new ILE applications.

7.1 Overview of ILE Concepts

ILE provides new concepts to support both a runtime environment with fire walls built around an application, and a development environment supporting the production of highly modularized, reusable code. These new concepts are summarized below. For full details of each of these facilities, please refer to the publication *Integrated Language Environment Concepts*, SC41-3606.

- **Procedures**

  A procedure is a set of self-contained HLL statements that perform a particular task and return to the caller. In ILE RPG/400 and ILE CL, there is one procedure per source member and one procedure per module. The procedure name is always the same as the containing *MODULE name for RPG IV and ILE CL. In ILE C/400, there may be multiple procedures (called functions in C) within a module. Refer to Program Entry Procedure on page 86 for this special form of a procedure.

- **Modules and programs**

  A module object (*MODULE) is the result compilation using the new ILE compile commands. A module cannot be run. In order to be run, a module must be bound into a program object (*PGM).

  An ILE program object (*PGM) is the result of binding one or more modules together. You run programs on the system, just as you did in OPM.

- **Static binding**

  In OPM, all calls to programs are dynamic, involving system overhead to perform authority checking and find the address of the program. Binding is the process of creating a program by packaging ILE modules and resolving symbols passed between those modules.

  When a dynamic call is made to an ILE *PGM object, the program is activated. This activation involves the initialization of all static storage required by variables used by the modules within the object. In the case of RPG IV, all variables are stored in static storage.

  There are two types of static bind available within ILE. Once a program and any related service programs have been activated, there is no difference in the system code that is run to perform a bound call by reference or a bound call by copy.

  1. **Bind by copy**

     This is the process of copying modules directly into a program object during the binding phase. Thus, the modules specified to be bound by copy are contained directly within the program object.

© Copyright IBM Corp. 1995

2. **Bind by reference**

   This is the process of making procedures indirectly available to a program (*PGM) through a service program (*SRVPGM). Modules bound in a *PGM by reference to a *SRVPGM are not copied into the *PGM object.

   Optional service programs (*SRVPGMs) is associated with the *PGM at bind time. The activation of these associated service programs involves the initialization of **all** static storage in **all** modules within the service programs.

An example showing a program that includes both procedures bound by copy and procedures bound by reference.

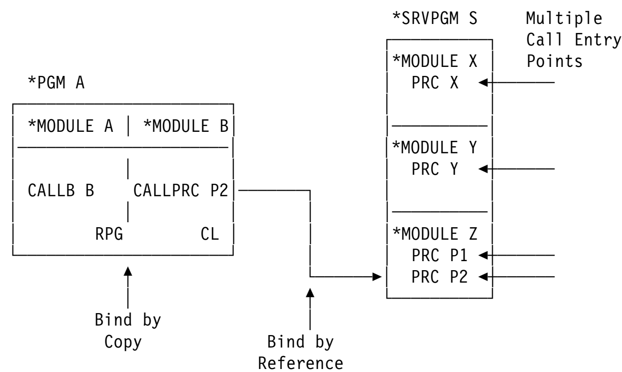


Figure 53. Bind by Copy and by Reference

This example shows the result of program A being created using the CRTPGM command. Procedures A and B (in modules A and B, respectively) are bound by copy. Procedures X, Y, P1 and P2 are bound by reference as they are contained in a service program object named S.

- **Service programs**

  Service programs cannot be run through a dynamic call. They act as a container for related procedures that are used in many different *PGMs. Thus, in order to easily maintain these popular procedures, they are stored in one place - a service program.

  Since activation of a *SRVPGM causes initialization of all static service programs, it involves the initialization of *all* static storage in *all* modules within the service programs. Even if your *PGM only uses one module from a service program, if the service program contains N>1 modules, then the static storage for all N>1 modules is initialized at activation of your *PGM.

- **Binding Language**

  This is a very simple language that controls which procedures and variables are available from a service program to other programs or other service programs.

- **Activation groups**

  An activation group enables partitioning of resources within a job. Therefore, an activation group is only seen by the job in which it was created. An activation group consists of static storage needed for variables belonging to activated programs, open files (open data paths), commitment control definitions and application programs.

When a new job is started, the system automatically creates two activation groups. These are collectively referred to as the default activation group, but are split into activation group number 1 and number 2. Output from Display Job (DSPJOB), Option 18 = Display activation groups, is shown in Figure 54 on page 85.

```
                          Display activation group
                                                          System:   RCHASM02
 Job:   P23KRZ75D      User:   QPGMR          Number:   014444

 Type options, press Enter.

 ----activation group-----
 Number         Name          In Use Indicator
 0000000001     *DFTACTGRP       Yes
 0000000002     *DFTACTGRP       Yes
 0000001230     PDSRVPG          No
 0000001224     QLGCASE          No
```

Figure 54. DSPJOB - Activation Groups

Notes:

1. Activation group number 1 is the system default activation group reserved for running system functions.
2. Activation group number 2 is the user default activation group, available to application programs.
3. PDSRVPG is the activation group enabling problem determination support.
4. QLGCASE is the activation group for monocasing. This activation group gets created in every job because service program QLGCASE is created with the activation group named QLGCASE.

All programs (system or user) run in an activation group. The activation group in which an application program runs is determined at program creation.

When you create ILE programs and service programs, you specify the activation group in which your program runs using the ACTGRP keyword. You can choose one of the following options:

1. NAMED activation group

   An activation group with a name you specified at the time of creating ILE programs.

2. *NEW activation group

   Every time the program is dynamically called, a new activation group is created. As soon as the program returns control to its caller, the *NEW activation group is deleted. Frequent use of this choice provides the worst performance within your application.

3. *CALLER activation group

   The program is run in the activation group of its calling program.

- **ILE Program Activation**

When a dynamic call to an ILE program is issued within a job, the system performs the following tasks, known as program activation:

1. Identify what activation group the *PGM should run in.

   If the activation group is NAMED and does not exist, then create it. If the activation group is *NEW, then create a new activation group in which to run the *PGM (this is very expensive in terms of CPU).

   If the *PGM has not been called in this activation group before (always the case with *NEW activation groups), then initialize all static storage for all associated modules, whether they are bound by copy or bound by reference.

2. Activate all service programs that have been bound to the *PGM. Identify which activation group the *SRVPGM should run in.

   If the activation group is NAMED and does not exist, then create it.

   If the *SRVPGM has not been called in this activation group before, then initialize all static storage for all modules in the service program.

3. Pass control to the Procedure Entry Point (PEP) in the first procedure specified in the MODULE list at *PGM creation time.

4. If the entry procedure is RPG IV without using the logic cycle, then control passes to the first executable statement in the calculation specifications after full opens of required files and resolution of passed parameters, exactly as happened in OPM.

When the program is deactivated and it runs in a *NEW activation group, then the activation group is deleted and all associated resources are returned to the system.

- **Binding Directory**

  A list of modules or service programs. The contents of a binding directory are only used by the binder if unresolved imports exist during either the bind of modules specified on the MODULE list, or modules exported from service programs specified on the BNDSRVPGM list.

- **Program Entry Procedure (PEP)**

  A PEP is system generated and the first procedure placed on the call stack following a dynamic call. This procedure is always given control first, following a dynamic call. The PEP ensures that the procedure *you* specified as the Entry module on the CRTPGM is given control following a dynamic call.

  Service programs never have PEPs on the call stack. This is because any procedure in a bound service program may be called (multiple entry points (MEP)); there is no concept of a first procedure to always be run (PEP) in a service program. (Remember that service programs cannot be run by a dynamic call.)

  The name of the PEP on the call stack depends on the ILE HLL used for the Entry module (ENTMOD) of the program. These names are:

  – _C_pep for ILE C
  – _Q_QRNP_PEP for ILE RPG/400
  – _CL_PEP for ILE CL

7.2 ILE Compile and Bind Commands

There are two categories of ILE compile commands provided:

1. Full-function ILE Compile and Bind Commands

   The Create Module and Create Program commands provide access to all of the features available within ILE. Greater design flexibility is provided as a result of splitting the compile and bind into two separate commands. The Create Module commands are used to compile a module object, hence the command is ILE compiler-dependent. CRTRPGMOD is used to create an RPG IV module and CRTCLMOD is used to create an ILE CL module. In order to obtain an executable *PGM object within ILE, it is necessary to perform a compile of source code into a module object (*MODULE) followed by a bind of the *MODULE object(s) into a program object (*PGM). Either the Create Program (CRTPGM) command is used to bind *MODULE objects into a program, or the CRTBNDxxx command is used to perform a one step compile-and-bind to create a program.

   The Create Service Program (CRTSRVPGM) is used to bind modules into a service program object. As previously stated, you cannot directly run a service program; you can only indirectly run procedures in a service program through bound calls from the *PGM object that was created with references to the service program.

2. Restricted ILE Compile and Bind Commands

   The Create Bound Program commands CRTBNDRPG and CRTBNDCL provide access to some but not all of the ILE facilities. Thus, these commands are designed to be simple to use and, consequently, do not provide the flexibility of the full-function ILE commands.

   Use of these commands enables you to take advantage of RPG IV. If you elect to run in OPM compatibility mode, then you cannot use the call bound (CALLB) operation code; the compile part of the process fails with message `RNV5378 Severity 30 'CALLB cannot be used when DFTACTGRP(*YES) is specified for CRTBNDRPG'`. If you specify DFTACTGRP(*YES), then your program is only run in the default activation group; hence you are unable to take full advantage of ILE named activation groups and resource scoping.

7.2.1 OPM Compatibility Mode

Compatibility mode is an ILE program attribute that, when enabled, makes an ILE program behave in a manner *compatible* with OPM program behavior. This facility is available for the ILE RPG/400 and ILE CL programming languages; it is not available for ILE C/400.

The way to enable OPM compatibility mode for an ILE program is to use the CRTBNDRPG or CRTBNDCL command and specify DFTACTGRP(*YES).

Use this command if you want to migrate all or part of your application from RPG III to RPG IV, but you do not want to take advantage of ILE bound calls, service programs or activation groups at this time.

If you enable compatibility mode (specify DFTACTGRP(*YES) on the CRTBNDxxx command), then the following ILE facilities are *NOT* available to you:

1. Bound calls
2. EXPORT/IMPORT keywords on D Specifications

3. Service programs
4. Use of named or *NEW activation group

7.2.2 Comparison of Compile/Bind Commands

Table 18 shows exactly which ILE facilities are supported by which ILE commands. It deliberately does not include commands related to service programs.

Table 18. Comparison of ILE Compile and Bind Commands

Features Available	**CRTxxxMOD**	**CRTPGM**	**CRTSRVPGM**	**CRTBNDxxx DFTACTGRP(*NO) 1**	**CRTBNDxxx DFTACTGRP(*YES) 1**
Purpose	Compile	Bind	Bind	Compile and Bind	Compile and Bind
Can bound call be used?	Yes	Yes	Yes	Yes	No
Use of Service Programs	n/a	Yes	Yes	No	No
Use of Binding Directory	n/a	Yes	Yes	Yes	No
Use of Non-Default ACTGRP (Named, *NEW, *CALLER)	n/a	Yes	Yes	Yes	No
Use of Default ACTGRP Only	n/a	No	No	No	Yes
Multiple *MODULES on MODULE Keyword	n/a	Yes	Yes	n/a	n/a
Optimization	Yes	n/a	Yes	Yes	Yes
Debug	Yes	n/a	Yes	Yes	Yes
OPM Compatibility Mode	n/a	No	No	No	Yes
*MODULE retained after compile	Yes	n/a	n/a	No	No

Note:

- 1 The CRTBNDxxx command in this table applies to the CRTBNDRPG and CRTBNDCL commands.
- n/a = not applicable

7.3 Activation Groups

All OPM programs run in the same activation group, the system-provided default activation group.

Within ILE, activation groups are used to logically partition applications that are used concurrently within the same OS/400 job. Activation groups exist for the life of the job that created them. Once the job is terminated, then all activation groups associated with that job are deleted.

Creation of a new ILE program or service program always involves the specification of the activation group in which the program runs. Thus, resources such as the order entry (OE) and accounts receivable applications are separated such that there is no conflict over files common to both applications, commitment control scoping, and variables used within programs common to both applications.

This separation of applications and application resources within a job is implemented in ILE through the use of activation groups, as shown in Figure 55.

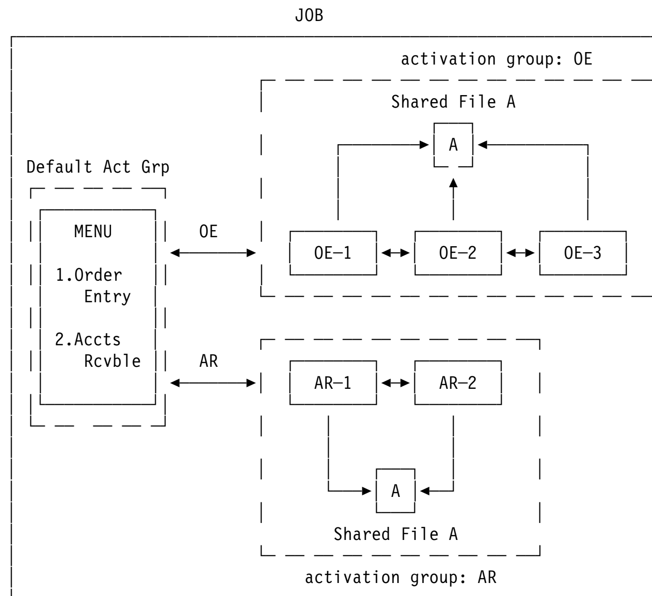


Figure 55. Example - ILE RPG/400 Application

Note that there are two open data paths in the job for File A, each of which is shared at the activation group level. This was, of course, not possible in OPM. Thus, in ILE we can take advantage of sharing open data paths and ensure application isolation within a named activation group.

7.3.1 Default activation group

OPM programs and ILE programs created with OPM compatibility mode always run in the default activation group. Specifically, these programs always run in default system activation group number 2, which is the user-portion of the default activation group available for your applications to use.

7.3.2 User-Named Activation Group

Upon a dynamic call to an ILE program that specifies a named activation group, if the activation group does not already exist within the job, then it is created. An important difference between named activation groups and the other types of activation groups is that a new activation group is only deleted when the Reclaim Activation Group (RCLACTGRP) command is issued.

Specify the name of an activation group using the ACTGRP keyword on the CRTPGM or CRTBNDxxx commands.

We recommend that, wherever possible, you design your application to run in a named activation group whose name is unique to your application. You must explicitly specify the name of your activation group on the application start-up program. Thus, all subsequently called programs may be created with activation group *CALLER, such that they run in the activation group of the program that dynamically called them (the named activation group).

7.3.3 Activation Group of Caller

We expect that most ILE programs are created to run in the same activation group as their calling program. In order to specify this, you should use ACTGRP(*CALLER) on the CRTPGM command. Control of where your program is run is then determined by the application start-up program or programs that explicitly specify your application′s activation group name.

You can also specify that an ILE service program be run in the same activation group as the ILE program to which it is bound by using the default of ACTGRP(*CALLER) on the CRTSRVPGM command.

7.3.4 System-Named Activation Group (*NEW)

This is the default on the CRTPGM command, and is deliberately not available on the CRTSRVPGM command. Whenever a program created with ACTGRP(*NEW) is called dynamically, a new activation group is created, the program is activated and run, and when the program returns control to its caller (through RETURN, LR or a hard leave), the activation group is deleted.

Warning

Avoid using ACTGRP(*NEW), even though it is the default on CRTPGM. Use of this activation group option is the worst choice for performance.

We recommend that this option is **not used** within an ILE application. It is very expensive in terms of system resource to keep creating and deleting *NEW activation groups.

We recommend that you run ILE programs in a named activation group rather than allowing them to run in the default activation group.

7.3.5 Activation Group Recommendations

Application start-up program or programs should be created to run in NAMED activation group or groups, such that programs subsequently called can safely specify ACTGRP(*CALLER). Since activation group names can only be changed by re-creating the program, this approach minimizes your effort in case of a change.

Do not use ACTGRP(*NEW) unless it is absolutely necessary. Consider changing the default value to *CALLER, using the Change Command Default (CHGCMDDFT) command.

Your application design should ensure that ILE programs created with ACTGRP(*CALLER) are not called from the default activation group. When an ILE program is run in the default activation group, its files may be closed by the RCLRSC command. The RCLRSC command may be issued by some other application running in the default activation group and, therefore, outside the control of your application.

When an ILE program is run in a named activation group, its files may only be closed by the RCLACTGRP command, or by the use of the RPG IV SETON LR operation just as it functions today for OPM.

7.4 Differences Between Default and Non-Default Activation Groups

Any ILE application design must consider whether programs should be allowed to run in the default activation group or not.

The cause for an ILE program running in the default activation group is ONE of reasons below:

- The program was created with the ACTGRP(*CALLER) attribute, and its caller is running in the default activation group.
- The program is running in compatibility mode.

The default activation group is provided for the running of OPM programs and ILE programs with OPM compatibility mode. You should **NOT** design new ILE programs to run in the default activation group; for a new application you should use a named activation group.

At V3R1, the only way to determine which activation group a program is running is to use DSPJOB. There are no APIs that provide the activation group name to the calling procedure.

We now summarize the main differences between running an ILE *PGM created with ACTGRP(*CALLER) (thus not created with OPM compatibility mode) in the default activation group versus running it in a non-default activation group (for example, user-named or *NEW).

1. exception handling

   If you do not handle exceptions in your application, an extra CEE9901 message is generated when you run the *PGM in the default activation group.

2. Scoping

OVRSCOPE and OPNSCOPE on the OVRDBF and OPNDBF commands default to *ACTGRPDFN. Thus, when you open your files as SHARE(*YES), the file is opened scoped to the call level in the default activation group; it is opened scoped to the activation group level in a non-default activation group.

3. RCLRSC

   This command cannot be run in a non-default activation group. It causes files to be closed for ILE procedures run in the default activation group.

   This command also works differently for ILE compatibility mode programs to ILE non-compatibility mode programs.

4. Static Storage

   All RPG variables are kept in static storage. Once an ILE procedure has been run in the default activation group, the static storage associated with its variables are not returned to the system until end-of-job (the program is not deactivated until end-of-job).

7.5 The Call Stack

The call stack is a list of all OPM program names and ILE procedure names from which your job is currently running a statement. This is known as the *program stack* in OS/400 V2R2 and changed to the *call stack* in OS/400 V2R3.

The only way to run an ILE program is to dynamically call it; within an ILE program there may be 1-many bound procedures. For a dynamic call to an ILE program, we see a PEP on the call stack. For a bound call to an ILE procedure, we see the procedure name on the call stack. A dynamic call to an OPM program causes the program name to appear on the call stack. In Figure 56 we use the DSPJOB command, Option 11 to review the call stack.

```
                                 Display Call Stack
                                                             System:   RCHASM02
 Job:   P23KRZ75D      User:   QSECOFR        Number:   014850

 Type options, press Enter.
   5=Display details

      Request  Program or
 Opt   Level   Procedure         Library     Statement         Instruction
               QCMD              QSYS                             0353
               QUICMENU          QSYS                             00C1
          1    QUIMNDRV          QSYS                             0457
          2    QUIMGFLW          QSYS                             0486
          3    QUICMD            QSYS                             03E5
               _QRNP_PEP_  ...   ITSCID04 1
               CSR1              ITSCID04 2  0000000002
               DJ1               ITSCID04 3  0000000200

                                                                        Bottom
F3=Exit    F10=Update stack    4 F11=Display activation group   F12=Cancel
F16=Job menu   F17=Top   F18=Bottom
```

Figure 56. DSPJOB - Call Stack: Initial Screen

Notes:

1 A dynamic call was issued to an ILE program; hence there is a program entry procedure (PEP) on the call stack. In this example, the PEP is ILE RPG/400 (_QRNP_PEP) so we know that the called program was an ILE RPG/400 program.

Dynamic calls to ILE programs always results in a PEP being placed on the call stack corresponding to the ILE HLL used for the module specified on the ENTMOD keyword of the CRTPGM command.

2 The name of the ILE RPG/400 procedure specified as the PEP is CSR1, thus procedure CSR1 is called after the PEP.

3 Procedure CSR1 performed a bound call to procedure DJ1. We know that CSR1 is an ILE procedure from the next screen Figure 57.

At this point we still do not know either the name of the ILE program whose RPG PEP is on the call stack, or what activation group this program is running in.

We take F11 **4** from the DSPJOB Option 11 (Call Stack) screen in order to see the activation group attribute of our ILE program.

```
                              Display Call Stack
                                                           System:   RCHASM02
 Job:   P23KRZ75D      User:   QSECOFR        Number:   014857

 Type options, press Enter.
   5=Display details

      Request  Program or       ---activation group---
 Opt   Level   Procedure        Number      Name
               QCMD             0000000001  *DFTACTGRP
               QUICMENU         0000000001  *DFTACTGRP
          1    QUIMNDRV         0000000001  *DFTACTGRP
          2    QUIMGFLW         0000000001  *DFTACTGRP
          3    QUICMD           0000000001  *DFTACTGRP
               _QRNP_PEP_  ...  0000000120  NEXTONE  1
               CSR1             0000000120  NEXTONE
               DJ1              0000000120  NEXTONE

                                                                      Bottom
 F3=Exit F10=Update stack   2 F11=Display module  F12=Cancel  F16=Job menu
 F17=Top F18=Bottom
```

Figure 57. DSPJOB - Call Stack: Activation Groups

Note: **1** The activation group in which our ILE RPG/400 program is running is *NEXTONE*.

We still, however, do not know the name of our ILE program; therefore we must take **2** F11=Display Module to see the program name specified against the HLL-specific PEP.

```
                              Display Call Stack
                                                             System:   RCHASM02
 Job:   P23KRZ75D      User:   QSECOFR        Number:   014857

 Type options, press Enter.
   5=Display details

      Request  Program or       ILE         ILE         Control
 Opt   Level   Procedure        Module      program     Boundary
               QCMD
               QUICMENU
           1   QUIMNDRV
           2   QUIMGFLW
           3   QUICMD
               _QRNP_PEP_  ...  CSR1 1      CSR1        Yes 4
               CSR1             CSR1 2      CSR1        No
               DJ1              DJ1  3      CSR1        No

F3=Exit        F10=Update stack   F11=Display statement ID   F12=Cancel
F16=Job menu   F17=Top   F18=Bottom
```

Figure 58. DSPJOB - Call Stack: Modules

Notes:

1 We can now see that it was ILE program CSR1 that was dynamically called.

2 For ILE RPG/400 and ILE CL, this procedure name is always the same as the module name. For ILE C/400, there are many procedures in one module.

3 Notice that there is no PEP to identify the HLL of procedure DJ1 as it was executed by a call bound.

4 Notice the control boundary that resulted from calling the program CSR1 that was created to run in activation group NEXTONE.

7.6 Control Boundary

Please refer to **4** in Figure 56 on page 92 to see an example of a system screen using this new term.

A control boundary is a call stack entry used as the point to which control is transferred when an unmonitored error occurs, or an HLL termination verb is used.

A control boundary acts as a delimiter for:

- A run unit in ILE languages
- ILE exception message percolation

A dynamic call to an ILE program causes a PEP for that program to appear on the call stack. A procedure is a control boundary if one of the following is true:

1. The caller was an OPM program

2. The caller was running in a different activation group

If a procedure is a control boundary and it is also a PEP, then it belongs to an ILE *PGM.

If a procedure is a control boundary and it is not a PEP, then it belongs to an ILE *SRVPGM.

There are two types of control boundaries:

1. Hard control boundary

   If a control boundary is the **first** control boundary in an activation group, then it is a hard control boundary.

   It is not possible to have a hard control boundary in the default activation group; they only occur in non-default activation groups.

2. Soft control boundary

   A control boundary that is **not the first** control boundary in an activation group is called a soft control boundary. This type of control boundary can occur in any activation group.

7.6.1 Control Boundary Example

Which procedures are control boundaries in this example?

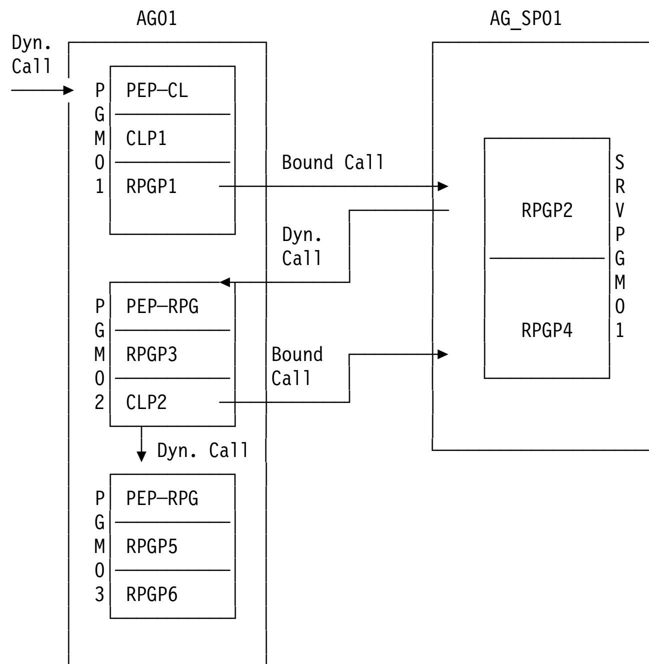


Figure 59. Control Boundary Example

1. The program entry procedure PEP-CL of program PGM01

   The PGM01 PEP-CL procedure is a control boundary as this is the first program in a named activation group. The PEP for PGM01 is invoked through a dynamic call to PGM01; remember that even though a dynamic call to program PGM01 has been issued, it is actually the PEP that is placed

on the call stack. Since program PGM01 was created with CL module CLP1 containing the entry point (ENTMOD), control is now passed from PEP_CL to procedure CLP1.

2. Procedure RPGP2 in service program SRVPGM01 is a control boundary because the (static) caller of this procedure is running in a different activation group.
3. Procedure PEP-RPG in program PGM02 is a control boundary because its caller is running in a different activation group.
4. When procedure CLP2 in program PGM02 calls procedure PEP_RPG for program PGM03 (dynamic call), then the PEP-RPG for program PGM03 is *not* a control boundary since program PGM03 is running in the same activation group as the calling program.

Note: A dynamic call to an ILE *PGM object always results in a PEP for that program being added to the call stack. Contrast this with a static call to a bound procedure, where it is simply the called procedure name that is placed on the call stack (with no PEP).

7.7 ILE Static Call Syntax

An ILE static call is also known as a bound call as it is always used to call a bound procedure. The command or method to call a bound procedure is ILE HLL dependent.

The following table summarizes the dynamic and static call semantics for the V3R1 ILE languages.

Table 19. Dynamic and Static Call Syntax

HLL	ILE Static Call	OPM or ILE Dynamic Call
RPG	CALLB	CALL
CL	CALLPRC	CALL
COBOL	CALL LINKAGE	CALL
C	Function call: procnam()	#PRAGMA LINKAGE (pgmnam,OS)

Note:

1. The CALLB operation code is not allowed in ILE RPG/400 compatibility mode programs.
2. The CALLPRC command is not allowed in ILE CL compatibility mode programs.

7.8 Binding Considerations

A binding directory is a list of module and service program names in a source member to be specified in the BNDDIR parameter on the Create Program command (CRTPGM). These names are only used if they contain an export that satisfies an unresolved import during the binding process.

The binder looks at the modules in a binding directory as strictly optional. It is very likely you want to create service programs binding together modules without any reference to each other. It is important you know that the binder picks up a module from a binding directory only if it provides an export for some currently unresolved imports. Even if you have specified through the binder

language that your service program is exporting a symbol from a module that is in the binding directory, this module is not picked up, unless some module in the service program makes an explicit reference to one of its exports.

As a result of this behavior to create a service program whose modules are completely independent from each other, you have to specify them explicitly in the MODULE parameter of the CRTSRVPGM command.

7.8.1 Exports and Imports

In OPM, when we needed to pass variables from one program to another program the only choice was to pass parameters; this involves system overhead to both pass the arguments to the called program and receive the parameters in the called program.

When a dynamic call (for example, CALL in RPG) is used to an ILE or OPM program, the only way to exchange variables is to pass parameters.

In ILE RPG/400, a new, more efficient method is available, exporting a variable from one RPG procedure and importing it into another.

This new export/import facility in RPG IV is enabled through the use of the EXPORT or IMPORT keyword on a definition specification against the variable you want to exchange.

Throughout this section, we refer to the CRTPGM command when referencing the bind process. While there are other commands providing bind capability, we concentrate on this command because it provides the most flexibility.

What is an export?

An export is an procedure name or variable name in one module that is eligible to be used by a procedure in a *different* module. Thus, the module exporting the variable or procedure is the module that defines it.

What is an import?

An import is a procedure name or variable name used in a module (declared) that does not exist in that module (is not defined in it). The import must, therefore, exist in a different module (within the same *PGM or in an attached *SRVPGM) in order for the program to be created. Thus, the import must be satisfied by a corresponding export (with the definition) in order for the binder (CRTPGM) to create the program object.

It is the binder's job to build a list of exports and then use that list to satisfy imports. This list of exports is built using resources you specified on the CRTPGM command. This export list is built from resources in the following order:

1. Modules specified on the MODULE keyword
2. Modules used from the binding directory
3. Service programs

ILE Exports

There are 2 types of exports within ILE:

1. Implicit export

   All procedures within a *PGM are automatically available (exported) to all other procedures within the same *PGM. Thus, you do not have to code anything special to make an RPG IV procedure available for use in another RPG IV procedure within the same *PGM. Clearly, if during the bind process, the binder determines that you are trying to call a procedure (CALLB in RPG IV) that does not exist within the set of procedures comprising your *PGM, you receive a message saying `there is an unresolved import request`. The system tried to find an implicit procedure export within the *PGM to match the import procedure specified on the CALLB, but none existed.

2. Explicit export

   In RPG IV, you can make both variables and procedures that are defined in a *SRVPGM available for use in either a *PGM, or in another *SRVPGM by:

   a. Coding EXPORT on all variables you want to export (variables must be an internal export before they are candidates for being an external export)
   b. Using binding source and coding EXPORT against the variables or procedures you want to export.

7.8.1.1 ILE Program (*PGM) Binding

An ILE program typically contains more than one procedure. It may or may not have been created with references to service programs. The procedures directly available within a *PGM object are those procedures that are bound by copy into the program. These are:

- In modules coded on the MODULE keyword of the CRTPGM command.
- In modules specified in a binding directory that were used at program creation. The binding directory is specified using the BNDDIR keyword of the CRTPGM command.

  In this instance, these modules are only included in the program because they satisfied an export for an import that was not satisfied from the modules named on the MODULE keyword.

Procedures indirectly available to a *PGM object are those that are bound by reference and, therefore, are contained in a *SRVPGM that is attached to the program.

Modules contained within a service program are bound to the *PGM object by reference. Procedures and variables in service program modules can only be used in the *PGM if you have *made them available* from the service program, that is, exported them from the service program using binding source or (not recommended) EXPORT(*ALL).

The binder tries to resolve exports and imports within modules bound by copy BEFORE resolving them from modules bound by reference (in a *SRVPGM).

Procedure Export/Import

All procedures (bound by copy) into a *PGM object are automatically available (exported) for other procedures (bound by copy) in the same *PGM object to use (import) using bound calls (for example, through CALLB in RPG IV).

You cannot export a procedure name from a *PGM to a *SRVPGM.

Variable Export/Import

You can exchange variables between RPG IV procedures (bound by copy) in the same *PGM object using the EXPORT keyword in the calling procedure (for example, procedure using RPG IV CALLB) and the IMPORT keyword in the called procedure.

The way to pass a variable to or from an ILE CL procedure in a *PGM is by passing parameters.

You cannot export a variable from a *PGM to a *SRVPGM or to a different *PGM; you can, however, export a variable from a *SRVPGM to a *PGM. In order to pass a variable defined in an ILE *PGM to a different ILE *PGM or *SRVPGM, you must pass parameters (for example, use RPG IV *PLIST).

7.8.1.2 ILE Service Program (*SRVPGM) Considerations

Procedure Export/Import

All procedures (bound by copy) into a *SRVPGM object are automatically available (exported) for other procedures (bound by copy) in the same *SRVPGM object to use (import) using bound calls (for example, through CALLB in RPG IV).

You can export a procedure or data from a *SRVPGM by using binding language in a source member for the export specification (or using EXPORT(*ALL), which we do not recommend).

Variable Export/Import

You can exchange variables between RPG IV procedures in the same *SRVPGM object using the EXPORT keyword in the calling RPG IV procedure (for example, a procedure using RPG IV CALLB) and the IMPORT keyword in the called RPG IV procedure.

You can export a variable outside a *SRVPGM by specifying it as EXPORT in the binding source (or using EXPORT(*ALL) which we do not recommend).

The way to pass a variable to or from an ILE CL procedure in a *PGM is by passing parameters.

7.8.2 RPG Initialization Considerations for an ILE *PGM or *SRVPGM

ILE RPG/400 variables are initialized:

1. Upon a dynamic call to a program that is not already activated in the activation group
2. Upon the first bound call to an ILE RPG/400 procedure in the activation group

EXPORT Variables

If you have coded EXPORT against a data item in the RPG IV definition specification, then that field is only initialized by ILE upon a dynamic call to the ILE program in which it is defined. Unless the activation group is ended, this field never is reinitialized by the system, even if you SETON LR in the procedure in which it is defined. This means that even after LR, the field contents remains unchanged if the program is called again in the same activation group.

You are responsible for coding any re-initialization of RPG EXPORT fields in your application.

7.8.2.1 RPG IV Export and Import Example

Figure 60 on page 101 shows an example for exporting and importing data *and* procedures. Additionally you can see the capability of creating an interface to the service program SPGM01 through binding language.

Working Through the Example

To create the service program SRVPGM01, we could use the command:

```
CRTSRVPGM SRVPGM(SPGM01) MODULE(SP1 SP2 SP3 SP4 SP5)
          EXPORT(*SRCFILE) SRCMBR(SPGM01)
```

where the binding source for member SPGM01 of binding source file QSRVSRC is:

```
STRPGMEXP
EXPORT     SYMBOL(Y)
EXPORT     SYMBOL(SP4)
ENDPGMEXP
```

These few statements define the so-called *public interface* to service program SPGM01.

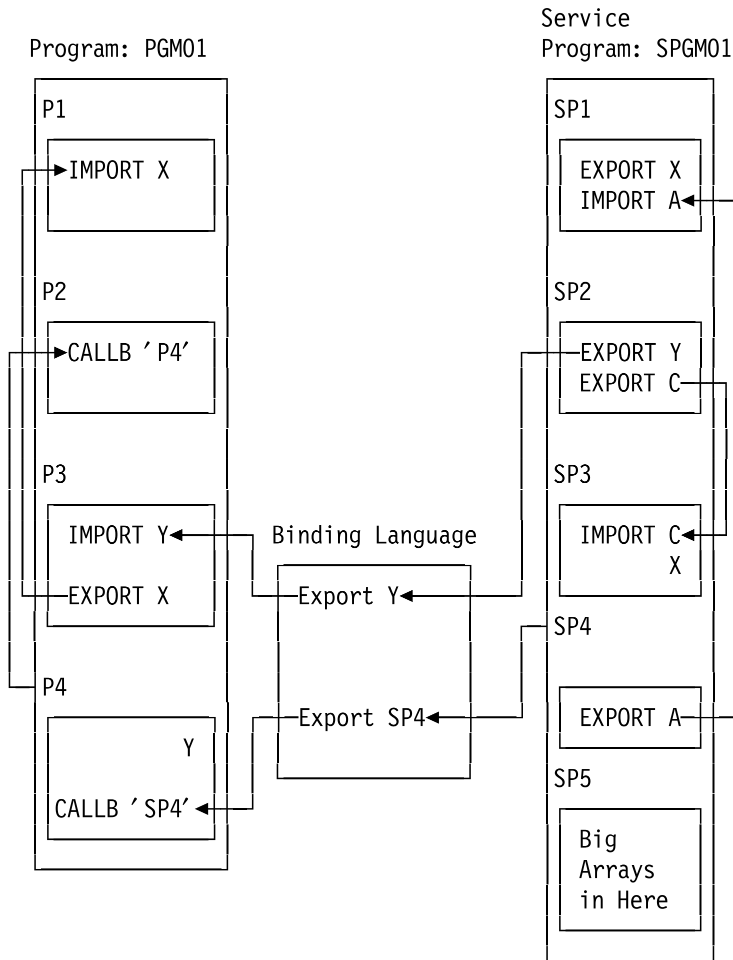


Figure 60. Export and Import Relationship Example

To create program PGM01, we would use the command:

```
CRTPGM PGM(PGM01) MODULE(P1 P2 P3 P4) BNDSRVPGM(SPGM01) ACTGRP(NOT_NEW)
```

We will work through this example, starting with the service program SPGM01. Remember that it is the binder (CRTPGM) that resolves all the imports and exports between modules.

- Procedure SP1

  Variable X is exported. Since there is no IMPORT X coded in other modules in the *SRVPGM and we cannot import variables from a *PGM object, we conclude that this variable has been specified as export for future use. Thus, at some point we will add a module to the service program that needs to import X from procedure SP1. Note that this variable X has nothing to do with the variable X coded in program PGM01 or the variable X coded in procedure SP3.

  Variable A is imported. At bind time, the binder first checks for exports within the *SRVPGM to satisfy this import. It finds that variable A is exported (defined) in procedure SP4.

- Procedure SP2

  Variable Y is exported. While there are no local imports within the service program needing this export, there is an EXPORT Y coded in the binding source for this *SRVPGM; thus we know that this variable is exported to a *PGM that is bound to this service program.

  Variable C is exported and is satisfied by an import in procedure SP3.

- Procedure SP3

  Variable C is imported from procedure SP2.

  Note that this procedure uses a variable X. This is shown without the use of EXPORT to denote that it is private to procedure SP2. In C programming language, this is known as a *local automatic variable*, since this X is known only within the procedure that defines it.

- Procedure SP4

  Variable A is exported and is satisfied by an import in procedure SP1.

- Procedure SP5

  This procedure is included in SPGM01 for future use. Currently, it is not referred to by any other procedures in the *SRVPGM nor is it exported to PGM01 using the binding source.

  We have indicated that there are large arrays inside SP5. While this procedure is not referenced anywhere, when we call program PGM01, the static storage used for these arrays in SP5 is still initialized. This initialization uses CPU resources and is unnecessary in this example. Thus, we could improve the performance of PGM01 by removing procedure SP5 from SPGM01. We have deliberately included SP5 to illustrate a common mistake in ILE. **We do NOT recommend that you code your application this way.**

- We have binding source to control what variables and procedures from SPGM01 are used by other *PGMs and *SRVPGMs. In this example, only variable Y in procedure SP2 and procedure SP4 is used outside SPGM01.

We can now work our way through the ILE program PGM01:

- Procedure P1

  Variable X is imported from procedure P3.

- Procedure P2

  A bound call is made to procedure P4. Since this procedure is bound by copy to PGM01, we do not bother to look in the service program.

- Procedure P3

  This procedure imports variable Y. While there is a Y in procedure P4, the binder ignores it as it does not have an export coded against it. Since there is no EXPORT Y coded in PGM01, the binder now searches the binding source for the service program SPGM01. Since Y is specified in the binding source and EXPORT Y is specified in procedure SP2 in SPGM01, we have now found an export in SPGM01 for our import of Y in PGM01.

  Variable X is exported to procedure P1 in PGM01. Note that we cannot export variables or procedures from a *PGM to a *SRVPGM.

- Procedure P4 has an import (bound call) to procedure SP4.

Note that there is a variable Y used in this procedure. Since it does not have an EXPORT or IMPORT coded against it, it does not have anything to do with any other instances of variable Y outside P4.

The binder first checks whether there is a procedure SP4 bound by copy into program PGM01. None is found. Next, procedures in SPGM01 bound by reference and exported using binding source are searched. We find SP4 is exported from SPGM01 in the binding source, thus we have satisfied this import.

7.8.3 Unresolved References

When you develop a large modular application, you might need to test it even if all the functions have not been implemented. This is accomplished by using the **UNRSLVREF* value for the creation option parameter. The program or service program is created even if there are symbols that have not been resolved. Any runtime reference to those symbols causes an MCH3203 error to occur.

Once the missing symbols are available, you must re-create the program to eliminate the unresolved references. Only the binder listing and the job log of the job where the bind has been run can provide information about references (unresolved or resolved) in your programs.

7.8.3.1 Circular References

In some cases, you might have circular references among service programs, for example, if program X exports symbol A and imports symbol B and, at the same time, service program Y imports A and exports B. If possible, you should combine the modules in larger service programs to avoid service programs referencing each other. This recommendation suits both, performance and a consistent application design.

If you cannot avoid circular references, you have to create your service program by using a special option on the CRTSRVPGM. The unresolved reference (*UNRSLVREF) value for the OPTION parameter provides a way to create your service program without resolving imported procedures or variables. The service program is created with its public interface, that is the exported procedures and variables are available for referencing. You would then create your other service programs and resolve their references. Once those service programs are created, you can re-create the one that has unresolved references.

Given programs X and Y referencing each other with symbols A and B, you would use the binder source language to specify that service program X exports symbol A and creates it with *UNRSLVREF:

```
CRTSRVPGM SRVPGM(X) MODULE(X)
          EXPORT(*SRCFILE) SRCFILE(QSRVSRC) SRCMBR(YOURNAME)
          OPTION(*UNRSLVREF)
```

Now you can create service program Y, using the binder language to notify that you are exporting symbol B:

```
CRTSRVPGM SRVPGM(Y) MODULE(Y)
          EXPORT(*SRCFILE) SRCFILE(QSRVSRC) SRCMBR(YOURNAME)
```

You can now re-create service program X, resolving the imports provided by service program Y:

```
CRTSRVPGM SRVPGM(X) MODULE(X) BNDSRVPGM(Y)
          EXPORT(*SRCFILE) SRCFILE(QSRVSRC) SRCMBR(YOURNAME)
```

Remember that calling an entry point in either X or Y causes the activation of the other service program, even if it is not strictly needed, with a performance drawback.

The double creation process only takes place once.

7.8.4 Service Program Signature

The Integrated Language Environment Binder supports the creation of two types of bound program objects; programs (*PGM objects) and service programs (*SRVPGM objects). As described in the *Integrated Language Environment Concepts* manual, SC41-3606, service programs differ in a couple of significant ways from regular programs, one of those differences being that only service programs have signatures. A service program signature is fairly analogous to the *level check* values that are associated with *FILE objects. The intent of both mechanisms is to provide a way of detecting when the user of an object is *out of synchronization* with the level of the object being used.

A system-generated signature is 16 bytes long and is stored as part of the service program object. With OS/400 Version 3 Release 1 the keywords LVLCHK(*YES|*NO) and SIGNATURE(*GEN|′string′) have been added to the STRPGMEXP command. This allows you to optionally perform a level check and to specify a signature string.

A service program has one *current* signature and may contain zero or more *previous* signatures. When the binder creates a bound program that references (or imports) a variable or procedure that is defined (or exported) by a service program, the binder creates a record in the bound program, saving the name and library and *current* signature of the service program. Before a program is run, it must go through an activation step. Generally, the activation step is done transparently as part of the machine processing when a program is called (for example, the CL CALL command). During activation of an ILE bound program, any service programs needed by the bound program are also activated. When activating the service program, the signature in the program is compared with the signature or signatures in the service program. If the signature of the program matches any of the signatures in the service program, activation continues. If no signature match is found, a *signature mismatch* exception is signalled, and program activation ends (that means the bound program does not run).

The developer of a service program has to exercise great care when changing any service program exports, since the algorithm used to generate the signature is sensitive to the names of the exported data and procedures, including the order of the names. For example, adding a new EXPORT would cause the signature to change; as would deleting an EXPORT, changing the EXPORT name, or changing the order of EXPORT statements. Sometimes the signature would change even though the service program was being changed upward compatibly (for example, by just adding a new exported procedure). The developer of a changed service program needs to know all of the program and service program objects that are bound to the service program. Those depending program and service program objects have to be re-bound with the updated service program in order to avoid *signature mismatch* conditions at activation time.

7.8.5 Service Program Recommendations

The following list explains some recommendations you should consider when developing service programs:

- Use binder control source statements (that is, STRPGMEXP, EXPORT, and ENDPGMEXP) instead EXPORT(*ALL) on the CRTSRVPGM command.
- If you retain previous signatures to ensure upward compatibility for applications bound to a service program, keep the following rules for system-generated signatures in mind:
  - The order in which the modules are processed.
  - The order in which the symbols are exported from the copied modules determine the signature for service programs.
- Avoid a service program signature change through program fixes.
- Add new exports at the end of the binder source (that is just before the ENDPGMEXP statement).
- Avoid reordering or removing existing EXPORT statements in your binder source.

7.8.6 Updating Programs without Re-binding

An ILE program is generally made of many different modules bound together and very likely the application developer does not ship all the modules objects to the customer. When a module is changed, re-binding the entire application might be very cumbersome. That is why OS/400 provides the way to update a program or a service program without any need to re-bind starting from the modules. Two commands, *UPDPGM* and *UPDSRVPGM*, allow you to substitute an old module for a new one.

The use of these two commands is particularly suitable in two situations:

- You want to update a specific part of a complex program without reshipping the entire program or all the modules.
- You want to debug a specific module at the customer site and debug information is not in the production version of the application.

The system allows you to replace a module with any other module of the same name, unless you introduce new unresolved references. If the new module has new imports, the system tries to resolve them and eventually aborts the update if the imports cannot be found. You can even replace a module with a newer version having fewer imports. It might happen that some modules in the program (or service program) result in not being referenced. We call these modules ″marooned″ modules. With the OPTION(*TRIM) on both update commands, you get rid of the marooned modules.

If you replace a module with another having fewer exports, the re-creation does not fail if the missing exports are not needed for re-binding, or if they are not referenced by the EXPORT list in a service program re-creation. However, if you specify EXPORT(*ALL) for a service program, the signature changes.

If the replacing module has more exports, EXPORT(*ALL) again changes the signature of a service program. Pay attention to the situation where the new exports are duplicate procedures; that is, they are already defined in some other module. If the module being replaced is positioned before the module with the

old exports, the new exports win and very likely the behavior of your application changes.

7.9 Resource Scoping

The default in OPM for database and device file resources is that they are available to the call stack level that created them. If we take explicit action, say against a database file by use of the command:

```
OVRDBF FILE(X)..SHARE(*YES)
```

or we have created file X with SHARE(*YES), then any programs that open file X after the override has been issued uses the same ODP. We could also say that the ODP for file X exists at the call level and is visible to any program called after that call level. The override has been used with the file open to change the scoping of the file open.

The scope of a resource determines:

- Its visibility, meaning, whether other procedures or programs running in the job can use it. If a resource is not seen, then it is not used.
- Its existence, meaning, when the resource is cleaned-up or deleted.

Additional information on resource scoping and overrides is found in the publication *Data Management*, SC41-3710.

7.9.1 Overrides and File Opens

When opening a file overridden with *share open data path* SHARE(*YES), the system tries to find an existing ODP:

1. At any call level if you are in the default activation group.
2. In the same activation group if you are in a non-default activation group.
3. At the job level.

You can directly influence where the ODP is created by using the OPNSCOPE keyword on the OVRDBF...SHARE(*YES) command. Thus, an ODP that is opened scoped to activation group level is not seen or used by a program in a different activation group.

Warning - Job Level Scoping - Use Carefully!

This is the most inflexible choice and should not be used within an ILE application. If an AS/400 system runs applications from multiple vendors, and these applications are accessible within the same OS/400 job, then job level scoping should *not* be used.

Consider banning the use of OVRDBF...OVRSCOPE(*JOB) SHARE(*YES) within your organization. If coded, this may adversely affect any other applications that run concurrent with yours within a job.

7.9.1.1 File Open Scope

The scope of a file′s Open Data Path (ODP) affects which programs **are eligible** to use the ODP. It does not directly cause these eligible programs to use the ODP. You must use OVRDBF...SHARE(*YES) to cause eligible programs to share the same ODP.

The default scope capability on OPNxxxF command is:

```
OPNxxxF ... OPNSCOPE(*ACTGRPDFN)
```

Possible OPNSCOPE values are:

*ACTGRP
*ACTGRPDFN
*JOB

Use of OPNSCOPE(*ACTGRPDFN) means ″it depends″:

- When running in the default activation group,
  then the ODP is scoped to the call level.
- When running in a non-default activation group,
  then the ODP is scoped to the activation group level.

7.9.1.2 Override Scope

The scope of an override affects which programs see the override and thus use it before they open any files.

The default scope capability on the OVRxxxF command:

```
OVRxxxF ... OVRSCOPE(*ACTGRPDFN)
```

Possible OVRSCOPE values are:

*CALLLVL

*ACTGRPDFN

*JOB

Use of OVRSCOPE(*ACTGRPDFN) means ″it depends″:

- When running in the default activation group,
  then the override is scoped to the call level.
- When running in a non-default activation group,
  then the override is scoped to the activation group level.

Thus, when running in a non-default activation group, you would scope the file open to the activation group and make it available to all other programs within the activation group using:

```
OVRDBF SHARE(*YES)
OPNDBF FILE(myfile) OPTION(option)
```

Notice the coding for ILE mirrors the coding for OPM; if you use the defaults, then the system automatically scopes to the activation group level for ILE.

V3R1 OVRxxxF Default Change

The override scope defaults have changed between OS/400 V3R1 and V2R3. This is particularly important for developers using ILE C/400 at V2R3, and for people migrating to ILE at V3R1.

V3R1 : OVRxxxF ... OVRSCOPE(*ACTGRPDFN)
V2R3 : OVRxxxF ... OVRSCOPE(*CALLLVL)

The possible override scope values are:

V3R1 : OVRxxxF ... OVRSCOPE = *CALLLVL or *ACTGRPDFN or *JOB
V2R3 : OVRxxxF ... OVRSCOPE = *CALLLVL or *JOB

7.9.1.3 Override Rules

There is only *one* override per call, activation group or job level applied to the file.

If multiple overrides of the same type (call/activation group/job) are issued for the same file at the SAME call stack level, then only the last one is used; any others issued at the same level are completely replaced.

Call level overrides are merged together in last-in-first-out sequence from the call stack.

Follow the six basic rules to determine how a file is opened:

1. Identify the ACTGRP that you are running.

   Ignore this step if you are running in the default ACTGRP.

2. Identify the highest call stack entry (lowest numeric) for the ACTGRP in which you are running (the oldest procedure that is on the call stack for this ACTGRP). Assume the call stack number you identified is N.

   Ignore this step if you are running in the default ACTGRP.

3. Process (merge) all *CALLLVL overrides at a call stack level that is lower than or equal (numerically greater than or equal) to call level N **irrespective** of ACTGRP.

   Ignore this step if you are running in the default ACTGRP.

4. Process any activation group level override for the activation group in which the open occurred.

   Ignore this step if you are running in the default ACTGRP.

5. Process (merge) any remaining *CALLLVL overrides at higher call stack levels (less numerically) than call level N.

6. Process *JOB level overrides.

7.9.2 Override Example

The example in Figure 61 on page 109 is deliberately complex in order to represent most situations that could occur. We do not, however, expect most ILE applications to apply all of the override options available against one file.

The sequence of how overrides are applied to an object is:

```
Call    OPM/   ACTGRP
Level   ILE    Number   Operation Executed

  2     OPM    *DFTAG   OVRPRTF FILE(YYY) FOLD(*YES) OVRSCOPE(*CALLLVL)
  3     ILE    000008   OVRPRTF FILE(ZZZ) TOFILE(YYY) DEV(P1)
                                LPI(6) OVRSCOPE(*CALLLVL)
  4     OPM    *DFTAG   OVRPRTF FILE(ZZZ) CPI(12) OVRSCOPE(*CALLLVL)
  5     ILE    000021   OVRPRTF FILE(YYY) DEV(P2) OVRSCOPE(*JOB)
  6     ILE    000021   OVRPRTF FILE(ZZZ) LPI(12) OVRSCOPE(*ACTGRPDFN)
  7     ILE    000008   OVRPRTF FILE(ZZZ) LPI(9) OVRSCOPE(*CALLLVL)
  8     ILE    000008   OVRPRTF FILE(ZZZ) DUPLEX(*NO)
                                OVRSCOPE(*ACTGRPDFN)
  9     ILE    000008   OVRPRTF FILE(YYY) LPI(5) OVRSCOPE(*ACTGRPDFN)
 10     ILE    000008   OPEN FILE(ZZZ)
```

Figure 61. Override Example

7.9.2.1 Override Example Solution

What is the result of the open all call level 10?

The open results in the following:

`FILE(YYY) DEV(P2) LPI(5) FOLD(*YES) CPI(12)`

Here is the step-by-step solution to the example:

1. At level 10, we are opening file ZZZ. So we first look for an override at level 10 for file ZZZ. There are none.
2. Move to the next lower call level and again look for an override. The override at level 09 is not for file ZZZ, so we ignore it and continue.
3. At level 8, we find an override and it is for file ZZZ. However, it is an activation group override (determined by OVRSCOPE). Therefore, we wait until we process all overrides with call levels greater than or equal to the call level of the oldest procedure in the activation group in which the open is occurring. In this case, the activation group the open is in, is 000008, and the oldest procedure for activation group 000008 is at call level 3. So, we wait until we process all call level overrides greater than or equal to level 3 before we process this one. Continue on.
4. At level 7, we find an override for ZZZ, and it is a call level override, so we process it. We pick up the attribute LPI(9) for file ZZZ. Continue on to the next lower level.
5. At level 6, we see another override for ZZZ. We do not process it because it is an activation group level override for a different activation group. We ignore this override completely.
6. At level 5, there is an override, but not for ZZZ. Move on to level 4.
7. At level 4, we find a call level override for ZZZ, so we process it. Now we pick up CPI(12) in addition to LPI(9). Continue on.
8. At level 3, we again find a call level override for file ZZZ, so we process it. Notice that this involves file redirection. This means that we are no longer opening file ZZZ, but instead we are opening file YYY. We also pick up

DEV(P1) and we replace LPI(9) with LPI(6). Remember, the lower the call level, the higher the priority. Continue on.

9. It is now time to process any activation group level overrides, because we have processed all call level overrides greater than or equal to the oldest program or procedure in the activation group. Due to the file re-direction, we are looking for any activation group level overrides for file YYY. At level 9, there was an activation group level override issued, specifying LPI(5). So at this point we have FILE(YYY) LPI(5) CPI(12) DEV(P1). Now we continue with call level overrides.
10. At level 2, we find a call level override for YYY, so we process it. We pick up FOLD(*YES). Continue on to level 1.
11. There are no overrides for call level 1, so we are finished with call level processing. All we have left is the job level.
12. At level 5, we see a job level override and it is for YYY, so we process it. Job level has the highest priority of any override, so we replace DEV(P1) with DEV(P2).
13. FINISHED...... The final result is:
    - CPI(12) from level 4
    - FILE(YYY) from level 3
    - LPI(5) from the AG level override issued at call level 9
    - FOLD(*YES) from level 2
    - DEV(P2) from the job level override issued at call level 5

7.10 Transparency

With the ILE, a new facility called *transparency* was added to certain system facilities. This new facility is best explained by an example.

The ILE CL source (type CLLE) shown in Figure 62 was used to create ILE *PGM EX1C. This source is contained in file QDESIGN in library GG244358. To install library GG244358, refer to Appendix A, “Diskette Install Instructions” on page 177.

```
/* CL program to highlight transparency +
   - performs an override SHARE(*YES) on file PF01      +
   - DSPJOB does not show the override (transparency off) +
   - DSPOVR does show the override (transparency on) */
             OVRDBF     FILE(PF01) SHARE(*YES)
             OPNDBF     FILE(PF01) OPTION(*INP)
             DSPJOB
             DSPOVR
             CLOF       OPNID(PF01)
             DLTOVR     FILE(*ALL)
```

Figure 62. Transparency Example

Since we only have one module, we used the command:

```
CRTBNDCL PGM(EX1C) DFTACTGRP(*NO) ACTGRP(APP_EX1C)
```

If you want to try this example, you should create a physical file (CRTPF) called PF01 in QTEMP. Note that there is only one QTEMP library in a job, thus it now acts as a repository for all applications that run in the same job. When the program EX1C is called, we first see the DSPJOB screen and take option 15 Display File Overrides.

```
                       Display All File Overrides

 Job  . . :   P23KRZ75D      User . . :   QSECOFR       Number . . :   014859
 Call level . . . . . . . . . . . . . :   *

 Type options, press Enter.
   5=Display override details

 Opt  File          Level   Type  Keyword Specifications


   (No file overrides)
```

Figure 63. DSPJOB - Display File Overrides

There are no overrides shown because the system program used to show overrides from DSPJOB does **not** use transparency. Since this system program runs in the default activation group, it only looks in the default activation group for any overrides. It cannot look in a user activation group, because it has not been told to look outside the activation group in which it is running (be transparent) to fulfill the request.

If you want to see all the overrides, you must use the DSPOVR command that has transparency turned on. You should also take F11-All File Overrides option, available only when overrides exist, in order to see if there are any other overrides that might overlay yours (such as one at the job level).

```
                       Display All File Overrides

 Call level  . . . . . . . . . . . :   *

 Type options, press Enter.
   5=Display override details

 Opt  File           Level   Type  Keyword Specifications
      PF01        *ACTGRP   DB    SHARE(*YES)
```

In order to set up an interactive test environment, it is often necessary to manually enter the File Override or File Open commands and then call programs. The additional consideration in the ILE environment is that if your program runs in a named activation group, then you must ensure that you access an IBM command processing routine that has transparency. Please be aware that the QCMD routine used to interactively enter CL commands for OPM application testing does **not** have transparency. Thus, all commands entered following a call to QCMD are run in the default activation group. The list below has been created to help you perform better problem determination.

- What activation group is the procedure in? Use SYSREQ Option 3 (DSPJOB) to determine the activation group you are currently in; look at the call stack entries.

  If your program was created with ACTGRP(*CALLER), be careful that it is not running in the default activation group by *mistake* and thus behaving differently to how it behaves in a non-default activation group.

- If your program works (does not issue errors) but does not work as designed, or if it runs in a named activation group, you must issue a RCLACTGRP command to delete the named activation group and deactivate any programs associated with that activation group.

  Problem scenario:

  Program PGM01 runs in activation group AG_PGM01. It does not issue errors, but you run it and find it does not function as designed. You recompile PGM01, with the changes and then call PGM01 again. The changes you made are not reflected.

  The compile has changed the pointer for program PGM01 in activation group AG_PGM01 to point to program PGM01 in QRPLOBJ. Clearly, you must use RCLACTGRP before retesting the recompiled program.

In summary, if you cannot recall the exact sequence of steps already taken and you want to retest, you should issue a RCLACTGRP from outside the test activation group (ensuring that you have returned back up the call stack to be outside the test activation group rather than performing a call and thus leaving one or many test activation group procedure or procedures on the call stack.

7.11 Ending an ILE Program

We will now review the RPG program termination options available within ILE. Also shown in Table 20 is the effect that the Reclaim Resource command has on ILE programs. In order to end a program or exit the application (leaving it active for later use), the end program operations of RETURN or SETON LR are used, just as they were in OPM.

Table 20. ILE Program Termination

	Default AG **OPM *PGM**	**Default AG** **ILE Compatibility Mode *PGM**	**Default AG** **ILE *PGM (Not Comp. Mode)**	**Non-Default AG** **ILE *PGM**
RETURN	Static storage still assigned Files left open	Static storage still assigned Files left open	Static storage still assigned Files left open	Static storage still assigned Files left open
SETON LR	Static storage returned to system Files closed	Static storage returned to system Files closed	Static storage still assigned but flagged for re-initialization Files closed	Static storage still assigned but flagged for re-initialization Files closed
RCLRSC	Static storage returned to system Files closed	Static storage returned to system Files closed	Static storage still assigned but flagged for re-initialization Files closed	n/a

Note: n/a = not applicable

7.12 Ending an Application

In ILE, there is a distinct difference between exiting an application and ending an application. Exiting an application means that you return control up the call stack to a point outside the application's activation group or groups, leaving the application activation group or groups, and all associated resources still *ready to go*.

Ending an application means that you are not using that application again in the job; thus you want to totally close down all activation group or groups associated with the application, closing down all its files and deactivating all its programs.

Note that there is no difference between ending and exiting an application running in a *NEW activation group; the activation group is deleted as soon as a control returns up the call stack to the caller of the application. Please refer to the section 7.3.5, "Activation Group Recommendations" on page 91 for further details.

Before examining how to end an application in ILE, we will first review how applications are typically ended in OPM.

7.12.1 OPM RPG Application Example

The OPM Order Entry application in Figure 64 on page 114 consists of three programs, OE-1 that calls OE-2, that calls OE-3. Without ILE run unit support, when the user decides to end the Order Entry application, OE-2 must call OE-3 to tell OE-3 to SETON LR, such that OE-3 ends. OE-3 then ends and returns control to program OE-2. Since it was within program OE-2 that ending the application was initiated, program OE-2 now ends with LR on and returns control to program OE-1, instructing program OE-1 to end with LR on. Program OE-1 then ends with LR on and returns control to the menu program.

Note that since ending the application was initiated within program OE-2, it was necessary to perform an extra dynamic call to program OE-3 in order to close down this program.

The way described would be the *proper* way to clean up resources used by an application.

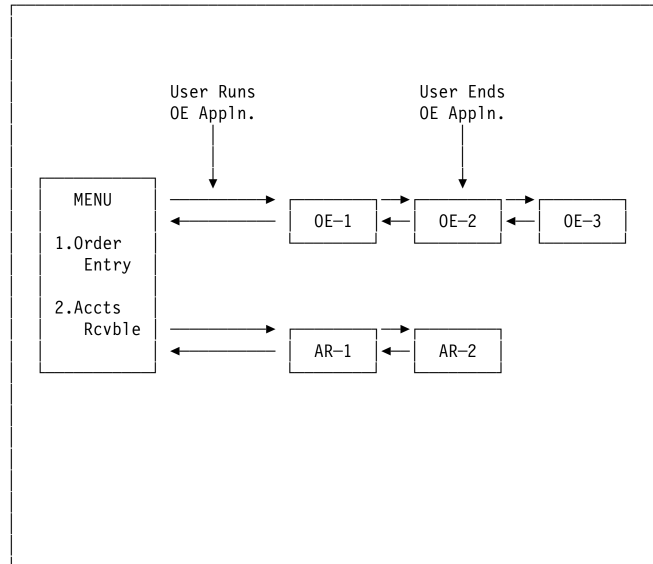


Figure 64. Example - Ending an OPM RPG Application

In some cases, there has been misuse or inefficient use of the RCLRSC command to clean up *dangling programs* retrospectively, since the application is not designed to perform clean-up.

Some applications have adopted an approach of using RPG RETURN everywhere, and then simply using RCLRSC at a higher call stack level. This lack of attention to ending an application can cause performance problems, not just in the job issuing the RCLRSC but for all jobs using files closed by the RCLRSC command.

7.12.2 ILE RPG/400 Application Example

Figure 65 on page 115 shows an application control benefit available in ILE that is similar to run unit control. It is one way to use activation groups to help make application ″clean up″ easier.

You recall that an activation group is a kind of ″sub job″ with its own overrides, file opens and commit cycles.

A run unit concept has been a part of the COBOL language standard and of COBOL/400 in the past. With ILE, other ILE languages can also benefit from the run unit concept.

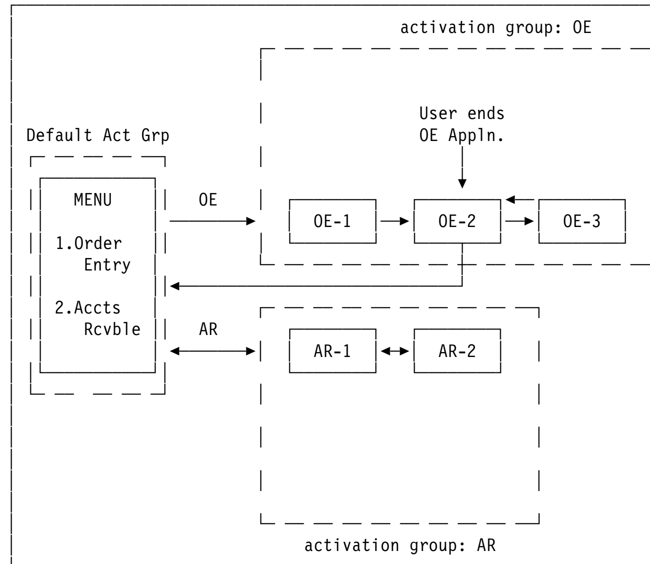


Figure 65. Example - Ending an ILE RPG/400 Application

From the menu, a user chooses to run the Order Entry (OE) application.

With run unit support, the order entry application is set up as its own run unit by specifying that it run in its own activation group (OE). Then when the user chooses to exit the application from OE-2, the OE-2 program can run an operation that exits the entire OE application, closing up files and cleaning program activations for the entire run unit. This simplifies the coding required for application exit operations. This new operation might be compared to the LR in an RPG program, except that this new operation (in RPG, the CEETREC bindable API enables a normal run unit termination) acts as a ″giant LR″ -- ending not only the program, but also the application.

The mechanism used to define the boundaries of a run unit is called a control boundary within an activation group. A control boundary serves as a delimiter for an application run unit, such that the boundary of the application run unit is then defined. Please refer to 7.6, “Control Boundary” on page 94 for more details on control boundaries.

All the programs in an application are created specifying the option that they run in a specific Activation Group. Thus a run unit spans an entire activation group *if* there is only one control boundary in the activation group.

7.12.3 Ways of Ending an ILE Application

We assume that you have coded your ILE application to run in a named activation group. There are three ways of ending an ILE RPG/400 application.

- End application normally and exit to caller of application.

  Use the *end run unit normally* bindable API, CEETREC.

- Use RETURN, SETON LR or a mixture of both in all programs on the call stack, such that control is returned to the caller of the application.

  You should issue the RCLACTGRP ACTGRP(name) command to delete the application′s activation group.

- Exit the application, signalling that ending the application was abnormal, and return to the caller of the application.

  Use the *end run unit abnormally* bindable API, CEE4ABN if an exception your application considers fatal has occurred.

  You may have to issue a RCLACTGRP ACTGRP(name) to delete the application's activation group.

Once your application running in activation group APP_ag1 has ended, you should clean up the activation group using the RCLACTGRP ACTGRP(APP_ag1) command. This will (providing there are no programs/procedures on the call stack for the activation group):

- Close all open files scoped to the activation group.
- Return static storage to the system for all programs and procedures called within the activation group, thus deactivating these programs.
- Delete the activation group.

When ending an ILE application running under commitment control, it is necessary to ensure that any uncommitted database changes are correctly handled, activated programs are ended, and the activation group is deleted. You should not consider following these steps if the activation group is needed later in the same job.

Table 21 (Page 1 of 2). Effect on Resources, Depending on the Way of Ending a Procedure at a Stack Level Closest to a Hard- or Soft Control Boundary

AG	Ending a procedure	HCB			SCB		
		AG	PGM	File	AG	PGM	File
*NEW	Return	Gone	Gone	Close	-	-	-
	LR	Gone	Gone	Close	-	-	-
	Hard Leave	Gone	Gone	Close	-	-	-
Named	Return	Stays	Stays	Open	Stays	Stays	Open
	LR	Stays	Re-init	Close	Stays	Re-init	Close
	Hard Leave	Gone	Gone	Close	Stays	Stays	Open
	Hard Leave + RCLACTGRP	-	-	-	Gone	Gone	Close

Table 21 (Page 2 of 2). Effect on Resources, Depending on the Way of Ending a Procedure at a Stack Level Closest to a Hard- or Soft Control Boundary							
AG	**Ending a procedure**	**HCB**			**SCB**		
		AG	**PGM**	**File**	**AG**	**PGM**	**File**
***DFT**	**Return**	-	-	-	-	Stays	Open
	LR	-	-	-	-	Re-init	Close
	Hard Leave	-	-	-	-	Stays	Open
	Hard Leave + RCLRSC	-	-	-	-	Re-init	Close

Explanation:

This table applies to ILE mode programs only. When run in the default activation group (*DFT) and ending with **Hard Leave + RCLRSC**, this means that after return in an OPM call stack, a RCLRSC is used. When run in a named activation group and ending with **Hard Leave + RCLACTGRP**, this means that after return in the previous activation group, a RCLACTGRP ACTGRP(named) is used.

Gone Activation group is deleted or program is no longer active
Close File is closed
Stays Activation group is still active, program is still active
Open File is still open
Re-init Program is still active, but static storage is initialized upon next call entrance

In order to directly view the effects of the ILE end options, please refer to example 7.12.3.3, “Example - Ending ILE Applications and Procedures” on page 118.

Issue a HARD LEAVE

When a procedure issues a hard leave, if the nearest control boundary is a HARD CONTROL BOUNDARY (thus there is only one control boundary in the activation group), this will:

- Close all open files scoped to the activation group.
- Return static storage to the system for all programs and procedures called within the activation group.
- Delete the activation group.

Issue a RCLACTGRP

This is the ILE equivalent to the OPM RCLRSC command. It should not be overused as it is a heavy resource-user since it performs work similar to RCLRSC. If you specify RCLACTGRP *ELIGIBLE, then all ILE activation group or groups in the job that do not have procedures on the call stack is deleted.

We recommend that you control clean-up in your application by using RCLACTGRP ACTGRP(myactgrp) to clean up individual named activation groups, rather than using the ACTGRP(*ELIGIBLE).

End of Job

This is the heaviest resource consumer and should be avoided, where possible.

7.12.3.1 CEETREC - Normal Termination

This bindable API functions the same as a STOP RUN in COBOL in that the run unit in which the API is called is ended. If the nearest control boundary is also the first control boundary in a named activation group (a HARD CONTROL BOUNDARY) and this API is called, then static storage and ODPs for this activation group is freed and then the activation group is deleted.

Note that this API causes a NORMAL ending of a run unit. Thus, if the application runs under commitment control, the activation group deletes any pending database changes that are committed.

Calling this bindable API is simple as it is a bound call with optional variables. Thus, in RPG IV a call involves:

```
 C                   callb     'CEETREC'
```

For more information on this and other bindable CEE APIs provided with the ILE refer to the publication *System API Reference*, SC41-3801.

7.12.3.2 CEE4ABN - ABNormal Termination

The abnormal termination functions nearly the same as the normal, except that:

- Uncommitted changes are rolled back.
- A CEE9901 *Application error* exception message is sent to the caller of EVERY control boundary until either the message is handled or the application is ended.
-

```
         C                   callb     'CEE4ABN'
```

Note:

Message CEE9901 and other CEE messages are found in message file QCEEMSG in library QSYS.

7.12.3.3 Example - Ending ILE Applications and Procedures

Purpose

This example clearly demonstrates how the different end ILE application options work in practice.

You are able to cause procedures ET02, ET04 and ET06 to end with one of the following options:

1. RETURN - end procedure
2. SETON LR - end procedure
3. Normal End Run-Unit (CEETREC) - end run unit
4. Abnormal End Run-Unit (CEE4ABN) - end run unit

How to Run the Example

The library GG244358 is shipped on diskette with this publication. Restore this library using the instructions in Appendix A, “Diskette Install Instructions” on page 177. This library contains all source and program code for this ILE example. Please see member DESEX1 in source file GG244358/QDESIGN for the commands used to create the ILE modules and ILE programs in this example.

Run the example by entering the commands:

```
ADDLIBLE GG244358
RCLACTGRP *ELIGIBLE
CALL ET01
DSPJOB
```

We suggest that you use the same end option (for example, return) in each of the three ILE procedures ET02, ET04 and ET06.

Notice that when you use return in all of the procedures, both activation groups END and AG01 are still present in the job (all programs activated in these activation groups are still there, with their static storage assigned; there is no way to see this on the system) and all files are still open.

If you use the *end run unit* option in all of the procedures, then all programs are deactivated, all files are closed and the activation groups are deleted.

We now suggest you use a combination of RETURN and CEETREC (NORMAL end of run unit).

Purpose of Example

To illustrate what happens when you take different ILE procedure end options, please use the DSPJOB command to review activation groups, open files and the program call stack.

When control returns from program ET01, please ensure you issue the DSPJOB command to see whether activation groups END and AG01 are still present in the job or not.

Diagrammatically, the programs and procedures that you are calling are now shown in Figure 66 on page 120.

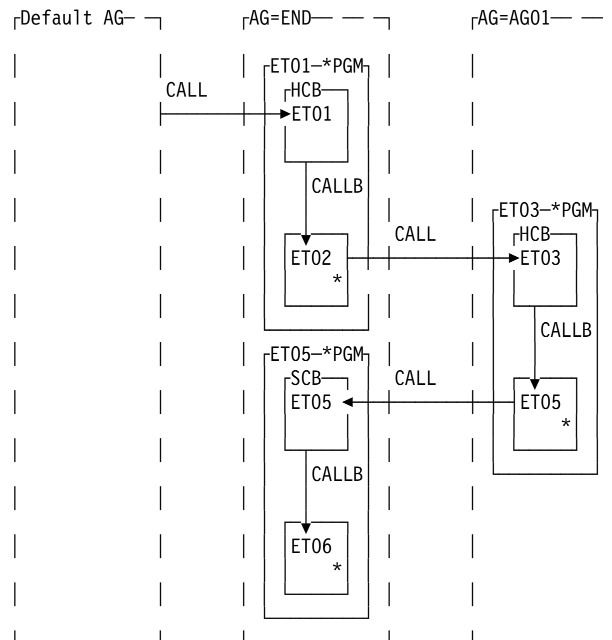


Figure 66. Example - Ending an ILE Application

Notes:

1. * Indicates the procedure in which you can choose how to end:
   - RETURN - end procedure
   - SETON LR - end procedure
   - Normal end run unit (CEETREC) - end run unit
   - Abnormal end run unit (CEE4ABN) - end run unit
2. HCB - Hard Control Boundary
3. SCB - Soft Control Boundary

What is in the example?

Program ET01 consists of procedures ET01 and ET02, and is created to run in activation group END.

Program ET03 consists of procedures ET03 and ET04, and is created to run in activation group AG01. This program is dynamically called from procedure ET02 in program ET01.

Program ET05 consists of procedures ET05 and ET06, and is created to run in activation group END. This program is dynamically called from procedure ET04 in program ET03.

Note that procedures ET01 and ET03 are hard control boundaries in activation groups END and AG01 respectively. Procedure ET05 is a soft control boundary. It is the presence of this soft control boundary in activation group END that *prevents* the activation group END from being deleted when a hard leave is issued from procedure ET06.

Each of the procedures in these programs uses RPG display to keep track of what is happening; ILE CL procedure DLY issues a Delay Job of three seconds for you to note the RPG DSPLY output. ILE CL procedure DJ issues a Display Job for you to review the call stack (option 11), open files (option 14) and activation groups (option 18). Finally, procedure ETEND is called from procedures ET02, ET04 and ET06 enabling you to specify the end option. Procedures ETEND, DJ and DLY are deliberately NOT shown on the diagram, in order to keep the diagram simple. Please review the source in file GG244358/QDESIGN to see the exact statements that are being run.

If you use the end run unit abnormal (CEE4ABN) option, we recommend that you review the job log in order to better understand the ILE errors that are being placed.

7.12.4 Use of RCLRSC

This command is only run to clean up resources assigned to the default activation group. It is not used to clean up resources in either a user-named or *NEW activation group as it cannot see these resources.

When this command is run against OPM programs and ILE compatibility mode programs, then the programs′ ODPs are deleted and all static storage attributed to these programs is freed.

When this command is, however, run against ILE procedures activated in the default activation group (for example, created with CRTPGM...ACTGRP(*CALLER)) and called by another program or procedure running, ODPs are deleted and all associated static storage is marked for refresh upon the next call to the procedure.

Chapter 8. Development Environment

Today, change is intrinsic to most software engineering endeavors. Applications consist of many components, some of which are based on previous versions and all of which may be undergoing constant revision. The problems of application development are further complicated by the interdependences that exist between components. Changing just one component may result in changes to several other components that depend on it.

To build and maintain high-quality applications efficiently, application development organizations must have a consistent and systematic approach to manage changes they are making to their applications. They are looking for standard methods and procedures and, where possible, automation to improve productivity and reduce backlogs.

Development organizations need to be able to organize and manage all the components of an application as a unit. They also need to be able to control the baseline or master version of an application. They often want to control multiple versions of components or of entire applications. And they want to be able to make quick fixes to the code they are developing. They need a mechanism that allows for shared access to components, and they want the means to plan and manage their entire development process. They must reduce the time they spend in maintaining applications so that they have more time to develop new applications.

8.1 Application Development Manager/400

The Application Development Manager/400 product answers many of the needs of today′s application developers. It provides a team of application developers, working in an AS/400 environment, with a mechanism for efficiently and effectively managing their development environment and its application objects throughout the life of the application.

Benefits

- **A standard development process**

  A development team can define the application that suits its organization and methods.

- **Increased productivity**

  This product organizes both the developers writing the code and the code to be written. As developers write their code and compile and test it, they work efficiently and productively in a well-organized development environment where changes to their code are managed.

- **Flexibility and versatility**

  The structure defined at the beginning of a project does not restrict the development team. This structure is changed and refined at any time, as the demands of the project change. Developers are added to or removed from the project, and code is shared and reused.

© Copyright IBM Corp. 1995

- **Support for several versions of an application**

  Developers can create and maintain multiple versions of an application in both the Application Development Manager/400 development environment and in a production environment. They can easily identify which versions of source and objects belong to a particular version of the application.

- **An automated build process**

  Developers can rely on the powerful build process to build, or compile, the source code for an application more quickly. They no longer have to analyze the relationships between pieces of code; the build process does this for them automatically.

- **Data security and integrity**

  The structure that the Application Development Manager/400 environment provides ensures the integrity and security of production, test, and development versions of the code. Developers are able to work with the different versions of the code. And they work in an environment where they are assured that they cannot overwrite one another′s changes.

- **An audit trail**

  A project log records what has changed in the application, the commands used to change the hierarchy or components it contains, who issued the commands, and when the activity took place.

- **Notification about the status of any component of the application**

  Developers receive messages that tell them that a part they are requesting is already being changed, that it exists in another branch of the project hierarchy, and that a change made in one version may also have to be made in the other version. This is useful when fixes to a production version of a part have to be propagated to the follow-up version of the part. The person doing this work needs to know where the part is in the hierarchy and who has checked it out to a development group.

- **Ability to package applications**

  The Application Development Manager/400 product provides a mechanism that automates the packaging of applications through the use of functions of the SystemView System Manager/400 licensed program. For more details on how this is achieved, see the Redbook, *Software Life Cycle Management with SystemView Sytem Manager/400*, GG24-4187.

8.1.1.1 Who is using Application Development Manager/400?

The two main types of users for this product are:

- **The project administrator**

  The person who creates a project is automatically authorized to work on the project as the project administrator. This person defines the phases, such as development and testing, through which components of an application go before they are actually placed in a production environment. This person takes the following actions:

  – Maintain the project hierarchy, divided into groups.
  – Enroll and remove developers from the project and grant them authority to the different groups.

- **The developer**

  The application developer is any member of the development team who has been given access to the application by the project administrator. Application developers usually only have **update** access to specific groups. They do the following tasks:

  – Create or change source code (parts) of the application in a development group. Parts to be modified are copied from the application group already under control of Application Development Manager/400.
  – Compile parts and applications using the build process that automates much of this work.
  – Test the application (or parts), either within the control of ADM/400 or outside by moving the code to another test environment.

In this chapter, we discuss more in detail what actions should be taken for a successful implementation of the Integrated Language Environment concept. We have chosen the Application Development Manager/400 product as the base for our development environment.

The *Application Development Manager/400 Introduction and Planning Guide*, GC09-1807, gives more detailed information on the usage of this product.

8.1.2 Naming Conventions

Since there are so many objects (parts) involved in the development process of ILE applications, some naming conventions are very helpful to identify relations and dependencies between the different parts. Let us first start with a list of typical part types involved:

- Source
- Module
- Program
- Service program
- Binding source
- Binding directory
- Build option

The build option (BLDOPT) is a very important part, since it can influence the creation of other parts. Application Development Manager/400 takes by default the normal create commands from the system; the next level of control occurs when you create a QDFT BLDOPT part in which you describe your defaults for a certain project or group. And for complete control, you can make a BLDOPT part with the same name as the source part that you want to build (create). This overrules all other BLDOPT create definitions.

Creating a PGM requires a build option part with the same name as the entry point module.

Figure 67 on page 126 shows an example of how this is used as documentation as well.

```
CRTPGM  PGM(MYPROGRAM) MODULE(MODULE1            +
                              MODULE2            +
                              MODULE3 )          +
                         TEXT('My new program')  +
                    BNDSRVPGM(SRVPGM01)          +
                       ACTGRP(MLGILE)            +
                       USRPRF(*USER)             +
                      REPLACE(*YES)              +
                          AUT(*EXCLUDE)
```

Figure 67. Example of a Build Option for Documentation

Figure 67 shows you what program is built, which modules were used, what service programs are bound, and the activation group it is run in. Remember that without the use of &ADM, you have to document this somewhere else; otherwise you are losing control in the change management process.

If you are not using Application Development Manager/400 and you need to re-create the program, all information needed for a re-create is available in the program description itself, but this information is not available in an outfile (only interactive or as *print). If you use the example of the pointer program in 3.7, "Example Using Pointers in RPG IV" on page 53 you can store that information in an outfile, and generate your own CRTPGM.

Note: The UPDPGM and UPDSRVPGM commands are not supported in Application Development Manager/400.

8.1.3 Relationships

What are the relations between some of the parts that are mentioned before?

```
Type         Creates         Has a create-relation with
=======================================================
SOURCE       MODULE

MODULE       PGM             Binding directory
                             Other modules
                             Entry module (PEP)
                             Service program

MODULE       SRVPGM          Binding source
                             Binding directory
                             Other modules
                             Service program
```

Figure 68. Create Relations

In Application Development Manager/400, the same table should be constructed as follows:

```
Create Part    Triggered by    Has a create-relation with
==================================================================
MODULE         SOURCE

PGM            MODULE          Binding directory
                               Other modules
                               Entry module (PEP) (E-module)
                               Service program

SRVPGM         BNDSRC          Binding source
                               Binding directory
                               Other modules
                               Service program
```

Figure 69. Trigger Relations

So a naming convention that fits the Application Development Manager/400 environment could look like:

```
Create Part          Involved parts          Remarks
=========================================================================
MODULE=modulename    SOURCE =modulename      Required
                     BLDOPT =QDFT            Optional Project default

PGM   =E-modulename  MODULE =E-modulename    Required
                     MODULE =modulename      Optional
                     SRVPGM =servicepgm      Optional
                     BNDDIR =binddirname     Optional
                     BLDOPT =E-modulename    Required
                                              Contains the CRTPGM with
                                              MODULE     ......
                                              TEXT       ......
                                              ENTMOD     ......
                                              BNDSRVPGM  ......
                                              BNDDIR     ......
                                              ACTGRP     ......

SRVPGM=servicepgm    BNDSRC =servicepgm      Required
                     BNDDIR =binddirname     Optional
                     MODULE =modulename      Optional
                     BLDOPT =servicepgm      Required
```

Figure 70. Part Naming Rules

Using this kind of naming convention is probably not taking care of everything, for example, if you are using the E-module (Entry module) also as a non-entry module in other program binds, the naming convention is broken. But you need to think about using some kind of a naming convention in your environment, since there are so many more objects (parts) involved today in an OPM environment.

8.2 Introduction of the Walk-Through Scenarios

We use a modified version of the mailing list application as described in the *Application Development by Example, SC41-9825*, manual as the base application for this chapter of the Redbook. The source code was first migrated into ILE and enhanced to reflect some of the typical scenarios needed for this ILE example.

A description of the mailing list application is found in D.2, “Mailing List Application Description” on page 194. Figure 71 on page 128 shows the menu interface for the mailing application that we are creating.

We recommend access to an AS/400 system while going through this scenario.

```
                       Mailing List Menu
                                                          System:  RCHASM02
Select one of the following

     1. Inquire into Mailing List Master
     2. Maintain Mailing List Master
     3. Submit mailing by account number
     4. Submit special analysis report
     5. Query Mailing List file

Selection
===>

F3=Exit
```

Figure 71. Mailing Application Menu

The migration is done using the CVTRPGSRC command. A full description of this kind of process is found in Chapter 4, “Conversion Considerations” on page 57. All the sources are located in the library GG244358 supplied with this publication. In Appendix A, “Diskette Install Instructions” on page 177 you will find the instructions on how to install this library on your system.

The CVTPART command in Application Development Manager/400 also provides the capability of converting an RPGSRC part to a RPGLESRC part. The same is true for CL sources as well.

8.2.1.1 Scenarios

- 8.2.2, “Setup of the Application Development Manager/400 Environment” on page 129

  We will first create the Application Development Manager/400 environment to work with. This includes the definition of a project and the definition of groups to work with.

  All the migrated source parts are imported into this environment and then we will build the initial application. This application is the same as the application described in the appendix D.2, “Mailing List Application Description” on page 194.

  For the most part, this is done by running a program. The last part of this scenario is done manually to show the aspects of building (binding) programs.

- 8.2.3, “Enhance the Mailing Application (Service Programs)” on page 132

  This scenario introduces the use of service programs and guides you in creating and changing the necessary sources. Furthermore, it shows the capabilities of Application Development Manager/400 in knowing what should be rebuilt, since it has stored all the relations from the previous build processes.

- 8.2.4, “Enhance a Service Program (Signature Implications)” on page 138

  We continue with the service programs by merging them. Since not every procedure in a service program is used by the program that uses that service program, we can merge the two service procedures into one service program.

  We discuss and show the implications of the signature checking.

- 8.2.5, “Import/Export Variables in ILE” on page 140

We start with the setup of the Application Development Manager/400 environment to work with and discuss the needs for this implementation.

8.2.2 Setup of the Application Development Manager/400 Environment

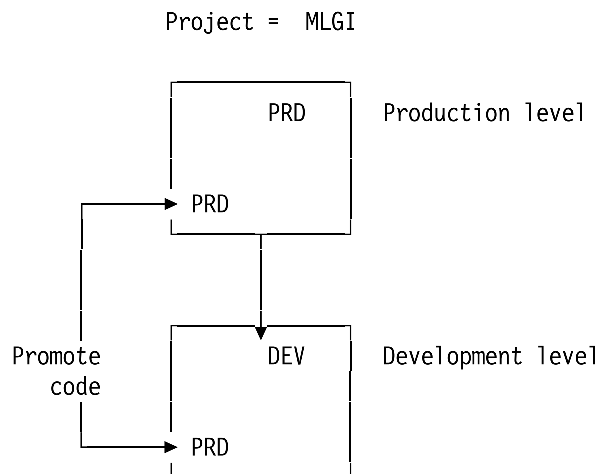


Figure 72. Project Structure

1. Create the project as shown in Figure 72 to work with all the necessary groups.

   Call the following program to create the project and groups and import all the source members:

   ```
   CALL  GG244358/CRTADMENV
   ```

   A complete description of the program is in D.1.1, “ADM Setup” on page 189

2. Build the application by:

   - Starting ADM/400 with the command STRPDM
   - Take option 4, type MLGI and press the Enter key
   - Take option 12 to work with project MLGI
   - Take option 12 to work with group PRD
   - Take option 14 and build the parts by prompting for the BLDPART command (F4)

```
                              Build Part (BLDPART)

 Type choices, press Enter.

 Project . . . . . . . . . . . . PRJ          > MLGI____

 Group  . . . . . . . . . . . . . GRP          > PRD_____

 Type . . . . . . . . . . . . . . TYPE         > *ALL____
 Part . . . . . . . . . . . . . . PART         > *ALL____
 Language . . . . . . . . . . . . LANG           *ALL____
 Search path  . . . . . . . . . . SCHPTH         *DFT____
 Scope of build . . . . . . . . . SCOPE          *NORMAL
 Force build  . . . . . . . . . . FORCE          *NO
 Build mode . . . . . . . . . . . BLDMODE        *COND
 Save list  . . . . . . . . . . . SAVLST         *NO
 Perform bind step  . . . . . . . BINDSTEP       *YES
 Part list  . . . . . . . . . . . PARTL          *NONE
```

Figure 73. Building the Initial Application

The build report should look like this:

```
5763PW1   V3R1MO              Application Development Manager/400 - Build Report    12/18/94

Project . . . . . . . . . . . . . . . . :  MLGI                                     15:04:16
Group  . . . . . . . . . . . . . . . . :  PRD
Type . . . . . . . . . . . . . . . . . :  *ALL
Part . . . . . . . . . . . . . . . . . :  *ALL
Search path  . . . . . . . . . . . . . :  *DFT
Scope of build . . . . . . . . . . . . :  *NORMAL
Force build  . . . . . . . . . . . . . :  *NO
Build mode . . . . . . . . . . . . . . :  *COND
Save list  . . . . . . . . . . . . . . :  *NO
Perform Bind Step  . . . . . . . . . . :  *YES
Search path part used  . . . . . . . . :  *DFT
Search path used . . . . . . . . . . . :  MLGI                        PRD


5763PW1   V3R1MO        Application Development Manager/400 - Build Outputs   12/18/94    15:04:16
      DDSSRC-PF            FILE          BLDOPT
Part        Group          Created     Part        Group      Reason for Building
----------  -------------- ----------  ----------  --------   --------------------------------------
MLGREFP     PRD            MLGREFP     *DFT        *NONE      Source part has not been built before.
MLGMSTP     PRD            MLGMSTP     *DFT        *NONE      Source part has not been built before.
      DDSSRC-LF            FILE          BLDOPT
Part        Group          Created     Part        Group      Reason for Building
----------  -------------- ----------  ----------  --------   --------------------------------------
MLGMSTL     PRD            MLGMSTL     *DFT        *NONE      Source part has not been built before.
MLGMSTL2    PRD            MLGMSTL2    *DFT        *NONE      Source part has not been built before.
MLGMSTL3    PRD            MLGMSTL3    *DFT        *NONE      Source part has not been built before.
MLGNAML     PRD            MLGNAML     *DFT        *NONE      Source part has not been built before.
```

Figure 74 (Part 1 of 2). Build Report Initial Application

```
      DDSSRC-DSPF           FILE          BLDOPT
Part        Group           Created     Part        Group     Reason for Building
----------  --------------  ----------  ----------  --------  ------------------------------------
MLGINQD     PRD             MLGINQD     *DFT        *NONE     Source part has not been built before.
MLGMNUD     PRD             MLGMNUD     *DFT        *NONE     Source part has not been built before.
MLGMTND     PRD             MLGMTND     *DFT        *NONE     Source part has not been built before.
MLGNAMD     PRD             MLGNAMD     *DFT        *NONE     Source part has not been built before.
      RPGLESRC              OBJECT        BLDOPT
Part        Group           Created     Part        Group     Reason for Building
----------  --------------  ----------  ----------  --------  ------------------------------------
MLGINQR     PRD             MLGINQR     QDFT        PRD       Source part has not been built before.
MLGLBLR     PRD             MLGLBLR     QDFT        PRD       Source part has not been built before.
MLGLBLR2    PRD             MLGLBLR2    QDFT        PRD       Source part has not been built before.
MLGMTNR     PRD             MLGMTNR     QDFT        PRD       Source part has not been built before.
MLGNAMR     PRD             MLGNAMR     QDFT        PRD       Source part has not been built before.
MLGRPTR     PRD             MLGRPTR     QDFT        PRD       Source part has not been built before.
      CLLESRC               OBJECT        BLDOPT
Part        Group           Created     Part        Group     Reason for Building
----------  --------------  ----------  ----------  --------  ------------------------------------
MLGMNUC     PRD             MLGMNUC     QDFT        PRD       Source part has not been built before.
MLGMTNC     PRD             MLGMTNC     QDFT        PRD       Source part has not been built before.
MLGRPTC     PRD             MLGRPTC     QDFT        PRD       Source part has not been built before.
MLGRPTC2    PRD             MLGRPTC2    QDFT        PRD       Source part has not been built before.
                                * * * * *  E N D  O F  B U I L D  R E P O R T * * * * *
```

Figure 74 (Part 2 of 2). Build Report Initial Application

3. All modules are created; use F5 to refresh your screen.

   So far it looks similar to building a non-ILE application; instead of the modules, we would have created programs. In order to achieve our objective of the application structure as shown in Figure 75, we need to decide which modules are part of what program. For the build process, this is documented in the build option (BLDOPT) part for a program.

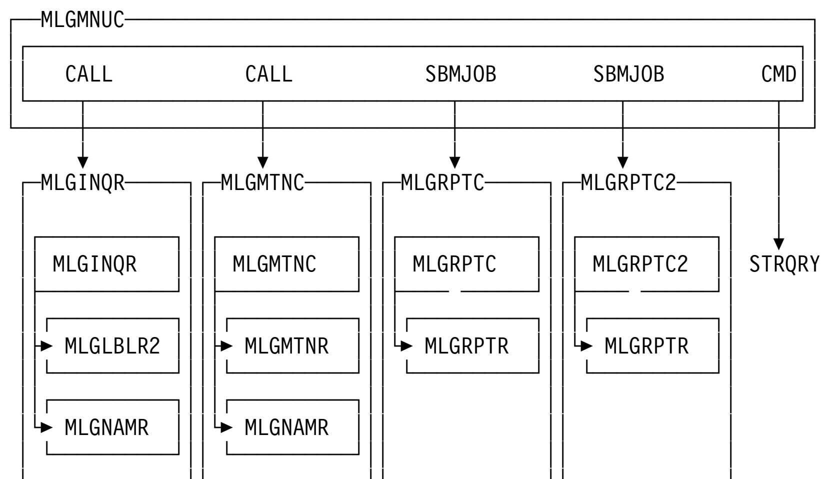


Figure 75. Initial Program Structure

4. Create all the BLDOPTs for the programs to be built.

   You can use the default BLDOPT PGMDFTBLD to make a copy from and then change the content of the BLDOPT. Or you can use:

   ```
   CALL GG244358/CPYBLDOPT
   ```

to have them all copied in as a part and have some data copied into the MLGMSTP file (account master file) as well.

Manually you can copy the data with the CPYF command:

```
                              Copy File (CPYF)

 Type choices, press Enter.

 From file  . . . . . . . . . . .   MLGMSTP       Name
   Library  . . . . . . . . . . .     GG244358    Name, *LIBL, *CURLIB
 To file  . . . . . . . . . . . .   MLGMSTP       Name, *PRINT
   Library  . . . . . . . . . . .     MLGI.PRD    Name, *LIBL, *CURLIB
 From member  . . . . . . . . . .   *FIRST        Name, generic*, *FIRST,
 To member or label . . . . . . .   *FIRST        Name, *FIRST, *FROMMBR
 Replace or add records . . . . .   *REPLACE      *NONE, *ADD, *REPLACE
 Create file  . . . . . . . . . .   *NO           *NO, *YES
 Print format . . . . . . . . . .   *CHAR         *CHAR, *HEX
```

Use F5 to refresh the screen and display the build options with option=5 and see whether you agree on their content.

5. Use option 14 against the MODULEs, for which you need to bind a program:

   a. MLGINQR
   b. MLGMNUC
   c. MLGMTNC
   d. MLGRPTC
   e. MLGRPTC2

   All programs are created according to the specifications in the CRTPGM command in the build option (BLDOPT) of the same name as the program.

6. Use option 45 to Add project library list, which sets the correct library list for the job.

7. Run the application with option=16 for the MLGMNUC program, and play around with it.

8. Leave the development environment by F3=Exit and another F3=Exit.

8.2.3 Enhance the Mailing Application (Service Programs)

We enhance the mailing application by using service programs. Instead of implementing procedures MLGNAMR and MLGRPTR within the individual programs, we move them to service programs and make them available to the application through a bind by reference. At the end, the application structure should look like Figure 76 on page 133.

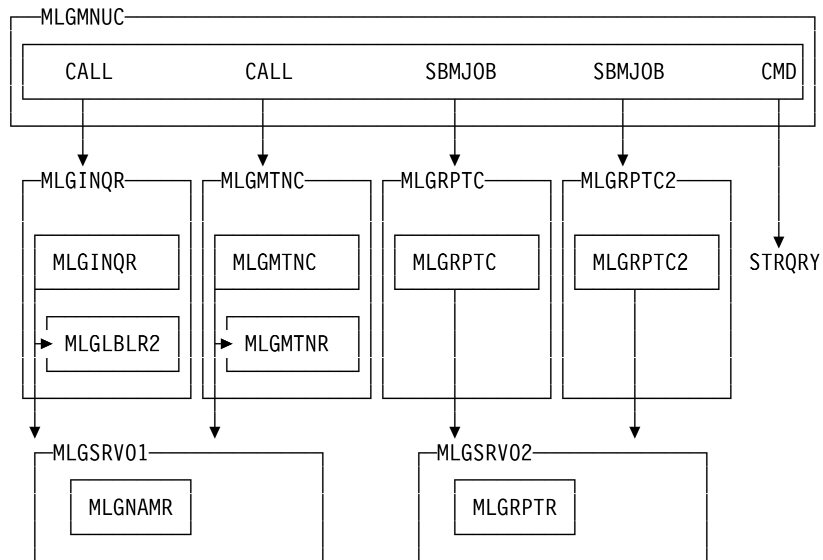


Figure 76. Program Structure Scenario-2

Start the development environment with STRPDM, select option=4 and enter project MLGI. Use option 12 to *work with* the project MLGI and select option 12 again for the Group DEV. Make sure that the Scan hierarchy on the defaults screen (F18) is set to Y. The screen you look at now is shown in Figure 77:

```
                          Work with Parts Using PDM

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  . . .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Print
  8=Display information   13=Change information   14=Build

Opt  Part        Type        Language    Group
     MLGMNUC     CLLESRC     CLLE        PRD
     MLGMTNC     CLLESRC     CLLE        PRD
     MLGRPTC     CLLESRC     CLLE        PRD
                                                  More ...
```

Figure 77. Working with Application Parts

Our base to work from is now the development group.

1. Create parts, with F6=Create

```
                              Create Part (CRTPART)

 Type choices, press Enter.

 Project  . . . . . . . . . . . . > MLGI          Name
 Group  . . . . . . . . . . . . . > DEV           Name
 Type . . . . . . . . . . . . . . > BLDOPT        Name, BLDOPT, BNDDIR...
 Part . . . . . . . . . . . . . . > MLGSRV01      Name, *GENERATE
 Language . . . . . . . . . . . .   *DFT          *DFT, name, *NONE, BND,
 Prompt create command  . . . . .   *NO           *NO, *YES
 Promote code . . . . . . . . . .   *GRP          *GRP, *NONE
 Source file  . . . . . . . . . .   *TYPE         *TYPE, name
 Part list  . . . . . . . . . . . > *NONE         *NONE, name
 Text description . . . . . . . .   *BLANK
```

Figure 78. Create a new Part Command

Create the following parts:

- MLGSRV01 BLDOPT
- MLGSRV02 BLDOPT
- MLGSRV01 BNDSRC
- MLGSRV02 BNDSRC

Scroll to the parts just created. Use option=2 (Change) to enter the information needed for the different parts just created. You can copy the information from members in the following files in *library GG244358*:

- File-QMLGBLDOP2 (for build options)

  ```
  Member = MLGSRV01
           MLGSRV02
  ```

  When entering the build options, notice that the BLDOPT is already created completely with all available commands enclosed in comment statements. Delete the complete source and copy the build option as previously directed.

- File-QMLGBNDSR2 (for binding source)

  ```
  Member = MLGSRV01
           MLGSRV02
  ```

2. Create the service programs using option 14 against the BNDSRC parts.
3. Make sure that all parts are checked in when you start, or run the command:

   ```
   CHKINPART PRJ(MLGI) GRP(PRD) TYPE(*ALL) PART(*ALL)
   ```

4. Check out the following parts from the PRD group:

   - MLGINQR BLDOPT
   - MLGMTNC BLDOPT
   - MLGRPTC BLDOPT
   - MLGRPTC2 BLDOPT

   by using option=28 against them.

5. Use F17 to subset the list shown, and include only the DEV parts.

```
                              Subset Part List

Type choices, press Enter.

  Part . . . . . . . .   *ALL                                  *ALL, name
                                                               *generic*
  Type . . . . . . . .   *ALL                                  *ALL, type
                                                               *generic*
  Language . . . . . .   *ALL                                  *ALL, *NONE
                                                               language
  Group  . . . . . . .   dev <----------------------           *ALL, name

  From date  . . . . .   01/01/00                              Earliest date

  To date  . . . . . .   12/31/99                              Latest date

  Text . . . . . . . .   *ALL
```

Figure 79. Subsetting Parts List

6. Change the BLDOPT parts just checked out; use option=2.

   Remove the modules that become part of the service program from the module list on the MODULE parameter of the CRTPGM command in all the build options parts and add the service program to it. Copy the example of the changed code from the GG244358/QMLGBLDOP2 source file with the same member name as the BLDOPT part you just checked out.

```
                             Create Program (CRTPGM)

 Type choices, press Enter.

 Label  . . . . . . . . . . . . .
 Program  . . . . . . . . . . . . PGM          > &ZE
   Library  . . . . . . . . . . .              >   &O
 Module . . . . . . . . . . . . . MODULE       > &ZE
   Library  . . . . . . . . . . .              >   *LIBL
                                               > MLGNAMR <------REMOVE
                                                   *LIBL
                          + for more values    > MLGLBLR2
                                                   *LIBL
 Text 'description' . . . . . . . TEXT           *ENTMODTXT
```

Figure 80. Example of Changing a BLDOPT of the CRTPGM Command

7. Re-create the following programs to reflect the changes made.

   Use the subset part list F17 to show only type MODULE in all groups

```
                        Work with Parts Using PDM

 Project . . . . . . . .   MLGI
 Specified group . . . .   DEV
 Position to . . . . . .              Position to type  . . .

 Type options, press Enter.
   2=Change       3=Copy    4=Delete   5=Display    6=Print
   8=Display information   13=Change information   14=Build

 Opt  Part        Type        Language    Group
 14   MLGINQR     MODULE      RPGLE       PRD
 14   MLGMTNC     MODULE      CLLE        PRD
 14   MLGRPTC     MODULE      CLLE        PRD
 14   MLGRPTC2    MODULE      CLLE        PRD
                                                   More ...
```

Change the Subset part list to show all the newly created programs in DEV. Notice that even when they are triggered against the MODULE in the PRD group, the program is created in the DEV group (under normal development conditions the developer only has read access in the PRD, and would not even be able to create or destroy programs in the PRD group by accident).

8. By using option=45, make sure that your library list is in the correct order for the test.

   Run the MLGMNUC program with option=16, or use CALL MLGMNUC, select option 1 and enter a valid search identification.

   You can check with the use of SYSREQ and option 3, the call stack of your job, and verify that the MLGNAMR procedure is now a part of the MLGSRV01 service program.

```
                          Display Call Stack Detail
                                                           System:
 Job:   P23LAWTXD      User:   ITSCID23       Number:   016002

 Request level  . . . . . . . . . . :
 Program  . . . . . . . . . . . . . :   MLGSRV01
   Library  . . . . . . . . . . . . :     MLGI.DEV
 ILE Module . . . . . . . . . . . . :   MLGNAMR
   Library  . . . . . . . . . . . . :     MLGI.PRD
 Procedure  . . . . . . . . . . . . :   MLGNAMR
```

Figure 81. Display Call Stack Detail

   Exit the application.

9. Promote all tested changes to the production environment

   Before promoting, we discuss another feature of Application Development Manager/400: the archive function. It is possible for every promote of a part to ask for archiving of the part. It certainly is not always necessary to archive on all levels of development, but you better do it on the last (product ready) level.

   The archived part is stored in a source file with the same name as the part. The source file is created in a library that the name is constructed as follows: Short Projectname, Underscore, Short Groupname, as in our example MLGI_PRD.

You can archive a maximum of five versions; the names are assigned as **archive1** through **archive5**. You should normally retrieve them by using SEU in change mode of the part that you want the previous version of.

The default for the promote (option 30) is ARCHIVE(*NO). So we create a special user-defined option called PA (promote archive) as follows:

- F16 - Get the user-defined options
- F6 - Create a new user-defined option
- Enter PA and the command PRMPART and press F4.

```
                            Promote Part (PRMPART)

Type choices, press Enter.

Project  . . . . . . . . . . . . > &ZP
Group  . . . . . . . . . . . . . > &ZG
Type . . . . . . . . . . . . . . > &ZT           Name, *ALL, BLDOPT,
Part . . . . . . . . . . . . . . > &N            Name, *ALL
Extended promote . . . . . . . . > *YES          *NO, *YES
PARTL processing option  . . . . > *PART         *LIST, *PART, *BOTH
Part list  . . . . . . . . . . .   *NONE         *NONE, name
Archive  . . . . . . . . . . . . > *YES          *NO, *YES
```

Figure 82. Defining a User Option for Promote Part

Using the PA option, we promote the following parts to the next level. Keep in mind you cannot promote program objects. If you promote a source part and have specified the Extended promote (*YES), a search is started to see if there are any created parts from this source that are promoted at the same time. (Language sources for programs and binding source for service programs)

```
                          Work with Parts Using PDM

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Prin
  8=Display information   13=Change information   14=Buil

Opt  Part        Type        Language    Group
PA   MLGSRV01    BNDSRC      BND         DEV
PA   MLGSRV02    BNDSRC      BND         DEV
PA   MLGINQR     BLDOPT      *NONE       DEV
PA   MLGMTNC     BLDOPT      *NONE       DEV
PA   MLGRPTC     BLDOPT      *NONE       DEV
PA   MLGRPTC2    BLDOPT      *NONE       DEV
PA   MLGSRV01    BLDOPT      *NONE       DEV
PA   MLGSRV02    BLDOPT      *NONE       DEV
```

Figure 83. Promoting and Archiving Parts

Use F5 to refresh. Notice that the service programs are promoted too.

Build all necessary parts after the promotion in the PRD group. Use the command:

```
BLDPART PRJ(MLGI) GRP(PRD) TYPE(*ALL) PART(*ALL)
```

Check the Build report to see if all is built correctly. Although you specified TYPE(*ALL) PART(*ALL), only the necessary parts are created. Application Development Manager/400 checks in its relation directory to determine what parts need to be built.

10. Delete all parts from the DEV group that are no longer needed.
11. Run the CALL MLGMNUC to check if all is still functioning well.

8.2.4 Enhance a Service Program (Signature Implications)

We enhance service program MLGSRV01 to also include the function of MLGSRV02. At the end, the application structure should look like this:

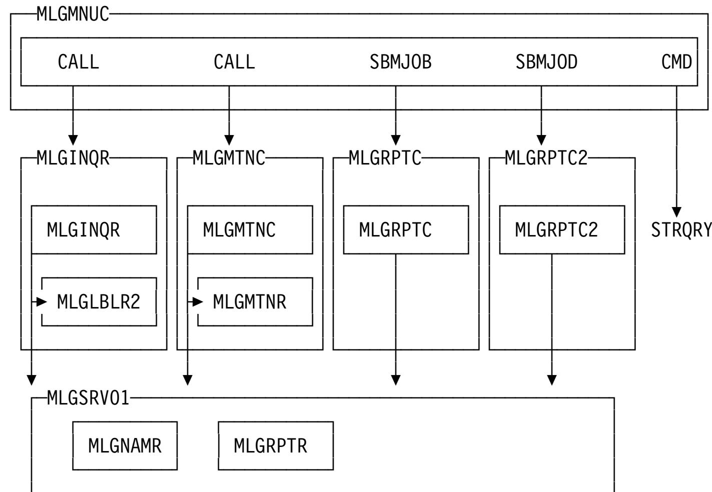


Figure 84. Program Structure Scenario-3

1. Make sure that the Subset part list includes all parts from PRD.
2. Check out the MLGSRV01 part for BNDSRC and BLDOPT from Group PRD.
3. Update both parts reflecting the change for adding module MLGRPTR.

```
BNDSRC part
    /*  Version for enhancement scenario - 3    */
    strpgmexp
    export symbol(mlgnamr)
    export symbol(mlgrptr)<--------------added line
    endpgmexp



BLDOPT part added                                            added

         CRTSRVPGM  SRVPGM(&ZE) MODULE(*LIBL/MLGNAMR MLGRPTR) +
                      EXPORT(*SRCFILE) SRCFILE(&L/&F) +
                      SRCMBR(&ZN) ACTGRP(*CALLER) USRPRF(*USER) +
                      REPLACE(*YES) AUT(*EXCLUDE)
```

Figure 85. BNDSRC and BLDOPT Updates for Scenario-3

4. Create the service program MLGSRV01, by putting option 14 against the MLGSRV01 BNDSRC part in the DEV group.

5. Run the following command:

```
RCLACTGRP *ELIGIBLE
```

The reclaim of the activation group is necessary, because the old service program MLGSRV01 is still active in the activation group MLGILE. Without ending that activation group, this old version is used instead of the new one.

To run the modified application, perform the following:

- Option=45 to set the library list
- Option=16 RUN MLGMNUC program
- Select menu option=1

You receive a message MCH4431, stating a program signature violation.

For more information on signature violation in a service program see 7.8.4, “Service Program Signature” on page 104. Resolve the problem by changing the binding language in the BNDSRC. Use the GG244358/QMLGBNDSR3 source file member MLGSRV01 to copy the information from. It shows how you can maintain the old signature (*previous) as well as a new (*current) signature.

You can also use the SCOPE(*EXTENDED) on the build part for the service program, which investigates, based on the changed signature, which programs needs to be rebuilt.

6. Re-create the service program, using option=14 against the MLGSRV01 BNDSRC

7. Run the following:

```
RCLACTGRP *ELIGIBLE
option=16 RUN MLGMNUC program
```

It should work correctly now, since it can still use the *previous signature.

8. Promote the two changed parts with the ″PA″ option.

The MLGSRV01 service program is promoted automatically.

9. Re-build in PRD

Since you have promoted all the changed parts, nothing needs to be rebuilt. Try the BLDPART PRJ(MLGI) GRP(PRD) TYPE(*ALL) PART(*ALL) command. Check the build report, which shows that no relations are *STALE (parts are out of sync), and nothing had to be build.

10. Finally, we have to change the parts that use MLGSRV02, to re-direct them to service program MLGSRV01.

In order to achieve our goal we have to checkout the BLDOPT parts for:

```
28  MLGRPTC     BLDOPT      *NONE       PRD
28  MLGRPTC2    BLDOPT      *NONE       PRD
```

Change the name of the BNDSRVPGM into (MLGSRV01) for both build options.

11. Rebuild the parts:

```
                          Work with Parts Using PDM

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  . . . . .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Print     7=Rename
  8=Display information   13=Change information   14=Build    16=Run .

Opt  Part        Type        Language    Group
     MLGNAMR     MODULE      RPGLE       PRD
14   MLGRPTC     MODULE      CLLE        PRD    <--------
14   MLGRPTC2    MODULE      CLLE        PRD    <--------
```

12. Check whether the rebuild was successful, using the DSPPGM (option=5) on one of the created programs, for example, MLGRPTC; the used service program should now be MLGSRV01.
13. Promote the BLDOPT parts with PA.
14. Delete the programs in the DEV group.
15. Build the parts in group PRD by using:

    ```
    BLDPART PRJ(MLGI) GRP(PRD) TYPE(*ALL) PART(*ALL)
    ```

16. Delete all parts of MLGSRV02.

    The parts have become obsolete since the function is now included in MLGSRV01. Decide what should happen to the archived parts, by default they are **NOT** deleted.

    You should realize that if you want to keep a copy of the last version, you need to make a copy yourself into the archive file before deletion.

```
                          Work with Parts Using PDM                  RC

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  . . . . .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Print     7=Rename
  8=Display information   13=Change information   14=Build    16=Run ...

Opt  Part        Type        Language    Group
4    MLGSRV02    BNDSRC      BND         PRD     <----------
4    MLGSRV02    BLDOPT      *NONE       PRD     <----------
4    MLGSRV02    SRVPGM      RPGLE       PRD     <----------
```

8.2.5 Import/Export Variables in ILE

This final scenario handles the import and export capabilities within the ILE language and the use of part lists for the implementation in Application Development Manager/400.

The objective is to eliminate all parameter list passing between modules and service program in the MLGINQR and MLGMTNC program, and replace it by using import/export variables.

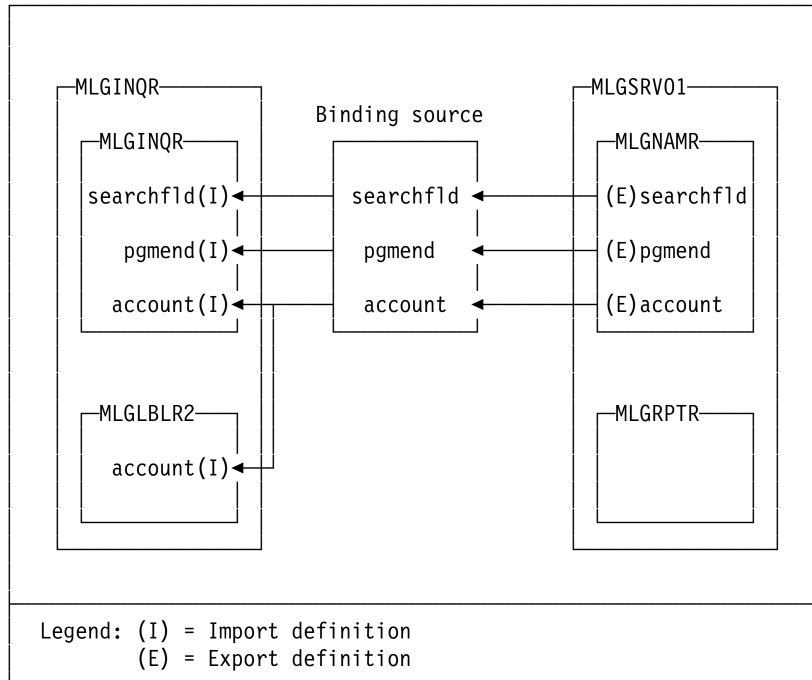


Figure 86. Relation Between Import/Export Variables and the Binding Language

As shown in Figure 86, the variables are resolved at create program time. The procedure that has the (E) export is the owner of the storage of the variable; the (I) import shares that storage space.

Note: It is not to say that this is the preferred way of passing information between programs, since CL procedures cannot exchange information using import and export variables, but between ILE procedures in general, this is the fastest way of exchanging information.

For the Application Development Manager/400 we introduce the usage of the part list. The PARTL (part list) part is used:

- As a list containing all the parts that take part in the change process
- As a list of all the parts that were created as a result of the BLDPART of the parts in the previous list
- As a list of all the parts that were promoted

1. For that reason, we create three PARTL parts in the PRD group.

   Check if you are in the PRD specified group; if not, change to it and then create the PARTL parts (F6=Create):

   ```
   SCENARIO4    PARTL   <-- List with parts to work on
   SCENARIO4C   PARTL   <-- Output list of Created parts
   SCENARIO4P   PARTL   <-- Output list of Promote process
   ```

```
                          Create Part (CRTPART)

Type choices, press Enter.

Project  . . . . . . . . . . . . PRJ          > MLGI
Group  . . . . . . . . . . . . . GRP          > PRD
Type . . . . . . . . . . . . . . TYPE         > PARTL      <---
Part . . . . . . . . . . . . . . PART         > SCENARIO4  <---
Language . . . . . . . . . . . . LANG           *DFT
Prompt create command  . . . . . PRMPT          *NO
Promote code . . . . . . . . . . PRMCODE        *GRP
Source file  . . . . . . . . . . SRCFILE        *TYPE
Part list  . . . . . . . . . . . PARTL        > *NONE
Text description . . . . . . . . TEXT           *BLANK
```

2. Use option 2 to change the content of the SCENARIO4 PARTL with:

```
MLGINQR     RPGLESRC
MLGLBLR2    RPGLESRC
MLGMTNR     RPGLESRC
MLGNAMR     RPGLESRC
MLGSRV01    BNDSRC
PARTL       SCENARIO4C
PARTL       SCENARIO4P
```

   Or use the CPYF command

```
CPYF FROMFILE(GG244358/MYPARTL) TOFILE(MLGI.PRD/SCENARIO4)
                                MBROPT(*REPLACE)
```

3. Change to the specified DEV group.

   Now check out SCENARIO4 PARTL to the DEV group, with option=28.

4. Check out all of the other parts that are in the change process.

   All of the parts are mentioned in the part list SCENARIO4. You can check them out one by one, or you can use a user-defined option and a program mentioned in D.1.3, "Check out PARTL parts" on page 193 that can automatically check out all parts from the SCENARIO4 part list.

   Create a user option with F16, called ″XP;″ the command should be:

```
 ″XP″   call gg244358/chkoutprtl (&l &n &zt &zn &zp &zg)
```

   Run the ″XP″ option against the SCENARIO4 part list.

   Ensure that all parts in the PRD group are checked in (otherwise you run into errors):

```
CHKINPART PRJ(MLGI) GRP(PRD) TYPE(*ALL) PART(*ALL)
```

   Ensure they are all checked in. Now run the ″XP″ option. All parts should now be checked out to the DEV group.

5. Now the process of changing the application can begin.

6. Change MLGINQR

   Since some of the fields that were exchanged through a parameter list might have a different name in both procedures, we use a data structure to rename the storage position to a common name for the exchange of information (import/export).

```
0004.00  *=Changed ============================================
0005.00  * Fields to be used for the scenario of Import/Export
0006.00 Daccount         ds                  import
0007.00 d acct                          5p 0
0008.00 Dsearchfld       ds                  import
0009.00 d search                       10
0010.00 dpgmend          s              1    import
0011.00  *=================================================
```

The parameter list fields have been commented out.

```
0021.00 C                   CALLB     'MLGNAMR'
0022.00  *=Changed ===================================================
0023.00 C*                  PARM                  SEARCH
0024.00 C*                  PARM                  ACCT
0025.00 C*                  PARM                  PGMEND
0026.00  *============================================================
```

Figure 87. Source Changes for MLGINQR

You will find a changed source to replace this one in the file GG244358/QMLGSRC4 member MLGINQR

7. Change MLGLBLR2

```
0008.00  *=Changed ===========================================================
0009.00  * Field to be used for the scenario of Import/Export
0010.00 Daccount         s              5p 0 import
0011.00  *==================================================================
0012.00 IMLGMSTR      01
0013.00  *=Changed ===========================================================
0014.00 C*    *ENTRY        PLIST
0015.00 C*                  PARM                  ACCT              5 0
0016.00  * Fieldname ACCT changed into "account"
0017.00 C     account       CHAIN     MLGMSTP                            99
0018.00  *==================================================================
```

Figure 88. Source Changes for MLGLBLR2

Copy from GG244358/QMLGSRC4 member MLGLBLR2.

8. Change MLGMTNR

   Copy from GG244358/QMLGSRC4 member MLGMTNR.

9. Change MLGNAMR

   The variables used by MLGNAMR are export, which means that MLGNAMR is the owner of that storage. You can have ONLY one export of a variable and multiple imports. Exporting between a service program and a program procedure requires that the service program is the owner (export) of the variable.

```
0008.00  *=Changed =====================================================
0009.00  * Fields to be used for the scenario of Import/Export
0010.00 Daccount         ds                  export
0011.00 d mlacct                        5p 0
0012.00 Dsearchfld       ds                  export
0013.00 d search                       10
0014.00 dpgmend          s              1    export
         * Add a search *ALL possibility
0015.00 c                  if       search = '*ALL      '
0016.00 c                  eval     search = '          '
0017.00 c                  endif
         * All lines commented out
0018.00 C*    *ENTRY       PLIST
0019.00 C*                 PARM                    SEARCH
0020.00 C*                 PARM                    MLACCT
0021.00 C*                 PARM                    PGMEND            1
0022.00  *=============================================================
```

Figure 89. Source Changes for MLGNAMR

Copy from GG244358/QMLGSRC4 member MLGNAMR.

By the way, we also enhanced the search program by allowing *ALL as a search name, which starts the subfile display at the beginning of the data file.

10. Change the binding language source MLGSRV01.

```
0001.00 /*  Version from enhancement scenario - 4    */
0002.00 strpgmexp pgmlvl(*current)
0003.00 export symbol(mlgnamr)
0004.00 export symbol(mlgrptr)
0005.00 export symbol(searchfld)     <---------  added
0006.00 export symbol(account)       <---------  variables
0007.00 export symbol(pgmend )       <---------
0008.00 endpgmexp
```

Figure 90. Binding Source Changes for MLGSRV01

Copy from GG244358/QMLGBNDSR4 member MLGSRV01.

11. Use the part list to build all of the parts (option=14 + F4 prompt).

```
                             Build Part (BLDPART)

 Type choices, press Enter.

 Project . . . . . . . . . . . . PRJ          > MLGI
 Group . . . . . . . . . . . . . GRP          > DEV
 Type . . . . . . . . . . . . . . TYPE        > PARTL
 Part . . . . . . . . . . . . . . PART        > SCENARIO4
 Language . . . . . . . . . . . . LANG          *ALL
 Search path . . . . . . . . . . SCHPTH       > *DFT
 Scope of build . . . . . . . . . SCOPE       ► *EXTENDED  <------
 Force build . . . . . . . . . . FORCE          *NO
 Build mode . . . . . . . . . . . BLDMODE       *COND
 Save list . . . . . . . . . . . SAVLST         *NO
 Perform bind step . . . . . . . BINDSTEP       *YES
 Part list . . . . . . . . . . . PARTL        ► SCENARIO4C <------
```

You can use the PARTL to built a subset of parts. All parts in the part list are subject to be built, although the rule still applies that parts are only built if it is really necessary.

Specify SCENARIO4C on the PARTL of the BLDPART command to store all of the information regarding the created parts in this build process. Specify SCOPE(*EXTENDED) to make sure that all of the related parts are rebuilt. If you do not specify the extended scope, the MLGMTNC part is not rebuilt.

This SCENARIO4C part list is used to determine which objects eventually need to be shipped to the production library from the PRD group, in order to have an up-to-date application. Publication *Software Life Cycle Management with SystemView Sytem Manager/400* explains in detail how to use the part list in the distribution of changed objects as PTFs.

Displaying the SCENARIO4C part list, using option=5, you get this:

```
                          Display Physical File Member
File . . . . . . :   SCENARIO4C          Library  . . . . :
Member . . . . . :   QALYPRTL            Record . . . . . :
Control  . . . . .                       Column . . . . . :
Find . . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6..
MODULE    MLGINQR   Mailing list inquiry
MODULE    MLGLBLR2  Mailing list label printing ONE LABEL
MODULE    MLGMTNR   Mailing list master maintenance
MODULE    MLGNAMR   Mailing list name search
SRVPGM    MLGSRV01
PGM       MLGINQR   Mailing list inquiry
PGM       MLGMTNC   Mailing list maintenance
                           ****** END OF DATA ******
```

Figure 91. All Created Parts Using the PARTL

12. Check that the application works well, before you start the promote process:

```
Option=45 for setting the library list
Option=16 against MLGMNUC program or CALL MLGMNUC
Try the new added *ALL search value
```

13. Promote all the parts from the part list using option=30 as in Figure 92:

```
                          Work with Parts Using PDM                  RCHASM02

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  . . . . .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Print     7=Rename
  8=Display information   13=Change information   14=Build    16=Run ...

Opt  Part        Type        Language    Group
30   SCENARIO4   PARTL       *NONE       DEV
     SCENARIO4C  PARTL       *NONE       DEV
     SCENARIO4P  PARTL       *NONE       DEV
```

Figure 92. Promoting Parts

Use F4 to prompt.

```
                          Promote Part (PRMPART)

Type choices, press Enter.

Project  . . . . . . . . . . . . PRJ         > MLGI
Group  . . . . . . . . . . . . . GRP         > DEV
Type . . . . . . . . . . . . . . TYPE        > PARTL
Part . . . . . . . . . . . . . . PART        > SCENARIO4
Extended promote . . . . . . . . EXTEND      ► *YES       ◄───────
PARTL processing option  . . . . PARTLOPT    ► *BOTH      ◄───────
Part list  . . . . . . . . . . . PARTL       ► SCENARIO4P ◄───────
Archive  . . . . . . . . . . . . ARCHIVE     ► *YES       ◄───────
```

Change the EXTEND, PARTLOPT, PARTL and the ARCHIVE parameters.

```
                         Display Physical File Member
File . . . . . . :   SCENARIO4P          Library  . . . . :   MLGI.DEV
Member . . . . . :   QALYPRTL            Record . . . . . :   1
Control  . . . . .                       Column . . . . . :   1
Find . . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
PARTL     SCENARIO4
BNDSRC    MLGSRV01  Mailing Service program 01
SRVPGM    MLGSRV01
PGM       MLGINQR   Mailing list inquiry
RPGLESRC  MLGINQR   Mailing list inquiry
MODULE    MLGINQR   Mailing list inquiry
RPGLESRC  MLGLBLR2  Mailing list label printing ONE LABEL
MODULE    MLGLBLR2  Mailing list label printing ONE LABEL
RPGLESRC  MLGMTNR   Mailing list master maintenance
MODULE    MLGMTNR   Mailing list master maintenance
RPGLESRC  MLGNAMR   Mailing list name search
MODULE    MLGNAMR   Mailing list name search
                          ****** END OF DATA ******
```

Figure 93. Content of the Promote Part List after the Promote

You notice in Figure 93 that the MLGMTNC program is not promoted; in general program and module objects cannot be promoted. Specify option EXTEND(*YES) to promote those objects with the source. Since the source is not in the DEV group, MLGMTNC was rebuilt but not based on a source (change), but based on a changed service program and a changed module in MLGMTNC. So this program still needs to be re-created in the PRD group.

14. Promote the two remaining PARTL parts SCENARIO4C and SCENARIO4P:

```
                         Work with Parts Using PDM                RCHASMO

Project . . . . . . . .   MLGI
Specified group . . . .   DEV
Position to . . . . . .               Position to type  . . . . .

Type options, press Enter.
  2=Change       3=Copy    4=Delete   5=Display    6=Print     7=Rename
  8=Display information   13=Change information   14=Build    16=Run ...

Opt  Part        Type        Language    Group
30   SCENARIO4C  PARTL       *NONE       DEV
30   SCENARIO4P  PARTL       *NONE       DEV
```

Use F4 prompt.

```
                              Promote Part (PRMPART)

 Type choices, press Enter.

 Project  . . . . . . . . . . . . PRJ         > MLGI
 Group  . . . . . . . . . . . . . GRP         > DEV
 Type . . . . . . . . . . . . . . TYPE        > PARTL
 Part . . . . . . . . . . . . . . PART        > SCENARIO4C and SCENARIO4P
 Extended promote . . . . . . . . EXTEND        *NO
 PARTL processing option  . . . . PARTLOPT      *PART  <--------
 Part list  . . . . . . . . . . . PARTL       > *NONE
 Archive  . . . . . . . . . . . . ARCHIVE       *NO
```

15. And finally delete part MLGMTNC from the DEV group.

16. Rebuild whatever is necessary in the PRD group.

    Use the part list SCENARIO4 to re-create all objects that did not make it through the promote process, such as MLGMTNC. Switch to the specified group PRD and use option 14 against the PARTL SCENARIO4 and prompt:

```
                              Build Part (BLDPART)

 Type choices, press Enter.

 Project  . . . . . . . . . . . . PRJ         > MLGI
 Group  . . . . . . . . . . . . . GRP         > PRD
 Type . . . . . . . . . . . . . . TYPE        > PARTL
 Part . . . . . . . . . . . . . . PART        > SCENARIO4
 Language . . . . . . . . . . . . LANG          *ALL
 Search path  . . . . . . . . . . SCHPTH      > *DFT
 Scope of build . . . . . . . . . SCOPE       ► *EXTENDED  <----------
 Force build  . . . . . . . . . . FORCE         *NO
 Build mode . . . . . . . . . . . BLDMODE       *COND
 Save list  . . . . . . . . . . . SAVLST        *NO
 Perform bind step  . . . . . . . BINDSTEP      *YES
 Part list  . . . . . . . . . . . PARTL       ► SCENARIO4C <----------
```

As expected, nothing is added to the SCENARIO4C part list, since only the same objects are built here. The MLGMTNC part is not added, because the process checks that a part is only added if it does not exist yet.

8.2.5.1 Conclusion

The mechanism of the part list is very important for controlling the change, rebuild and distribution process. It also gives instant documentation on the parts that have been worked on, parts that needed to be rebuilt, and finally the parts that needs to be promoted.

8.3 Use Binding Directories in Application Development Manager/400

If you are binding several modules into a PGM or SRVPGM, there are a few different ways you can reference modules:

1. Specify modules on the MODULE parameter of the CRTPGM or CRTSRVPGM command in the BLDOPT part.
2. Reference a service program on the BNDSRVPGM parameter of either the CRTPGM or CRTSRVPGM command.
3. Reference modules, service programs and binding directories, through bind directory parts (BNDDIR).

If you want to build the relationship between PGM and MODULE parts, then method 1 and 2 mentioned previously are recommended. These methods result

in the proper build relationship information, and the PGM or SRVPGM is rebuilt, if any of the modules change.

Method 3 creates a build relationship between a binding directory and a program, resulting in a massive rebuild when the content of that binding directory changes (add or delete of a module) and *NOT* a rebuild when a module within that service program changes.

8.4 How to Manage Without Application Development Manager/400

If you have gone through the scenarios, you might have a better understanding of what making changes in an modularized environment requires. For those who do not have access on their machine to the Application Development Manager/400 product, we have prepared a setup in the conventional way as well. The application according to the Figure 75 on page 131 is completely created for you when you run:

```
CALL CRTMLGAPP ('my_mlglib')
```

Follow all the scenarios to enhance and change the application without Application Development Manager/400, and look what effort it costs to get it right the first time. Imagine what it could look like in a development environment of a couple of thousand parts.

Documentation is more critical to the final result (quality of the application) than ever. So think about Application Development Manager/400 and how your application parts relate to each other while you develop, rather then worry about them.

8.5 Copyright Your Software

You might have wondered when using the DSPPGM command, why the copyright information is usually empty for your programs. The answer is: So far only the C language has implemented the use of a copyright statement during the compile of a module. When a program is created, the copyright information from the program entry module (PEP) is used for the program.

When you want to create a program with a copyright statement, here is an example of how to fix it.

1. Create a C-module that has only one function, namely to call dynamically the mailing menu program (MLGMNUC).

   Look at the source MAILING in source file GG244358/QMLGSRC. Use this to create the C-module. (The C-module is already in the library).

2. Create the program MAILING as follows:

```
                          Create Program (CRTPGM)

 Type choices, press Enter.

 Program  . . . . . . . . . . . . PGM         > MAILING
   Library  . . . . . . . . . . .             >   *CURLIB
 Module . . . . . . . . . . . . . MODULE      > MAILING
   Library  . . . . . . . . . . .             >   *LIBL
                          + for more values   > MLGMNUC
                                              >   *LIBL
 Text 'description' . . . . . . . TEXT          *ENTMODTXT


                         Additional Parameters

 Activation group . . . . . . . . ACTGRP      > MLGILE
```

3. Use the DSPGM command and look at the copyright information.

4. CALL MAILING and see that it still functions as before.

If you have used the Application Development Manager/400 exercise:

- Import the MAILING *MODULE as a module part
- Create a build option for MAILING, binding the modules MAILING and MLGMNUC for activation group MLGILE
- Run option=14 against the imported module

Chapter 9. Performance

Anyone involved in ILE application design should read this chapter in conjunction with the Integrated Language Environment Concepts and Chapter 7, “ILE Design Considerations” on page 83 before migrating any part of an OPM application to ILE, or designing a new ILE application. An additional source for information is the language section of the V31 Performance Capabilities Reference, ZC41-8166, an IBM internal document. Please contact your IBM representative for the latest edition.

For this project, we have tried to gather as much performance information as possible to help you to identify critical areas in your application development process. The results we have obtained might not always apply to a *real world* environment and we ask you to verify this information before using it.

9.1 Compile Time

The new internal structuring of programs to allow the ease of binding and debugging means that compiling an entire application or program takes longer than it did with RPG/400 compiler. On the average, with default options, the ILE RPG/400 compilation takes greater than two times as long as the OPM RPG/400 compiler.

With RPG IV and modular application building, you find that you do not have to recompile entire programs or applications for each code change. The modular design associated with the ILE generally has smaller source units. In ILE, programs are built by binding module objects. Changes to the module objects are then rebound to the program object. So, you do not need to recompile the entire program object. When you build smaller functions, compiling/binding/testing are done quickly. As shown in Chapter 8, “Development Environment” on page 123, ADM/400 helps you in building only those components of an application that are affected by changes.

The following list intends to give some suggestions in managing and improving compile-time performance:

- Design modular applications

  Modular programming offers faster application development and a better ability to reuse code. Programs are developed in smaller, more self-contained procedures. These procedures are coded as separate functions, and then bound together to build an application. By building applications that combine smaller and less complex components, you can compile and maintain your programs faster and easier.

- Use sufficient working memory

  See 9.1.1.4, “Working Memory Requirements for Compile Time” on page 153.

- Use OPTION(*NOGEN) for initial compiles

  *NOGEN compiles the module or program but does not complete the build of a program object. It is used to fix and edit compile errors.

- Use the value of DBGVIEW adequate for your purpose

  Requesting debug information requires more compile time. For example, DBGVIEW(*LIST) results in a slower compilation time than DBGVIEW(*STMT).

© Copyright IBM Corp. 1995

If the level of debug information you need is that provided by DBGVIEW(*STMT), selecting *LIST would unnecessarily slow down compilation time.

- Use OPTIMIZE(*BASIC) instead of OPTIMIZE(*NONE)

9.1.1 Compile Options

Options on the create commands allow you to influence the compile time. Some of those are:

- OPTIMIZE
- DBGVIEW
- OPTION(*NOGEN)
- OUTPUT(*NONE)

RPG IV compile and bind time takes significantly longer than OPM compile time and the compiler needs more memory than OPM RPG. Thus, a sufficient memory pool improves compile time.

9.1.1.1 Effect of Debug Options

Requesting debug information requires more compile time and creates bigger objects. Tests show that the compile times for the DBGVIEW options increases in performance cost in the following order:

- DBGVIEW(*NONE)
- DBGVIEW(*STMT)
- DBGVIEW(*SOURCE)
- DBGVIEW(*LIST)
- DBGVIEW(*ALL)

Do not ship production level code with debug data; if you have used the defaults, then source level statement debug data DGBVIEW(*STMT) is included in your programs. Once testing is completed, we recommend that you remove debug data from your ILE programs using:

```
CHGPGM PGM(ilepgm) RMVOBS(*DBGDTA)
```

Warning - Giving away source code

Do not ship production code with DBGVIEW(*LIST); this option causes a complete copy of the source code to be placed in the program object, and can easily be viewed using the DMPOBJ command.

9.1.1.2 Effect of OPTION(*NOGEN)

Use of *NOGEN in RPG IV performs significantly better than compiling with the *GEN option. Use *NOGEN to do the early compiles of programs to remove edit or compile errors.

9.1.1.3 Effect of Optimize Options

We recommend that you compile your ILE programs with OPTIMIZE(*BASIC) rather than the default of OPTIMIZE(*NONE). Use of OPTIMIZE(*BASIC) has minimal impact on compile time but it results in improved runtime performance.

OPTIMIZE(*FULL) takes significantly more time than OPTIMIZE(*NONE) to compile, and does not significantly improve performance over OPTIMIZE(*BASIC).

9.1.1.4 Working Memory Requirements for Compile Time

Working memory size is the amount of memory required to do a task satisfactorily. Think of working memory size this way: given infinite memory, the compiler runs at its natural speed. If you restrict memory, the compiler has to swap pages to DASD, making it run slower. The more memory is restricted, the more time the compiler spends swapping memory pages.

A phenomenon of ILE compilation is that compile time processing requires more working memory than OPM program compiles. The ILE compilers and translators, in general, produce more internal tables for binding and debug information than OPM compilers and translators used. If memory is restricted, compiles may be orders of magnitude slower than OPM. Increasing memory size adds a gain in elapsed time of compilation.

9.2 Program Object Size Comparisons

Binding and debugging ILE programs requires more information about data and more internal tables. More information is needed to manage how modules of different ILE languages are bound into a program or service program. This results in larger program object size.

The disk storage requirements for RPG IV programs has increased 1.5 to 3 times relative to OPM RPG/400.

Conclusions/Recommendations: Here are several options to consider to reduce storage requirements for your program objects:

- Remove debug date

  After creating your module, program or service program, you can reduce the object size by using the *Remove observable info* option on the CHGMOD, CHGPGM and CHGSRVPGM commands with special value *DBGDTA to remove the code generated into the object for debugging.

- Compress observability

  The *Program option* on the CPROBJ command allows you to compress only the observable parts (*OBS) and the creation templates of an object.

- Remove the observability and creation template.

  The CHGMOD (Change Module), CHGPGM (Change Program) and CHGSRVPGM (Change Service Program) commands allow you to delete the observability and the creation template to decrease the program size.

  Note: Use care when deleting the observability or the creation template. Removing observability prevents the program from being debugged; removing the creation template prevents programs from being retranslated. Be sure to keep your source code, or an offline copy of

the program with observability, to ensure that migration to future releases.

- Use of ILE modular design techniques

  Using service programs as a means of reusing code, reduce the overall storage requirements for your application.

- Selecting debug options on the compile command

  Generating DBGVIEW data may increase program object size from 20% to 200%, depending on the DBGVIEW options used.

  If DBGVIEW(*LIST), the listing view compile option is chosen, the compile listing used for debugging is stored with the object, thus greatly increasing the program object. Carefully weigh the advantage of having a compiler listing stored with your object against the additional storage requirements. Consider using DGBVIEW(*SOURCE). It may give you similar capabilities in debug, but results in a smaller program object size.

Under the assumption that you want to keep observability and creation templates for your programs a combination of CHGPGM ... RMVOBS(*DBGDTA) and CPROBJ ..., PGMOPT(*OBS) seems to be the best solution to reduce storage requirements. To achieve best results you might want to use the CHGMOD ... RMVOBS(*DBGDTA) before binding the modules into a program or service program.

9.2.1 Object Size Conversion Project

We converted 1,480 AS/400 RPG III programs to ILE RPG/400 using the CVTRPGSRC command. The CRTBNDRPG DFTACTGRP(*NO) ACTGRP(QILE) command was used to compile and bind the programs. The figures in this section are all based on the sample of 1,480 OPM RPG programs comprising a commercial application. ILE RPG programs are an average of 3.18 times bigger than OPM RPG programs. The static storage allocated to ILE RPG IV procedures remains allocated irrespective of whether LR is used. (Compare this with OPM, where LR returns static storage to the system.)

An ILE application requires more memory than an OPM application.

Table 22. Storage Requirements for ILE programs

Program Options	**Size - Bytes**	**Migrates to RISC**
OPM Default Compile	137,089,024	Yes
OPM Compressed Observability	89,225,728	Yes
OPM No Observability **1** Compressed Observability to No Observability	88,467,960	No
OPM No Observability **2** Full (Uncompressed) Observability to No Observability	40,250,368	No
ILE Default Compile DBGVIEW(*STMT)	436,343,296	Yes
ILE No Debug, Full Observability	328,303,616	Yes
ILE No Debug, Compressed Observability	149,104,128	Yes
ILE No Debug, No Observability	103,565,824	No

Notes:

1. **1** This figure results from removing observability from programs that already had compressed observability.
2. **2** This figure results from removing observability from programs that already had full, uncompressed observability.

Table 23. Size Ratio ILE to OPM programs

ILE to OPM Comparison	ILE to OPM Size Ratio
ILE Default Compile to OPM Default Compile	3.18
ILE No Debug, Compressed Observability to OPM Compressed Observability	1.67
ILE No Observability (from Compressed Observability) to OPM No Observability	1.17
ILE No Observability (from full Observability) to OPM No Observability	2.57

9.3 Runtime Performance

Runtime performance between the RPG IV compiler and the OPM RPG/400 compiler are basically the same. Using OPTIMIZE(*BASIC) costs less at compile time then OPTIMIZE(*NONE), and may result in a small performance boost. Since most RPG applications are I/O intensive, they show little performance improvement with full optimization, OPTMIZE(*FULL).

9.3.1 Working Memory Size for Runtime

Working memory size for runtime of ILE RPG/400 should be increased. However, if you run an OPM RPG/400 application in a pool barely big enough, and you switch to ILE, runtime might get dramatically worse.

If memory size is restricted, performance is degraded. For an ILE RPG/400 program, restricted memory size degrades performance more dramatically than OPM RPG/400.

9.3.2 Choice of Tools

There is a choice of tools available to assist with ILE procedure-level analysis. Each of these tools clearly indicates the program/module/procedure information relevant for both system routines and ILE application code.

- Display Job (DSPJOB)
- Sample Address Monitor (SAM)
- Timing and Paging Statistics PRPQ (TPST)

  The Print Trace Summary command in this PRPQ summarizes job trace data, providing you with a list of program or procedure flow within traced data. The report from this command has a summary of which programs called which other programs. Thus, this is used to help spot candidates for static binding.
- Job Trace

Job trace detail report shows the sequence of work performed in a job by system routines, ILE procedures, ILE programs and OPM programs. Job trace summary reports reflect counts for OPM program initializes only.

DSPJOB - Warning

Note that DSPJOB - Display Override facility shows ONLY the overrides inside the default activation group. It does not show any ILE named or *NEW activation group overrides.

9.3.3 Considerations

Often, performance and its impact are secondary issues in an application development project. They should be issues from the design phase on through the life cycle of application. The following section gives you some food for thoughts when designing an application to take advantage of the ILE concepts.

9.3.3.1 Activation Groups

Since activation groups are one of the most resource intensive components of the Integrated Language Environment, we start right here.

Named Activation Groups: Create application start-up programs (such as menu driver control programs) to run in a named activation group. For very large applications with multiple functional subsets, consider running each functional subset in a separate named activation group such that you have independent resource scoping for each subset. Create the majority of your programs with ACTGRP(*CALLER), and ensure that they are only called by the application start-up programs.

When a dynamic call is made to an ILE *PGM created to run in a named activation group, assuming that this is the first call to the program issued in the job, the system:

1. Creates the activation group (if not already created).
2. Activates the program and all associated service programs (if some service programs created with ACTGRP(*CALLER) needed by your program have already been activated by a different program in the same activation group, then they are not activated again).
3. Places a PEP on the call stack for this program.
4. Allocates and initializes static storage for all modules bound by copy in the *PGM.

   Allocate and initialize static storage for all modules bound by reference through *SRVPGMs. Note that if any of these service programs created with ACTGRP(*CALLER) have previously been activated in this activation group, then they are NOT initialized again.

   This step is known as the ILE initialization of static storage. It is the ONLY time when variables coded as EXPORT in RPG IV data specifications are initialized by the system.
5. Passes control to the procedure specified as the ENTMOD on the CRTPGM command.
6. Opens specified files.

Note that only files that are not already open as shared at activation group level, or as shared at job level are opened.

7. Runs the procedure specified on the ENTMOD.

   Note that the first bound call to any ILE RPG IV procedure causes RPG initialization of all variables excluding those specified as EXPORT in the data specifications. After the first bound call to a procedure is issued in the activation group, static storage is only reinitialized if you have coded LR.

***NEW Activation Groups:** Use this option to isolate programs that are called very infrequently (for example, once or twice) within an application for the duration of the job.

Consider that since service programs are likely to be heavily used, there is no facility to specify ACTGRP(*NEW) on the CRTSRVPGM command. This is deliberate, as use of ACTGRP(*NEW) on a frequently used ILE program delivers poor performance. The cause is not immediately apparent.

In OPM RPG, you recall that ending a program with RETRN rather than SETON LR delivers much better performance upon subsequent calls to the program, since the resources (variables and database files) are left in a ready-to-use state requiring little system effort when the program is reentered.

When a dynamic call is made to an ILE *PGM created with ACTGRP(*NEW), the system must:

1. Create a new activation group in the job.
2. Activate the program and all associated service programs (created with ACTGRP(*CALLER)). Any service programs created with ACTGRP(name) are only activated if they are not already active.
3. Place a PEP on the call stack for this program.
4. Allocate and initialize static storage for all modules bound by copy in the *PGM.

   Allocate and initialize static storage for all modules bound by reference through *SRVPGMs.

   This step is known as the ILE initialization of static storage. It is the ONLY time when variables coded as EXPORT in RPG IV Data Specifications are initialized by the system.
5. Pass control to the procedure specified as the ENTMOD on the CRTPGM command.
6. Open specified files
7. Run the procedure specified as the ENTMOD.

   Note that the first bound call to any ILE RPG IV procedures causes RPG initialization of all variables excluding those specified as EXPORT in the data specifications. After the first bound call to a procedure is issued in the activation group, static storage is only reinitialized if you have coded LR.
8. Start activation group closedown involving:
   - Closing open files
   - De-allocating static storage for activated programs
   - Deactivating all activated programs
9. Delete the activation group.

The cost of creating or deleting a new activation group is approximately .238 D45 CPU seconds. Thus ACTGRP(*NEW) should be used only in very special circumstances as a D45 is able to process four ACTGRP(*NEW) requests per CPU second.

ACTGRP Recommendation

We recommend that you use the Change Command Default command (CHGCMDDFT) on the CRTPGM command to change from ACTGRP(*NEW) to another value, such as ACTGRP(*CALLER).

Note that ADM implicitly provides you with this capability, as part of the method of building a program object. Please refer to Chapter 7, "ILE Design Considerations" on page 83 for further information.

Use of RCLACTGRP: The RCLACTGRP command is provided to enable you to clean up all resources assigned to your application's activation group or groups. Avoid the use of RCLACTGRP ACTGRP(*ELIGIBLE) in production application code; this command deletes **all** activation groups in the job that do not have procedures currently on the call stack **irrespective** of whether they are related to your application or another application. Consider using a call to the bindable API, CEETREC, from within the activation group to end an application (and delete the activation group) rather than issuing RCLACTGRP ACTGRP(named) from outside the activation group.

Database Files: This section assumes that you are running your ILE application in a named activation group.

Use shared open data paths. Open a file once in an activation group and share the file buffer with all the programs in the activation group. This is easy to code, just as it was for OPM; you simply specify:

```
OVRDBF FILE(db_file) SHARE(*YES)
```

The recommended method of opening the file is to use one or more file opener driver programs. The file opener program contains:

```
OPNDBF FILE(db_file) OPNOPT(*---)
```

ILE Bound Calls vs. OPM Dynamic Calls: We used a very simple RPG code to test the performance of an RPG IV bound call (CALLB) against a dynamic call. Prior to taking the test data, the OPM program was called once and ended with RETRN; the ILE program was called once, run in a named activation group, and ended with RETURN. The OPM program issued 1000 CALLs to another OPM program. The ILE program issued 1000 CALLBs to an ILE procedure.

The ILE bound call to an RPG IV procedure proved to be 4.06 times faster than an OPM dynamic call. Note that, if the OPM program had ended in LR rather than RETRN, then the ILE CALLB with return was 59.8 times faster.

We recommend that where a dynamic call to a program is repeated multiple times in the same process, you consider replacing it with an ILE bound procedure call using return.

An RPG subroutine call using EXSR is still much faster than both a static call (CALLB) and a dynamic call (CALL). Do NOT replace subroutine calls through EXSR with static bound calls using CALLB and expect a performance

improvement as a result of this change. You may, however, want to weigh the cost (in terms of working set size) of having multiple copies of the same code in different program objects versus having one copy of that code in a service program.

Error Handling: For best performance, handle errors in your RPG or CL procedures using INFSR and PSSR in RPG IV and MONMSG in CL. System exception handling is much more expensive in ILE than in OPM due to the ILE error handling philosophy of percolation and promotion.

Export and Import: For bound procedure calls, if your variables have the same name in both the caller and the one being called, use RPG IV EXPORT and IMPORT rather than passing parameters. Passing parameters requires that system programs be called to check that the number and type of variables passed matches those received. ILE EXPORT/IMPORT involves a pointer to the address of the variable being setup in the procedure importing the procedure at bind time, and not at runtime while reducing the amount of static storage at the same time.

Service Programs: Package related, frequently used, ILE procedures into the same service program.

Avoid packaging many procedures into the same service program. Remember that the static storage in all of the procedures in service programs bound to an ILE *PGM is initialized when the ILE *PGM is called irrespective of whether the ILE *PGM directly references the procedures in the service program or not.

Keep static storage in service program procedures to a minimum. Avoid storing large arrays in service program procedures.

If you need to exchange variables between dynamically called ILE programs, and the caller and callee procedures are RPG IV, then consider this approach as an alternative to passing parameters as it is much cheaper in terms of system resource cost:

- Using a service program only as a container for shared variables
- For RPG IV, code the variables as IMPORT in the RPG procedures in the *PGMs, and code the variables as EXPORT in the RPG procedures in the *SRVPGM.

Note that this approach is also valid if either the caller or callee procedure is in ILE C/400.

9.4 Performance Benefits of ILE

You can see an additional performance benefit in ILE RPG/400 with bound procedure calls. The performance of calls between bound procedures in an ILE program is faster than the between programs through external calls. This fast bound call makes it more practical to write AS/400 applications in a modular fashion when using the ILE RPG/400 compiler.

Traditional OPM RPG/400 programs were designed to avoid external calls where possible. Applications tended to be built with a few monolithic programs. Programs were designed to contain as much function as they could, so they would not have to pay the performance cost of external calls. These programs grew to be large and became difficult to maintain. Making a small change to a

program was a big effort. Making a significant change was a rewrite. With the improved bound call in ILE RPG/400, writing smaller functions that call and are called by other functions are done without the performance penalty associated with external calls. You can build applications by collecting or binding smaller functions, and realize the benefits of modular programming design.

In ILE, procedures are compiled into a module object. Module objects cannot be executed. They are the building blocks of program or service program objects. ILE applications are built by binding modules into programs or service programs.

As part of the suite of ILE languages, RPG modules can now bind with modules written in different languages. Programmers can build modules in the language most familiar to them. These modules are then bound into a program or service program. You can also choose the best language for the function you are building. For example, if a difficult calculation is better suited for a language such as C, you can build an ILE C/400* module containing that algorithm. Database I/O may be more efficient written in RPG as part of an ILE RPG/400 module. These two modules are then bound into one program object or service program object. ILE lets you use the right language for the function and combine those functions to create applications. This can help overall performance and development time.

Service programs are bound with program objects to enable fast bound calls to the procedures contained in the service program. You can use service programs to build a function that is used by different applications or programs. The service program are bound to each and any program that needs to call a procedure in that service program. Writing a service program that is bound to many programs simplifies code reuse . For example, the compiler runtime support is packaged into service programs. These service programs are bound to your compiled programs. When a compiler runtime function is called in your program, it is called through a bound call.

Because modular design has smaller source units, you are building your applications differently. You can build smaller building blocks for your application by using modules or service programs. If you have to change a function, you only need to compile that function and rebind. For example, with service programs, you can separate high-use, frequently changed code, such as tax routines. When you maintain this code in separate service programs, you improve the efficiency of maintenance by avoiding recompiling and retesting the main applications. In turn, the overall integrity of the application is improved.

The data in the next section is from measurements that were run using V1R1 level software. This section compares the results of the RPG II and RPG III compilations and executions in the ABS product lines. Performance comparisons were made between the AS/400 system, System/36, and System/38. All jobs were run in batch mode and in a stand-alone environment.

Chapter 10. Exception Handling

The purpose of this chapter is to describe the different exception and error handling techniques available with ILE, focusing on those available for the ILE RPG/400 and ILE CL programming languages. For more detailed descriptions of the language-specific exception handling methods, you might want to refer to the appropriate language publications.

Most programs benefit from the implementation of planned exception handling because it can both minimize end-user frustration, and reduce the performance overhead associated with the system handling the exception. In this chapter, we see that in ILE applications it is more costly (in system resource) to allow system default exception handling than it was in OPM applications.

The term *exception handling* is used within this chapter to refer to both exception handing and error handing. However, for consistency with RPG terminology, the term *error* is used in the context of *error indicator* or *error handling subroutine*.

10.1 What Is An Exception/Error?

There is a difference between exceptions and errors:

- An exception is a deviation from the normal events.
- An error is a mistake that should not have occurred.

There are two general types of exceptions that can occur:

- File exceptions:
  For example, the file does not exist or is currently unavailable.
- Program exceptions:
  For example, an invalid array index is used

Whenever a runtime error occurs, an exception message is generated.

Exception messages are associated with call stack (also known as invocation stack) entries.

10.1.1 File Exceptions

In order to determine the cause of a file exception, there are two places where file-related exception information is made available within RPG IV. These are:

1. File Open Feedback Area

   This is part of the ODP. The content is set during file open operations. It contains information about file opening operations.

2. File Input/Output Feedback Area

   The contents are updated for every I/O operation performed in the program. This is part of the ODP.

There are two facilities within RPG IV that enable us to handle file errors within the application. These are:

1. File Information Data Structure (INFDS)

© Copyright IBM Corp. 1995

This data structure comprises information from both the open feedback area and the input/output feedback area. It is specified for **every** file in a program. Should a file exception occur against a file with an INFDS coded, the contents of the data structure are used by the application to determine how processing should continue.

Note:

While ILE RPG/400 supports the use of 10 character file names and 10 character record format names, only 8 character names are supported within the file information data structure.

The layout of the INFDS is fully described in *ILE RPG/400 Reference*, SC09-1526-00.

2. Information Status Subroutine (INFSR)

   After a file exception has occurred, if an INFSR has been coded, it is given control thus allowing the application to determine what action to take.

   Note:

   If you code a single file error handling subroutine for all files in a program, then you must ensure that the first eight characters of all file names and record format names to be processed by this procedure are unique. This is because the file and record format names contained in the INFDS are limited to eight characters.

10.1.2 Program Exceptions

These are exceptions that occur while running applications that are **not** file exceptions. There are two related facilities within RPG IV that enable the developer to handle program exceptions and they are:

1. Program Status Data Structure (PSDS)

   One PSDS may be coded for each ILE procedure within an ILE RPG/400 procedure within an ILE program. Thus, in ILE there are multiple program status data structures within a program, with one or none for each RPG IV or ILE CL procedure within the program.

2. Program Error Subroutine (*PSSR)

   The program error subroutine receives control when an unexpected program exception occurs. To get control if a file exception occurs, code *PSSR on the INFSR keyword on the file description specification.

First, however, it is necessary to review the differences between ILE and OPM exception handling, assuming that there is NO error handling within an application.

10.2 Exception Handling Architecture

Figure 94 on page 163 shows the error-handling components for both OPM and ILE programs. Some portions of the exception handling mechanism are unique between ILE and OPM programs, while other portions are common.

This mechanism consists of these components:

- **Exception Message Architecture**

  Exception generation is common to both OPM and ILE programs.

- **Unhandled Exception Default Actions**

  The default actions taken for an unhandled exception differ between OPM and ILE.

- **HLL-Specific Handlers**

  Both OPM and ILE languages provide HLL-specific methods of handling exceptions.

- **ILE Direct Monitors**

  Some ILE languages provide the capability to more closely control exception handling through a direct monitor.

- **ILE Condition Handlers**

  ILE provides bindable APIs available to any ILE language to register condition handlers, except for ILE CL.

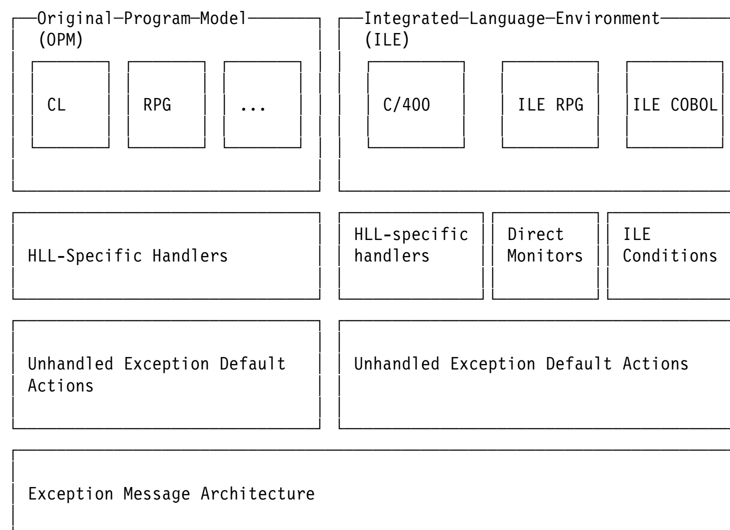


Figure 94. Error Handling Components for OPM and ILE

10.2.1 Job Message Queues and Call Stacks

The following resources are involved in the error handling process

Call stack entry
A call stack entry exists for each OPM program and for each ILE procedure in a bound program that is active within the job.

Call stack entry message queue
A call stack entry message queue exists for each call stack entry to facilitate sending and receiving messages between the programs and procedures running on the call stack.

For further discussion on call stack, refer to 7.5, “The Call Stack” on page 92.

External message queue
An external message queue (*EXT) exists for each job to communicate externally from the job. Messages sent to this message queue are used to communicate information outside the job′s program call stack. Inquiry and informational type messages sent to the external queue result in the Display External Messages screen being shown to the interactive user. Inquiry messages sent to the external queue from a batch job are automatically replied to with the default reply. Status type messages sent to the external queue in an interactive job result in the message being shown on line 24 of the display screen.

Job message queue
A single job message queue object is used by OS/400 to contain all the call message queues and the *EXT message queue for each job. The job message queue is used to produce the job log when the job completes.

Figure 95 graphically depicts these terms.

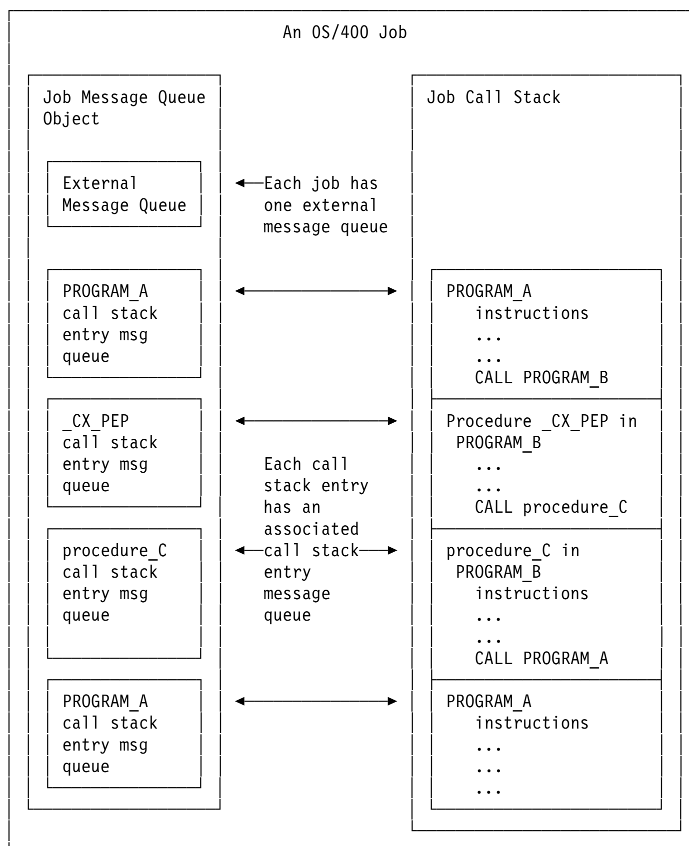


Figure 95. Job Message Queue/Call Stack example

10.2.2 Terminology

The terms frequently used in the error handling environment and a short explanation.

Control boundary
An ILE call stack entry is defined as a control boundary if either of the following are true:

- The immediately preceding call stack entry is in a different activation group.
- The immediately preceding call stack entry is an OPM program.

ILE transfers control to the call stack entry preceding the control boundary when an unhandled function check occurs, or when an HLL end verb is used.

For more information on control boundaries refer to 7.6, “Control Boundary” on page 94. Figure 96 and Figure 97 on page 166 illustrate control boundaries in both ILE and OPM.

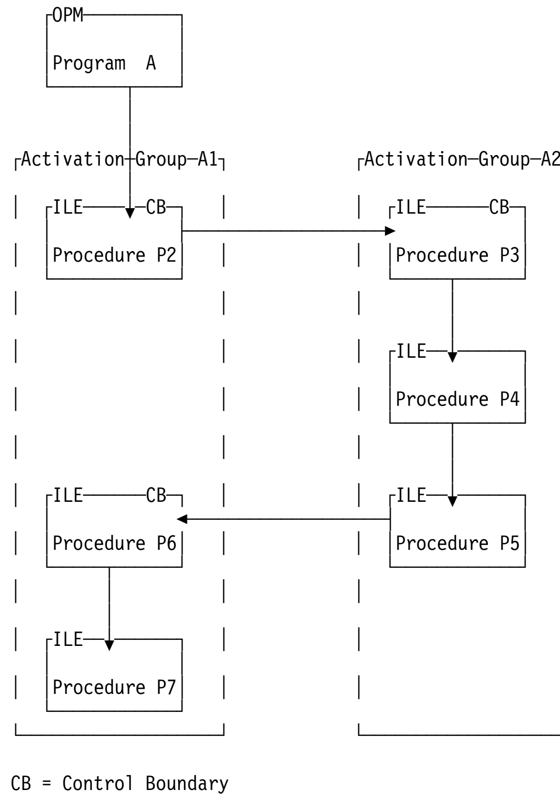


Figure 96. Control Boundaries Due to Changing Activation Groups

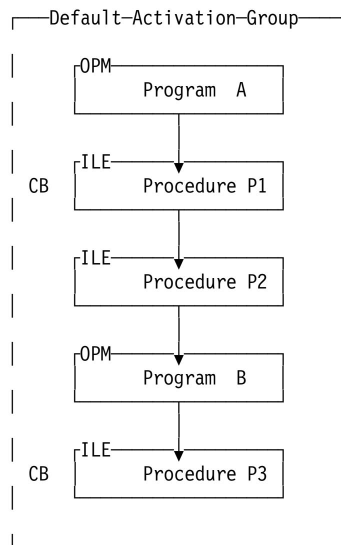


Figure 97. Control Boundaries Within OPM Default Activation Group

Percolation

When an exception message remains unhandled after calling the active exception handlers for the call stack entry, the exception message is *percolated* to the previous call stack entry. The exception handlers for this call stack entry are then called.

This percolation process continues until the exception has been percolated to a control boundary. If the exception is not handled in the control boundary, the default action for the exception message type is taken.

Handle cursor

During exception processing, successively lower priority handlers are called until the exception is marked as handled. The *handle cursor* is a pointer that keeps track of the next exception handler to be called to handle an exception.

If an exception is not handled by any of the exception handlers defined for a call stack entry, the exception message is percolated to the previous call stack entry. Percolation sets the handle cursor to the highest priority monitor in the previous call stack entry.

Resume cursor

The *resume cursor* is a pointer that keeps track of the location where a program can resume processing after handling an exception.

Normally, the system sets the resume cursor to the instruction following the instruction that receives an exception. The resume cursor may be moved by using the Move Resume Cursor (CEEMRCR) bindable API. When the resume cursor is moved to an earlier call stack entry, it points to the next statement following the call that suspended the program or procedure.

10.2.3 Exception Messages

Exception Message Types

Error handling for ILE and OPM is based on exception message types. An exception message is a message of one of these types:

Escape (*ESCAPE)
Indicates an error causing a program to end abnormally, without completing its work. The sending call stack entry up to the target call stack entry does not receive control after sending an escape exception message.

Status (*STATUS)
Describes the status of work being done by a program. You may receive control after sending a status exception message, depending on the way the target program handles the status message.

Notify (*NOTIFY)
Describes a condition requiring corrective action or a reply from the calling program. You may receive control after sending a notify exception message, depending on the way the target program handles the notify message.

Function Check
Function check is sent by the system as a part of the *default action* for an unhandled *ESCAPE exception. In ILE, a function check exception is a special exception message type that can only be sent by the system. In OPM, a function check is an *ESCAPE exception type with a message ID of CPF9999.

10.2.3.1 Sending an exception message

An exception message is sent in the following ways:

- Generated by the system

  OS/400 (including any HLL) generates an exception message to indicate a program error or status information.

- Message Handler API

  The Send program message (QMHSNDPM) API is used to send an exception (or informational) message.

- ILE API

  The Signal a Condition (CEESGL) bindable API is used to raise an ILE condition. This condition results in an escape or status exception message.

- Language-specific verbs
  - For ILE C/400, the *raise()* function generates a C signal that is represented as an exception message.
  - In ILE CL, the SNDPGMMSG command is used to send an exception.
  - RPG IV and ILE COBOL/400 do not have a similar function.

10.2.4 Types of Exception Handlers

As shown in Figure 94 on page 163, three types of exception handlers are supported by OS/400:

- Direct monitors
  - ILE C/400 *#pragma exception_handler* directive

  - Neither RPG IV, ILE COBOL, ILE CL nor OPM languages allow this capability.

- ILE condition handlers

  ILE provides the Register a User-Written Condition Handler (CEEHDLR) bindable API. This API allows you to identify a procedure at runtime that should be given control when an exception occurs.

- HLL-specific handlers

  Language-specific methods of handling exceptions.

  - ILE C/400 *signal* function
  - RPG IV **PSSR* and *INFSR* subroutines
  - ILE COBOL *USE* declarative for I/O error handling, imperatives in statement-scoped condition phrases such as *ON SIZE ERROR* or *AT INVALID KEY*
  - CL *MONMSG* command

    Please refer to the particular programming language manual for more information.

10.2.5 Exception Handler Priority

Priority of the types of handlers becomes important if HLL-specific error handlers are mixed with the additional ILE exception handler types in a single call stack entry.

RPG IV

1. Error indicator handling
2. INFSR I/O error handling
3. ILE condition handlers are called next

   Note:

   If multiple handlers have been registered, they are called in LIFO order

4. *PSSR error subroutine
5. RPG default handler (for unhandled exceptions)

ILE CL

1. HLL specific handler

All exception handlers for a call stack entry are called before an exception is percolated to the previous call stack entry and any active exception handlers on this call level are called.

10.2.6 Default Actions for Unhandled Exceptions

If an exception message remains unhandled after calling the handlers enabled for a control boundary call stack entry, the default action for the exception is taken. This action depends on the exception message type:

- **Escape**

  Put the escape message in the job log, and send a *function check* exception to the call stack entry where the *resume cursor* points. Exception processing now begins for the function check exception.

- **Status**

  Do not put the status message into the job log, and resume execution at the next statement in the sending procedure.

- **Notify**

  Keep the notify message in the job log, send the default reply to the notify message, and resume execution at the next statement in the sending procedure.

- **Function check**

  Do not put the function check message into the job log, move the *resume cursor* to the caller of the control boundary, and send the application failure escape exception message (CEE9901) to the caller of the control boundary.

10.2.7 Handling an Exception

An exception must be marked handled to resume program processing. Different methods apply, depending on handler type:

ILE C/400 direct monitors
Two methods:

- Control action specified on #pragma exception_handler statement
- QMHCHGEM API call inside exception handler

ILE condition handler
Refer to CEEHDLR API description in the System API Reference for the appropriate values.

Set a return code value inside exception handler.

HLL-specific handler
Exception is marked handled by the system prior to calling the exception handler.

10.2.8 Percolating an Exception

Some types of handlers have the option of being called to look at the exception, but decide to take no action. Different methods apply, depending on handler type:

ILE C/400 direct monitors
Do not use a control action on the #pragma exception_handler statement that marks the exception handled prior to being called and return from the handler without calling QMHCHGEM API to mark exception as handled.

ILE condition handler
Set a return code value inside exception handler.

Refer to CEEHDLR API description for the appropriate values.

HLL-specific handler
Exception is marked handled by the system prior to calling the exception handler, not possible to percolate.

10.2.9 Promoting an Exception

Some exceptions are modified by a handler to a different exception message type and message identifier. Different methods apply, depending on handler type:

ILE C/400 direct monitors
The QMHPRMM API is used to promote an escape or status exception message that has not yet been marked handled to a different escape or status exception. Some control of placement of the handle cursor is also available through this API. Refer to the QMHPRMM API documentation for specifics.

ILE condition handler
Set a return code value inside exception handler.

Refer to CEEHDLR API description for the appropriate values.

HLL-specific handler
This function is not available to HLL-specific handlers.

10.3 Steps in Exception Handling

What necessary steps are taken when an exception occurs in a program?

1. Exception is delivered.
2. System calls active exception handlers for the call stack entry where the exception was delivered.
3. If none of those handlers mark the exception as handled, it is moved to the callers call stack entry (*percolation*).
4. If the exception percolates to the control boundary, the default action for the exception message type is taken.

10.3.1 Exception Handling Flow

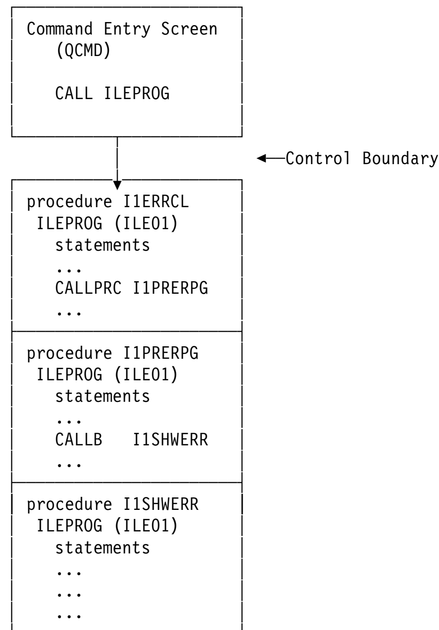
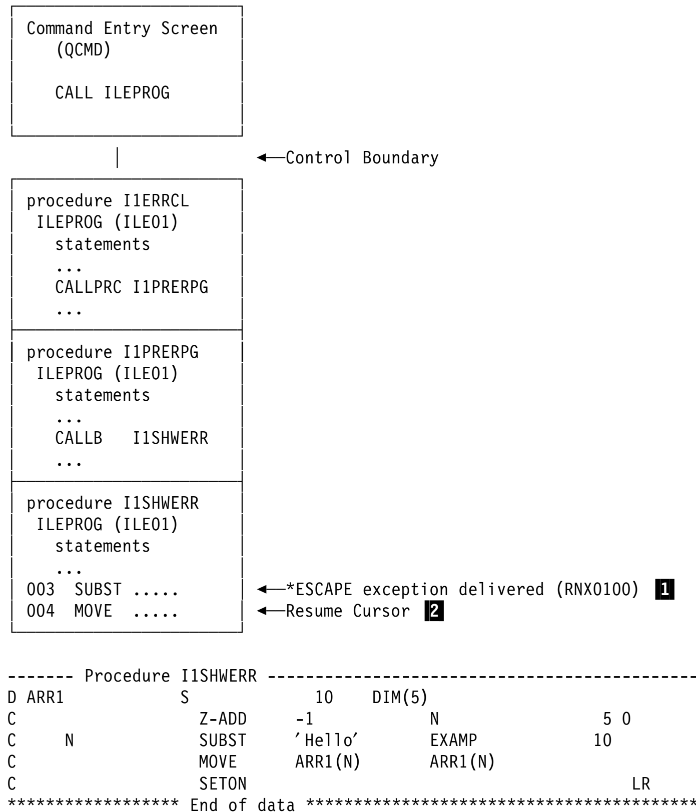


Figure 98. Exception Flow in Call Stack

In the previous call stack example, we call from the command entry screen an ILE program ILE01. In ILE01 procedure I1ERRCL has called procedure I1PRERPG that has called procedure I1SHWERR.




```
------- Procedure I1SHWERR --------------------------------------------
D ARR1            S              10    DIM(5)
C                   Z-ADD     -1             N                  5 0
C     N             SUBST     'Hello'        EXAMP             10
C                   MOVE      ARR1(N)        ARR1(N)
C                   SETON                                        LR
****************** End of data ****************************************
```

Figure 99. Exception Flow in Call Stack

Exception processing starts when an escape exception is sent and continues until the exception is handled.

Procedure I1SHWERR in ILE bound program ILE01 receives an *ESCAPE exception message (RNX0100) **1** when running the instruction SUBST.

When an exception is sent, the resume cursor is set to the instruction following the statement that received the exception **2**, and the handle cursor is set to the highest priority monitor in the call stack entry that received the exception.

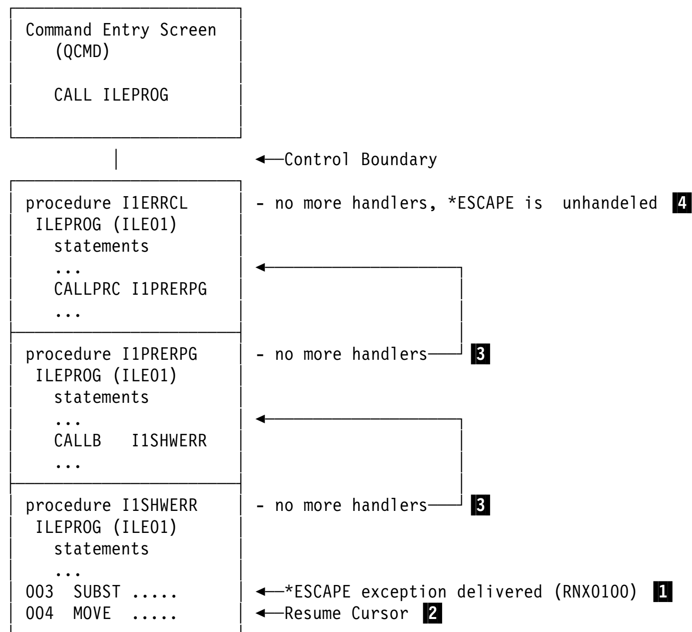


Figure 100. Exception Flow in Call Stack

3 Since in none of the procedures, any error handling is provided, all depends on the rules for the default HLL handlers.

4 In our example the ILE RPG/400 default handler percolates the unhandled *ESCAPE message up the call stack to the caller of the procedure.

This process continues until the control boundary is reached.

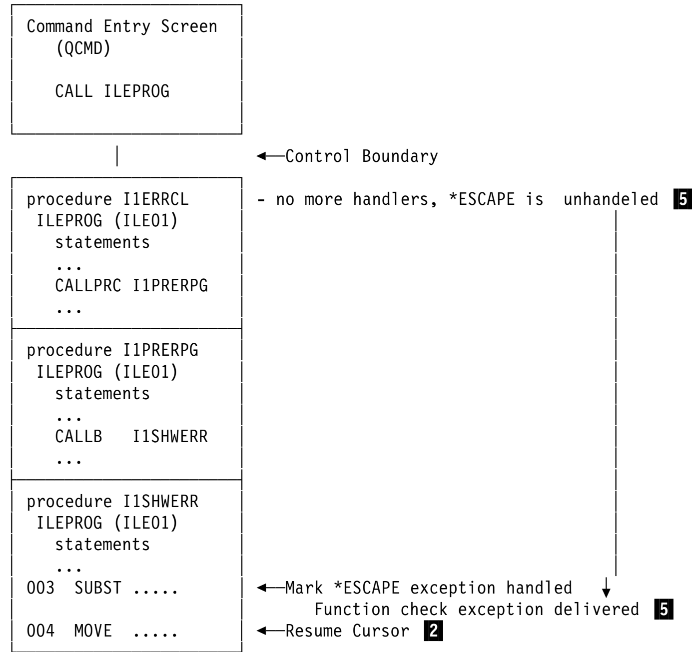


Figure 101. Exception Flow in Call Stack

5 At the control boundary, the unhandled *ESCAPE message is turned into a function check and delivered at the original starting point.

6 The ILE RPG/400 default handler handles this function check and issues an inquiry message RNQ0100. If you reply to the message with ″C″ (Cancel), the call stack is ended and the function check is percolated to the caller.

7 This process is repeated until we reach the control boundary. At the control boundary, every unhandled function check results in an *ESCAPE exception CEE9901 to the caller of the control boundary. See Figure 102.

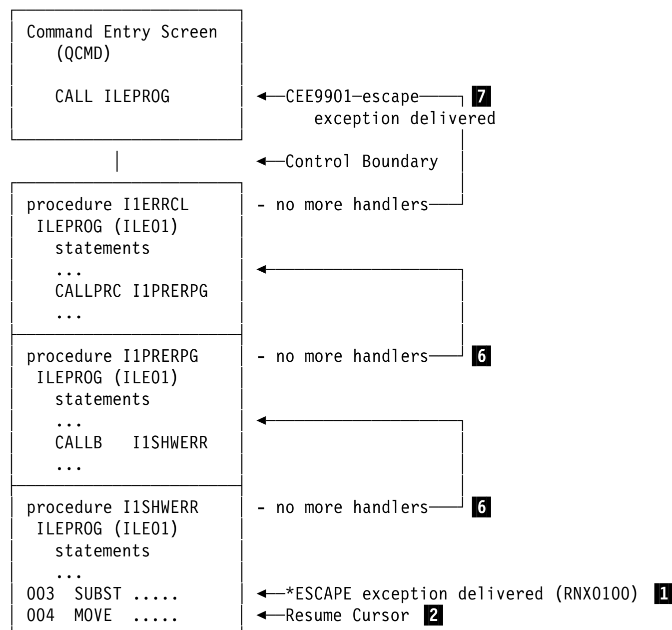


Figure 102. Exception Flow in Call Stack

10.3.1.1 Run an ILE Error Handling Example

You can run the example as described in 10.3.1, "Exception Handling Flow" on page 171 by using the following commands:

- ADDLIBLE GG244358
- CALL ILE01

Notice that when you are prompted for the first error, it is not clear that the first set of percolations have been done, since you are replying to a normal inquiry message.

```
Value used is out of range for the string operation (C G D F).


Type reply, press Enter.
  Reply . . .   ____________________________________________
```

Take the SYSREQ option 3, and display the JOBLOG (option 10), you should have the following information displayed:

```
3 > call ile01
 1     Value used is out of range for the string operation.
 2     Function check.  RNX0100 unmonitored by ILE01 at statement 0000000003,
      instruction X'0000'.
 3     Value used is out of range for the string operation (C G D F).
```

1 This describes message RNX0100 the original *ESCAPE.

2 This is the function check created by the control boundary after the unhandled *ESCAPE message

3 This is message RNQ0100, provided based on the function check by the ILE RPG/400 default handler.

If you answer all of the other messages (or take the default), and display the JOBLOG again, the function check has disappeared. This function check CPF9999 and the whole percolation process is not visible in the JOBLOG after the program has returned with the CEE9901 exception.

10.4 Comparing OPM and ILE Exception Handling

The major difference for the developer between OPM and ILE exception handling is the change of the message IDs. In most cases, the four-digit message number stays the same, while the three-character prefix changes, for example the equivalent for RPG0100 in OPM is now RNX0100. For the different behavior of both environments, checkout the following example provided on diskette in the back of this publication:

1. ADDLIBLE GG244358
2. CALL OPM01 (runs the same code with dynamic CALLs as ILE01 with bound calls)

10.4.1 Performance Impact

It is obvious, although not completely visible that the percolation process in ILE for unhandled errors requires more time then an unhandled error situation in OPM. So it is more important then ever to use your own exception handling where possible.

For RPG developers, the standard handling through the use of Error condition indicators, the I/O INFSR subroutines and the *PSSR is still an excellent solution.

The same set of programs used in the previous examples also have a counterpart with complete error handling capabilities implemented, they are called OPM02 and ILE02. Run them if you want to and notice the difference.

The programs also write some status information to an ERRORLOG file that is displayed at the end of the program.

10.4.2 ILE Condition Handler

Why should you use an ILE Condition handler if you run in a RPG IV and ILE CL environment? For C language developers, no HLL-specific handlers exist, so for the C environment this is the main solution, besides the use of direct monitors. Two reasons for condition handlers could be:

1. Language independent error handling interface
2. One service program that handles all application errors

   No need to change individual programs if changes in the error handling are required.

When your application is converted from OPM to ILE, you should not start eliminating all other means of error handling in your programs, since both methods are used at the same time. However, you should investigate the impact of the priority in which errors are handled when an ILE Condition handler and INFSR subroutines and *PSSR are all present in the same program.

Appendix A. Diskette Install Instructions

With this publication, there is a 3.5-inch diskette that contains the code for most examples quoted.

This code is provided for your use without restriction, to use for demonstrations, educational or as a framework to build your own application. Please understand that the examples are not intended to be fully operational applications, but are trying to show certain aspects of the new application development environment ILE, RPG IV, migration and Application Development Manager/400.

The diskette holds a library called GG244358. Further instructions on installing and prerequisites follow.

Prerequisites: In order to be able to compile the restore programs and the examples in the library, the products required are:

- OS/400 5763-SS1 (V3R1)
- ILE RPG 5763-RG1 (V3R1)

To run the Application Development Manager/400 scenarios, you need:

- ADTS/400 5763-PW1 (V3R1)
- Feature J1 and J2

Installation: Follow the steps as described:

1. Sign on as QSECOFR on a PS/2.

   A user profile with equal rights as QSECOFR works as well.
2. Create on the AS/400 system, a folder named GG244358 in the root directory.
3. Make the folder available as a shared folder for the PC.
4. Insert the diskette in drive A.
5. Copy all of the files from the diskette into shared folder GG244358.
6. Add a member to a source file (QCLSRC) in your own library:

```
ADDPFM FILE(yourlib/QCLSRC) MBR(CLP4358) SRCTYPE(CLP)
```

7. Use the CPYFRMPCD to copy the install program source from the folder:

```
CPYFRMPCD FROMFLR(GG244358) TOFILE(yourlib/QCLSRC)  +
                            FROMDOC(CLP4358.CLP)    +
                            TOMBR(CLP4358)
```

   As a result of the next action, you have a library installed called GG244358. Verify before running the program that a library of that name does not already exist on your system, otherwise, it is deleted.
8. Create and run this CL program.
9. Delete the following objects; they are of no further use:
   - Folder: GG244358
   - Source and program CLP4358 in yourlib

© Copyright IBM Corp. 1995

Appendix B. RPG IV Coding Examples

B.1.1 Using Pointers in RPG IV

This example is in source file QPOINTER in library GG244358.

RTVPGMMOD the command specifications

```
/*--------------------------------------------------------------------------*/
/* Keyword information:                                                     */
/*                                                                          */
/* FROMLIB    - Specify the library that you want to have searched for      */
/*              ILE programs                                                */
/*                                                                          */
/* OUTFILE    - Specify the File where you want to receive the information, */
/*              if the file does not exist it is created with a record      */
/*              length of 508.                                              */
/*              If the file name exists, it is used to store the information */
/*                                                                          */
/* MBROPT     - Specify when an existing file is used, whether you want     */
/*              to replace or add the records to it.                        */
/*                                                                          */
/* The program called by this command is RTVPGMMOD                          */
/*                                                                          */
/* Author: ITSC (ILE Residency)                                             */
/* Date  : 05-27-94                                                         */
/*                                                                          */
             CMD        PROMPT('Retrieve PGM modules')
             PARM       KWD(FROMLIB) TYPE(*NAME) LEN(10) MIN(1) +
                          PROMPT('From Library')
             PARM       KWD(OUTFILE) TYPE(QUAL1) MIN(1) +
                          PROMPT('Outfile name')
             PARM       KWD(MBROPT) TYPE(*CHAR) LEN(8) RSTD(*YES) +
                          DFT(*REPLACE) VALUES(*REPLACE *ADD) +
                          PROMPT('Replace or add records')
QUAL1:       QUAL       TYPE(*NAME) LEN(10)
             QUAL       TYPE(*NAME) LEN(10) DFT(*CURLIB) SPCVAL(*CURLIB) +
                          PROMPT('Library name')
```

RTVPGMMODC the CLLE program

```
/*------------------------------------------------------------------------*/
/* This source is used to create the first module of the RTVPGMMOD program    */
/*  it will do all the checking and creating if necessary for the           */
/*  environment.                                                              */
/*                                                                            */
/* The flow thru the program is described at the appropriate place.          */
/*                                                                            */
/* Author: ITSC (ILE Residency)                                               */
/* Date  : 05-27-94                                                           */
/*                                                                            */
/*------------------------------------------------------------------------*/
/* RTVPGMMODC Retrieve Program module information into a File */
/*                                                            */
             PGM        PARM(&LIB &OUTFILLIB &REPLACE)
/*------------------------------------------------------------------------*/
/* Input variables                                                            */
/*------------------------------------------------------------------------*/
             DCL        VAR(&LIB) TYPE(*CHAR) LEN(10)
             DCL        &OUTFILLIB *CHAR LEN(20)
             DCL        VAR(&REPLACE) TYPE(*CHAR) LEN(8)
/*------------------------------------------------------------------------*/
/* Program variables                                                          */
/*------------------------------------------------------------------------*/
                                                      /*                 */
             DCL        &OFIL *CHAR LEN(10)           /* Outfile Filename */
             DCL        &OLIB *CHAR LEN(10)           /* Outfile Library  */
             DCL        VAR(&CRTUSRSPC) TYPE(*CHAR) + /* Create USRSPC    */
                          LEN(1) VALUE('N')           /* object the       */
                                                      /* first time       */
             DCL        &WS *CHAR LEN(10)             /* Workstation name */
             DCL        &TXT  *CHAR LEN(80)           /* BRKMSG text      */
             DCLF       FILE(QADSPOBJ)                /* DSPOBJD outfile  */
             MONMSG     MSGID(CPF0000)                /* Catch all Errors */
/*------------------------------------------------------------------------*/
/* Start the mainline of the program                                          */
/*  Establish the process environment                                         */
/*------------------------------------------------------------------------*/
             DLTF       FILE(QTEMP/##DSPOBJD)         /* Delete DSPOBJD   */
                                                      /* Outfile          */
             MONMSG     MSGID(CPF2105)                /* Ignore not found */
             DSPOBJD    OBJ(&LIB/*ALL) OBJTYPE(*PGM) + /* Display all PGMs */
                          OUTPUT(*OUTFILE) +          /* in the library   */
                          OUTFILE(QTEMP/##DSPOBJD)
             OVRDBF     FILE(QADSPOBJ)  +             /* Override the file*/
                          TOFILE(QTEMP/##DSPOBJD) +   /* to DSPOBJD output*/
                          SECURE(*YES)                /* file             */
             CHKOBJ     OBJ(QTEMP/##MODLST) +         /* Check existence  */
                          OBJTYPE(*USRSPC)            /* of Userspace     */
             MONMSG     MSGID(CPF9801)  +             /* if NOT, set Yes  */
                          EXEC(CHGVAR VAR(&CRTUSRSPC) + /* indication on  */
                          VALUE('Y'))

             CHGVAR     VAR(&OLIB) +                  /* Assign outfile   */
                          VALUE(%SST(&OUTFILLIB 11 10)) /* library        */
             CHGVAR     VAR(&OFIL) +                  /* Assign outfile   */
                           VALUE(%SST(&OUTFILLIB  1 10)) /* filename      */
             CHKOBJ     OBJ(&OLIB/&OFIL) OBJTYPE(*FILE)  /* Check existence */
```

```
          MONMSG     MSGID(CPF9801) +                  /* If NOT, create   */
                       EXEC(CRTPF FILE(&OLIB/&OFIL) +  /* the Outfile      */
                       RCDLEN(508) +
                       TEXT('PGM Module reference information')  +
                       SIZE(10000 10000 10))

          IF         COND(&REPLACE *EQ '*REPLACE') +   /* Clear the outfile*/
                     THEN(CLRPFM  FILE(&OLIB/&OFIL))   /* if requested     */
          OVRDBF     FILE(MODLST) TOFILE(&OLIB/&OFIL)  /* Override the file*/
                                                       /* to the outfile   */

/*-----------------------------------------------------------------------------*/
/* Read and process the OBJD program records of the requested library         */
/*-----------------------------------------------------------------------------*/
 AGAIN:     RCVF       RCDFMT(QLIDOBJD)                /* read a record    */
          MONMSG     MSGID(CPF0864) +                  /* If End-Of-File   */
                       EXEC(GOTO CMDLBL(END))          /* goto END         */

          IF         COND((&ODOBAT *EQ  'RPGLE   ') +  /* If type is ILE   */
                       *OR (&ODOBAT *EQ 'CLLE    ') +
                       *OR (&ODOBAT *EQ 'CBLLE   ') +
                       *OR (&ODOBAT *EQ 'CLE     ') +
            THEN(DO)
              CALLPRC PRC(RTVPGMMODR) +                /* Call the retrieve*/
                      PARM(&ODOBNM &ODLBNM &CRTUSRSPC) /* program with the */
                                                       /* API interface    */
              IF      COND(&CRTUSRSPC *EQ 'E') +       /* If an error      */
                      THEN(GOTO CMDLBL(ERROR))         /* occurred, end the*/
                                                       /* command          */
              CHGVAR  VAR(&CRTUSRSPC) VALUE('N')       /* Set create USRSPC*/
                                                       /* to No            */
            ENDDO
          GOTO       CMDLBL(AGAIN)                     /* Read another     */
                                                       /* record           */
/*-----------------------------------------------------------------------------*/
/* Normal or Abnormal ending of the program                                   */
/*-----------------------------------------------------------------------------*/
ERROR:
          RTVJOBA    JOB(&WS)                          /* Retrieve the WSid*/
          CHGVAR     VAR(&TXT) +                       /* Build the error  */
                     VALUE(&ODOBNM *bcat &ODLBNM  +    /* message text     */
                     *bcat 'An error has occured, +
                     the program has stopped, +
                     check the JOBLOG and try again')
          SNDBRKMSG  MSG(&TXT) TOMSGQ(&WS)             /* Send the errormsg*/
END:
          ENDPGM                                       /* End the program  */
```

RTVPGMMODR the RPGLE program

```
/*-----------------------------------------------------------------------------*/
/* This source is used to create the second module of the RTVPGMMOD program   */
/* and will create a USRSPC if required.                                       */
/*                                                                             */
/* - The userspace will be filled with the information supplied by API        */
/*   QBNLPGMI, which will give a list of all modules in an ILE program        */
/*   using format PGML0100                                                     */
/*                                                                             */
/* - All module information is written to an outfile with recl = 508         */
/*                                                                             */
/* - Check the API manual SC41-8223 for more information on the               */
/*   structure of list entries (Chapter 2) and specifics on QBNLPGMI          */
/*                                                                             */
/* Author: ITSC (ILE Residency)                                                */
/* Date  : 05-27-94                                                            */
/*                                                                             */
/*-----------------------------------------------------------------------------*/
 *  Example using POINTERS for access to user spaces
 *
FMODLST    O  A F  508        DISK                                        Output file
DNAME             S             20    INZ('##MODLST  QTEMP     ')         Name/Lib userspace
D ATTRIBUTE       S             10    INZ('REF')                           Attribute usrspc
D INIT_SIZE       S              9B 0 INZ(10000)                           Init size usrspc
D INIT_VALUE      S              1    INZ(' ')                             Init value
D AUTHORITY       S             10    INZ('*CHANGE')                       Auth usrspc
D TEXT            S             50    INZ('Temporary Space Created by +    Text usrspc
D                                     RTVPGMMOD')
DPTR              S               *                                       Pointer field
DSPACE            DS                  BASED(PTR)                          Assign the start of
 *                                                                         the userspace
D SP1                        32767                                        First subfield in
 *                                                                         the userspace
 * ARR is used with offset to set the pointer to array                    Re-align Usrspc with
DARR                             1    OVERLAY(SP1) DIM(32767)              1 byte array
 * Offset is pointing to start of array                                   Offset value for the
DOFFSET                          9B 0 OVERLAY(SP1:125)                     start of the list
 *                                                                         data section in
 *                                                                         the userspace
 * Size has number of module names retrieved                              Number of list
DSIZE                            9B 0 OVERLAY(SP1:133)                     entries
DMODPTR           S               *                                       Pointer field
DMOD_ENTRYS       C                   CONST(500)                          Initial Array length
DMODARR           S            508    BASED(MODPTR) DIM(MOD_ENTRYS)       Re-align the start
 *                                                                         of list entries
 *                                                                         in the userspace
DX                S             11  0                                     Array index value
DMOD_INFO         DS                                                      Re-define of ONE
 *                                                                         list entry
D PGM_NAME                      10
D PGM_LIB                       10
D MOD_NAME                      10
D MOD_LIB                       10
D SRC_FILE                      10
D SRC_LIB                       10
D MEMBER                        10
D MOD_ATTR                      10
D                              428                                        Filler
```

```
 * Parameter fields for the QBNLPGMI Application Program Interface (API)
DFORMAT          S              8    INZ('PGML0100')                   API format name
DPGMLIB          S             20                                      Program/Library
D LIB            S             10                                       Library
D PGM            S             10                                       Program
DCRTUSRSPC       S              1                                      Y=Yes, N=No
DERROR           DS                                                    Error code
D IN                            9B 0 INZ(0)                            Bytes IN
D OUT                           9B 0 INZ(0)                            Bytes OUT
D EXCEPTION                     7                                      Message-id
D RESERVED                      1
D EXCEP_DATA                   40                                      Message data
 * End parameter fields
 *-------------------------------------------------------------------------------------
 * Receive program and library name you want to process
 *  as well as a signal whether you want the Userspace to be created
 *  a 'Y' means YES, every other value means NO
 *-------------------------------------------------------------------------------------
C     *ENTRY        PLIST
C                   PARM                    PGM
C                   PARM                    LIB
C                   PARM                    CRTUSRSPC
 *-------------------------------------------------------------------------------------
 * Create the Userspace if necessary
 *-------------------------------------------------------------------------------------
C                   IF        CRTUSRSPC = 'Y'
C                   CALL      'QUSCRTUS'                          99
C                   PARM                    NAME
C                   PARM                    ATTRIBUTE
C                   PARM                    INIT_SIZE
C                   PARM                    INIT_VALUE
C                   PARM                    AUTHORITY
C                   PARM                    TEXT
C                   ENDIF
 *-------------------------------------------------------------------------------------
 * Get a pointer to the user-space
 *-------------------------------------------------------------------------------------
C                   CALL      'QUSPTRUS'                               Call API
C                   PARM                    NAME
C                   PARM                    PTR
 *-------------------------------------------------------------------------------------
 * Call the API to list the modules in the ILE program
 *-------------------------------------------------------------------------------------
C                   EVAL      PGMLIB = PGM + LIB
C                   EVAL      IN = %SIZE(ERROR)                        Move the field size
 *                                                                      of ERROR field
C                   CALL      'QBNLPGMI'                               Call API
C                   PARM                    NAME
C                   PARM                    FORMAT
C                   PARM                    PGMLIB
C                   PARM                    ERROR
C                   SELECT
C                   WHEN      EXCEPTION <> *BLANKS                     When error occurs
 *                                                                      move information
 *                                                                      to program input/
 *                                                                      output parameters
C                   EVAL      %SUBST(PGM:1:10) = *BLANKS               Clear pgm field
C                   EVAL      %SUBST(PGM:1:7) = EXCEPTION              Return MSGID
C                   EVAL      %SUBST(LIB:1:10) = EXCEP_DATA            Return data
```

```
C                   EVAL      CRTUSRSPC = 'E'                              Set Error
C                   GOTO      END
C                   ENDSL
 *-----------------------------------------------------------------------------------
 * Set the based pointer for the module array
 *
 *  1. Field OFFSET contains the offset value of the start of the list data section
 *     in the array. (in this example X'104')
 *     So the byte we want the address of is (104 + 1)
 *  2. The pointer value of that field is moved into pointer field MODPTR
 *  3. Automatically the start of the array MODARR is re-aligned to this pointer value
 *     (see the definition of MODARR)
 *-----------------------------------------------------------------------------------
C                   EVAL      MODPTR = %ADDR(ARR(OFFSET + 1))
 *-----------------------------------------------------------------------------------
 * Write the module information to the outfile
 *-----------------------------------------------------------------------------------
C                   EVAL      X = 0                                        Set array index nbr
C                   IF        SIZE > MOD_ENTRYS                            Limit the number of
C                   EVAL      %SUBST(PGM:1:10) = 'MOD ARRAY'   Set error condition
C                   EVAL      %SUBST(LIB:1:10) = 'TOO SMALL.'   message
C                   EVAL      CRTUSRSPC = 'E'                              Set Error
C                   GOTO      END
C                   ENDIF
 *
C                   DO        SIZE                                         Do as many times as
 *                                                                          list entries avail.
C                   ADD       1             X                              Increase index nbr
C                   EVAL      MOD_INFO = MODARR(X)                         Move array entry to
 *                                                                          data structure
C                   WRITE     MODLST        MOD_INFO                       Write record
C                   ENDDO
 *-----------------------------------------------------------------------------------
 * End of the program
 *-----------------------------------------------------------------------------------
C     END           TAG
C                   RETURN
```

Appendix C. Migration Information

This appendix contains all kindsof migration-related information helpful in the process of going through the migration exercise.

C.1.1.1 Warning/Error Messages during CVTRPGSRC Execution

The following list contains warning and error messages you could receive in the print conversion report produced during the execution of CVTRPGSRC command. For further information about the messages, refer to the message file QRPGLEMSG in library QRPGLE.

RNM0501 Unable to determine RPG specification type.

Explanation: This RPG specification can either be interpreted as one of: &P -- a data structure subfield &P -- a program-described file field &P OR one of &P -- a rename of an externally-described data structure field &P -- a rename of an externally-described file field

User Response: The CVTRPGSRC command cannot properly interpret the specification; a data structure is assumed and a Definition specification is produced. A block of comments containing the corresponding Input specification code is also produced. If Input specification code is required, delete the Definition specification code and remove the comments from the corresponding Input specifications.

RNM0502 Comment has been truncated.

Explanation: The record length of the source physical file specified on the TOFILE parameter is not long enough to contain the entire comment.

User Response: Increase the record length of the TOFILE source physical file and try the conversion again.

RNM0503 The corresponding File Description specification is missing.

Explanation: The Extension specifications or Line Counter specifications are not valid in RPG IV. New File Description specification keywords (RAFDATA, FORMLEN and FORMOFL) are used to perform this function. The File Description specification must be contained in this source member in order for these keywords to be produced. The probable cause of this problem is that the File Description specification is contained in a /COPY member.

User Response: Specify *YES for the EXPCPY parameter of the CVTRPGSRC command when converting the primary RPG member to expand /COPY file member(s) into the converted source or manually add the RAFDATA, FORMLEN or FORMOFL keywords to the File Description specification prior to compiling the converted source.

RNM0504 The corresponding Extension specification is missing.

Explanation: The Extension specification is not valid in RPG IV and a new File specification RAFDATA keyword has been created to identify the file that contains the data records. The Extension specification must be contained in this source member in order for the RAFDATA keyword to be produced. The probable cause of this problem is that the Extension specification is contained in a /COPY member.

User Response: Specify *YES for the EXPCPY parameter of the CVTRPGSRC command when converting the primary RPG member to expand /COPY file member(s) into the converted source or manually add the RAFDATA keyword prior to compiling in RPG IV.

RNM0505 The corresponding Line counter specification is missing.

Explanation: The Line counter specification is not valid in RPG IV and new File specification keywords, FORMLEN and FORMOFL, are used to identify the number of printing lines available and the overflow line. The Line specification must be contained in this source member in order for the FORMLEN and FORMOFL keyword to be produced. The probable cause of this problem is that the Line specification is contained in a /COPY member.

User Response: Specify *YES for the EXPCPY parameter of the CVTRPGSRC command when converting the primary RPG member to expand /COPY file member(s) into the converted source or manually add the FORMLEN and FORMOFL keywords prior to compiling in RPG IV.

© Copyright IBM Corp. 1995

RNM0506 FREE operation code is not supported in RPG IV.

Explanation: The RPG III or RPG/400 program contains the FREE operation code which is not supported in RPG IV.

User Response: Remove the FREE operation and replace it with alternative code so that the programming logic is not affected prior to compiling the converted source.

RNM0507 RPG IV does not support the auto report feature.

Explanation: *AUTO is detected in the Output specification and the FROMFILE member type is not RPT or RPT38. When an auto report source member type is detected in an RPG III or RPG/400 program, the CVTRPGSRC command calls the CRTRPTPGM command to expand the source member prior to converting it.

User Response: Assign the correct source member type (RPT or RPT38) to the FROMFILE member and try the conversion again.

RNM0508 /COPY compiler directive found.

Explanation: In order for this RPG IV source to compile correctly, ensure that all /COPY source members included in this source member have also been converted to RPG IV.

User Response: Ensure that all /COPY source members are converted prior to compiling in RPG IV. In some cases, problems may result when attempting to convert and compile source members that make use of the /COPY compiler directive. If this situation results, specify *YES for the EXPCPY parameter on the CVTRPGSRC command to expand the /COPY member(s) into the converted source. For further information see the ILE RPG/400 Programmers Guide.

RNM0509 The type of specification is not valid or is out of sequence.

Explanation: The member is either not an RPG source member or contains compilation errors.

User Response: Correct any compilation errors before attempting the conversion by first compiling the program using the RPG/400 or RPG III compiler. Try the conversion again.

RNM0510 /TITLE compiler directive removed from compile-time data.

Explanation: In RPG/400 and RPG III the compiler directive /TITLE was allowed in the compile-time data section (**) of the program. This is not allowed in RPG IV and the Conversion Aid has removed this compiler directive from the converted code.

RNM0511 CALL operation code found.

Explanation: RPG specifications that contain CALL operation codes have been identified because the user may wish to: &P -- change the CALL operation code to CALLB to take advantage of static binding &P -- convert all programs in an application to RPG IV

RNM0512 Not enough compile-time data records found.

Explanation: The CVTRPGSRC command expects more groups of alternate collating sequence records, file translation records and compile-time data records then were found in this source member. The converted source member may contain errors.

User Response: Correct any compilation errors before attempting the conversion again by first compiling the program using the RPG/400 or RPG III compiler.

RNM0513 The CVTRPGSRC command has detected more compile-time data records than expected.

Explanation: Either the Control Specification (which indicates the existence of file translation and alternate collating sequence records) or some array definitions (Extension Specifications) are not contained within this source member. Therefore the CVTRPGSRC command must try to interpret the type of compile-time data records in order to convert them properly.

User Response: If the converted source member does not compile properly after the conversion process either: &P -- use the listing generated by the ILE RPG compiler to help correct any errors that have been detected &P OR &P -- specify *YES for the EXPCPY parameter of the CVTRPGSRC command when converting the primary RPG member to expand /COPY file members into the converted source

RNM0514 More than 32764 lines added to converted source member.

Explanation: During the conversion process, more than 32764 source code records have been generated in the converted source member.

User Response: The maximum number of records supported by the Source Entry Utility (SEU) is 32764. If you wish to edit this converted source member an editor other than SEU must be used.

RNM0516 Program described file has packed field defined to be greater than 30 digits.

Explanation: RPG III or RPG/400 defaults a packed field of 16 bytes for a program described file to be of length 30 digits. RPG IV does not allow this default. Unless the program defines this field somewhere else with a length of 30 digits a compiler error will result.

User Response: If this field is not defined somewhere else in your program create a stand-alone Definition specification with length 30 digits to bypass this problem. For your convenience a sample Definition specification has been generated as a comment in your source code. If required remove the asterisk from position 7 and move this line of source code into the Definition specification portion of the program.

RNM0517 DEBUG operation code is not supported in RPG IV.

Explanation: The RPG III or RPG/400 program contains the DEBUG operation code which is not supported in RPG IV.

User Response: Remove the DEBUG operation code and replace it with alternative code so that the programming logic is not affected prior to compiling the converted source.

RNM0518 Compile-time array definitions have been merged with a data structure subfield.

Explanation: An array has been defined on an Extension specification and on a data structure subfield in the RPG/400 source member. The CVTRPGSRC command has merged these two definitions into a single Definition specification in the converted source member. This merging process may have changed the order of the arrays in the converted source member. If this has happened the compile-time array records must either be reordered or named using the new **keyword syntax. However, the records cannot be ″named″ or reordered by the CVTRPGSRC command as not all the array definitions are contained within this source member.

User Response: Specify *YES for the EXPCPY parameter of the CVTRPGSRC command when converting the primary RPG member to expand /COPY file member(s) into the converted source or manually reorder the compile-time data records prior to compiling the converted source.

Appendix D. Development environment example code

D.1.1 ADM Setup

This is the code of the program CRTADMENV as mentioned in 8.2.2, "Setup of the Application Development Manager/400 Environment" on page 129 in the setup of ADM.

```
/*-CRTADMENV-----------------------------------------------------*/
/*                                                               */
/* Create the ADM example environment and import the sources     */
/*  to be used from the example library source file              */
/*                                                               */
/* Import the sources from the QMLGSRC file for the mailing      */
/*  application                                                  */
/*                                                               */
/*---------------------------------------------------------------*/
            PGM
            DCL        VAR(&ADM) TYPE(*CHAR) LEN(4) VALUE('MLGI')
            DCL        VAR(&ADMSRC) TYPE(*CHAR) LEN(10) +
                         VALUE('QMLGSRC   ')
            DCL        VAR(&XMPLIB) TYPE(*CHAR) LEN(10) +
                         VALUE('GG244358  ')

/*                                Name......Name......Name......Name......*/
            DCL        VAR(&PF      ) TYPE(*CHAR) LEN(2000) +
                         VALUE('MLGREFP   MLGMSTP                       +
                                                                         ')
/*                                Name......Name......Name......Name......*/
             DCL        VAR(&LF      ) TYPE(*CHAR) LEN(2000) +
                          VALUE('MLGMSTL   MLGMSTL2  MLGMSTL3  MLGNAML   +
                                                                         ')
 /*                                Name......Name......Name......Name.....*/
             DCL        VAR(&DSPF    ) TYPE(*CHAR) LEN(2000) +
                          VALUE('MLGINQD   MLGMNUD   MLGMTND   MLGNAMD   +
                                                                         ')
 /*                                Name......Name......Name......Name.....*/
             DCL        VAR(&CLLE    ) TYPE(*CHAR) LEN(2000) +
                          VALUE('MLGMNUC   MLGMTNC   MLGRPTC   MLGRPTC2  +
                                                                         ')
 /*                                Name......Name......Name......Name.....*/
             DCL        VAR(&RPGLE   ) TYPE(*CHAR) LEN(2000) +
                          VALUE('MLGINQR   MLGNAMR   MLGLBLR   MLGLBLR2  +
                                 MLGRPTR   MLGMTNR                      +
                                                                         ')
             DCL        VAR(&MBR) TYPE(*CHAR) LEN(10)
             DCL        VAR(&START) TYPE(*DEC) LEN(3 0) VALUE(1) /* +
                          Number of times that a library name is +
                          retrieved from the Library Table */

/*---------------------------------------------------------------*/
/* Create a project                                              */

             CRTPRJ     PRJ(&ADM) SHORTPRJ(&ADM) TEXT('ADM ILE +
                          Mailing example project')
/*---------------------------------------------------------------*/
/* Create the production environment                             */
```

© Copyright IBM Corp. 1995

```
             CRTGRP     PRJ(&ADM) GRP(PRD) SHORTGRP(PRD) +
                          PARENT(*NONE) NOTIFY(*DEVELOPER) +
                          TEXT('Production environment')
/*-----------------------------------------------------------------*/
/* Create the developers environment                               */

             CRTGRP     PRJ(&ADM) GRP(DEV) SHORTGRP(DEV) PARENT(PRD) +
                          NOTIFY(*DEVELOPER) TEXT('Development +
                          environment')
/*=================================================================*/
/*-----------------------------------------------------------------*/
/* Import the QDFT BLDOPT                                          */

             IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(QDFT) +
                          PRJ(&ADM) GRP(PRD) TYPE(BLDOPT) +
                          PART(QDFT) LANG(*NONE) SRCFILE(*TYPE) +
                          TEXT('Default Build Option  ')
/*-----------------------------------------------------------------*/
/* Import the PF    member parts                                   */
/*                                                                 */
             CHGVAR     VAR(&START) VALUE(1)
AGAIN1A:
             CHGVAR     VAR(&MBR) VALUE(%SST(&PF       &START 10))
             IF         COND(&MBR *EQ '          ') THEN(GOTO +
                          CMDLBL(AGAIN2))
             IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(&MBR) +
                          PRJ(&ADM) GRP(PRD) TYPE(DDSSRC) +
                          PART(&MBR) LANG(PF) SRCFILE(*TYPE) +
                          TEXT(*TEXT)
             CHGVAR     VAR(&START) VALUE(&START + 10)
             GOTO       CMDLBL(AGAIN1A)

/*-----------------------------------------------------------------*/
/* Import the LF    member parts                                   */
/*                                                                 */
AGAIN2:      CHGVAR     VAR(&START) VALUE(1)
AGAIN2A:
             CHGVAR     VAR(&MBR) VALUE(%SST(&LF       &START 10))
             IF         COND(&MBR *EQ '          ') THEN(GOTO +
                          CMDLBL(AGAIN3))
             IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(&MBR) +
                          PRJ(&ADM) GRP(PRD) TYPE(DDSSRC) +
                          PART(&MBR) LANG(LF) SRCFILE(*TYPE) +
                          TEXT(*TEXT)
             CHGVAR     VAR(&START) VALUE(&START + 10)
             GOTO       CMDLBL(AGAIN2A)

/*-----------------------------------------------------------------*/
/* Import the DSPF  member parts                                   */
/*                                                                 */
AGAIN3:      CHGVAR     VAR(&START) VALUE(1)
AGAIN3A:
             CHGVAR     VAR(&MBR) VALUE(%SST(&DSPF     &START 10))
             IF         COND(&MBR *EQ '          ') THEN(GOTO +
                          CMDLBL(AGAIN4))
             IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(&MBR) +
                          PRJ(&ADM) GRP(PRD) TYPE(DDSSRC) +
                          PART(&MBR) LANG(DSPF) SRCFILE(*TYPE) +
                          TEXT(*TEXT)
```

```
            CHGVAR     VAR(&START) VALUE(&START + 10)
            GOTO       CMDLBL(AGAIN3A)

/*-----------------------------------------------------------------*/
/* Import the CLLE  member parts                                   */
/*                                                                 */
AGAIN4:     CHGVAR     VAR(&START) VALUE(1)
AGAIN4A:
            CHGVAR     VAR(&MBR) VALUE(%SST(&CLLE     &START 10))
            IF         COND(&MBR *EQ '          ')  THEN(GOTO +
                         CMDLBL(AGAIN5))
            IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(&MBR) +
                         PRJ(&ADM) GRP(PRD) TYPE(CLLESRC) +
                         PART(&MBR) LANG(CLLE) SRCFILE(*TYPE) +
                         TEXT(*TEXT)
            CHGVAR     VAR(&START) VALUE(&START + 10)
            GOTO       CMDLBL(AGAIN4A)

/*-----------------------------------------------------------------*/
/* Import the RPGLE member parts                                   */
/*                                                                 */
AGAIN5:     CHGVAR     VAR(&START) VALUE(1)
AGAIN5A:
            CHGVAR     VAR(&MBR) VALUE(%SST(&RPGLE    &START 10))
            IF         COND(&MBR *EQ '          ')  THEN(GOTO +
                         CMDLBL(END   ))
            IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) MBR(&MBR) +
                         PRJ(&ADM) GRP(PRD) TYPE(RPGLESRC) +
                         PART(&MBR) LANG(RPGLE) SRCFILE(*TYPE) +
                         TEXT(*TEXT)
            CHGVAR     VAR(&START) VALUE(&START + 10)
            GOTO       CMDLBL(AGAIN5A)

END:
/*-----------------------------------------------------------------*/
/* Import the Build option for PROGRAMS                            */

            IMPPART    OBJ(&XMPLIB/&ADMSRC) OBJTYPE(*SRC) +
                         MBR(PGMDFTBLD) PRJ(&ADM) GRP(PRD) +
                         TYPE(BLDOPT) PART(PGMDFTBLD) LANG(*NONE) +
                         SRCFILE(*TYPE) TEXT('Default Build options ')
/*-----------------------------------------------------------------*/
/* Import the Message file                                         */

            IMPPART    OBJ(&XMPLIB/MLGMSGF) OBJTYPE(*MSGF) +
                         PRJ(&ADM) GRP(PRD) TYPE(MSGF) +
                         PART(MLGMSGF) LANG(*NONE) SRCFILE(*TYPE) +
                         TEXT('Message file ')
/*-----------------------------------------------------------------*/
            ENDPGM
```

D.1.2 Copy Build Options

```
/*-CPYBLDOPT------------------------------------------------------*/
/*                                                                */
/* Import all necessary build options for the programs           */
/*                                                                */
/* Copy the data for MLGMSTP from GG244358 into MLGI.PRD         */
/*                                                                */
/*----------------------------------------------------------------*/
             PGM
             DCL        VAR(&ADM) TYPE(*CHAR) LEN(4) VALUE('MLGI')
             DCL        VAR(&ADMBLD) TYPE(*CHAR) LEN(10) +
                          VALUE('QMLGBLDOPT')
             DCL        VAR(&XMPLIB) TYPE(*CHAR) LEN(10) +
                          VALUE('GG244358  ')
             DCL        VAR(&ADMPRD) TYPE(*CHAR) LEN(10) +
                          VALUE('MLGI.PRD  ')

 /*                                 Name......Name......Name......Name.....*/
             DCL        VAR(&BLDOPT  ) TYPE(*CHAR) LEN(2000) +
                          VALUE('MLGMNUC   MLGINQR   MLGMTNC   MLGRPTC   +
                                 MLGRPTC2                                +
                                                                         ')

             DCL        VAR(&MBR) TYPE(*CHAR) LEN(10)
             DCL        VAR(&START) TYPE(*DEC) LEN(3 0) VALUE(1) /* +
                          Number of times that a library name is +
                          retrieved from the Library Table */

/*----------------------------------------------------------------*/
/* Import the BLDOPT member parts                                 */
/*                                                                */
             CHGVAR     VAR(&START) VALUE(1)
AGAIN1A:
             CHGVAR     VAR(&MBR) VALUE(%SST(&BLDOPT   &START 10))
             IF         COND(&MBR *EQ '          ')  THEN(GOTO +
                          CMDLBL(END))
             IMPPART    OBJ(&XMPLIB/&ADMBLD) OBJTYPE(*SRC) MBR(&MBR) +
                          PRJ(&ADM) GRP(PRD) TYPE(BLDOPT) +
                          PART(&MBR) LANG(*NONE) SRCFILE(*TYPE) +
                          TEXT(*TEXT)
             CHGVAR     VAR(&START) VALUE(&START + 10)
             GOTO       CMDLBL(AGAIN1A)

/*----------------------------------------------------------------*/

END:
/*----------------------------------------------------------------*/
/* Copy the MLGMSTP file                                          */

             CPYF       FROMFILE(&XMPLIB/MLGMSTP) +
                          TOFILE(&ADMPRD/MLGMSTP) MBROPT(*REPLACE)
/*----------------------------------------------------------------*/
             ENDPGM
```

D.1.3 Check out PARTL parts

This is an example of how you can check out parts that are in a part list. A user-defined option should be used to run this program.

As a suggestion,we used the option ″XP″, but you can use any other option if you want. The interface should be:

```
XP    call gg244358/chkoutprtl (&l &n &zt &zn &zp &zg)
```

```
/*-----------------------------------------------------------------*/
/* Process the information from a PARTLIST and check out all       */
/*  the parts from that list                                       */
/*                                                                 */
/* If the part is not a PARTL then stop the program and send a     */
/*  message to the requester of this program.                      */
/*                                                                 */
/* Parts that are already checked out are skipped and a message is */
/*  send to the requester                                          */
/*                                                                 */
/* The program cannot handle generic parts found in a PARTL part   */
/*  (for example P*, P?, *ALL)                                     */
/*                                                                 */
/* If the PARTL part being processed lists another PARTL part, the */
/*  program checks out the listed PARTL part and not the parts listed*/
/*  inside.                                                        */
/*                                                                 */
/*-----------------------------------------------------------------*/
             PGM        PARM(&LIB &FILE &PARTTYPE &PARTNAME &PROJECT +
                          &GROUP)
             DCL        VAR(&LIB) TYPE(*CHAR) LEN(10)
             DCL        VAR(&FILE) TYPE(*CHAR) LEN(10)
             DCL        VAR(&PARTTYPE) TYPE(*CHAR) LEN(10)
             DCL        VAR(&PARTNAME) TYPE(*CHAR) LEN(10)
             DCL        VAR(&PROJECT) TYPE(*CHAR) LEN(32)
             DCL        VAR(&GROUP) TYPE(*CHAR) LEN(32)
             DCL        VAR(&WS) TYPE(*CHAR) LEN(10)
             DCL        VAR(&TEXT) TYPE(*CHAR) LEN(50)
             DCLF       FILE(MYPARTL)
             MONMSG     MSGID(CPF0000)

/*  Retrieve the workstation name                                  */

             RTVJOBA    JOB(&WS)

/*  Check if the part that is supplied is a PARTL part             */

             IF         COND(&PARTTYPE *NE 'PARTL     ') THEN(DO)
             CHGVAR     VAR(&TEXT) VALUE('Part ' *CAT &PARTNAME +
                          *cat ' is not a Partlist')
             SNDBRKMSG  MSG(&TEXT) TOMSGQ(&WS)
             GOTO       CMDLBL(END)
             ENDDO

/*  Override to the file of the PART list                          */

             OVRDBF     FILE(MYPARTL) TOFILE(&LIB/&FILE)

/*  Read from the partlist file                                    */
```

```
AGAIN:
             RCVF
             MONMSG     MSGID(CPF0864) EXEC(GOTO CMDLBL(END))
             CHKOUTPART PRJ(&PROJECT) GRP(&GROUP) TYPE(&LITYPE) +
                          PART(&LIPART)
             MONMSG     MSGID(ADM1626) EXEC(DO)
             CHKOUTPART PRJ(&PROJECT) GRP(&GROUP) TYPE(&LITYPE) +
                          PART(&LIPART) PRMCODE(*NONE)
             MONMSG     MSGID(ADM1672 ADM1602 ADM1617) EXEC(DO)
             CHGVAR     VAR(&TEXT) VALUE('Cannot check out part ' +
                          *CAT &LIPART *CAT ' - see JOBLOG.')
             SNDBRKMSG  MSG(&TEXT) TOMSGQ(&WS)
             ENDDO
             ENDDO
             MONMSG     MSGID(ADM1672 ADM1602 ADM1617) EXEC(DO)
             CHGVAR     VAR(&TEXT) VALUE('Cannot check out part ' +
                          *CAT &LIPART *CAT ' - see JOBLOG.')
             SNDBRKMSG  MSG(&TEXT) TOMSGQ(&WS)
             ENDDO


/*  Return to the reading process                                    */

             GOTO       CMDLBL(AGAIN)

/*  End the program                                                  */

END:
             ENDPGM
```

D.2 Mailing List Application Description

We used the mailing list application, as described in *Application Development by Example*, SC41-9852, as the base application for the development environment chapter in this publication. The original source code of the application came from the QUSRTOOL library shipped with each release of OS/400 and optionally installed on a customer's system. We modified the code somewhat to make it more suitable for the ILE and RPG IV environment.

Although not an actual customer application, the mailing list application used is defined in such a way that most readers should be able to associate its functions to what any application must do. This theoretical application creates and maintains a mailing list file (called the master file), prints mailing labels from the file, and provides analysis of the file.

D.3 Functional Scenario

There are five main processes in the Mailing List Application; these are:

1. Inquire into Mailing List Master File
2. Maintain Mailing List Master File
3. Submit mailing by account number
4. Submit special analysis report
5. Query Mailing List file

To run the Mailing List Application, call program MLGMNUC and the Mailing List Menu panel is shown.

D.3.1 Inquire into the Mailing List Master File

```
                       Mailing List Menu
                                                      System:  RCHASM01
Select one of the following
     1. Inquire into Mailing List Master
     2. Maintain Mailing List Master
     3. Submit mailing by account number
     4. Submit special analysis report
     5. Query Mailing List file




Selection
===>1
F3=Exit
```

Figure 103. Mailing List Menu

Option 1 on the Mailing List Menu shows you the Mailing List Inquiry panel.

```
                       Mailing List Inquiry
Search field . . . . . .: ____________
    or
Account number . . . . .: 15902






F3=Exit
```

Figure 104. Mailing List Inquiry Panel

Now enter an existing Account number, for example, 15902, and press Enter; the Mailing List Inquiry panel is now completed. You can also use a name search to find the customer number using the same program as described with Figure 109 on page 198 and Figure 110 on page 198

```
                       Mailing List Inquiry
Account number . . . . .:   15902
Account type . . . . . .:   2
Name . . . . . . . . . .:   Joseph Jones
Name search  . . . . . .:   JONES
Address  . . . . . . . .:   3008 Brook St
City . . . . . . . . . .:   Little Rock
State  . . . . . . . . .:   AR
Zip code . . . . . . . .:   44877





F3=Exit    F16=Print label
```

Figure 105. Account Number Inquiry

You can use F=16 if you want to print one label of this customer.

D.3.2 Maintain Mailing List Master File

Option 2 on the Mailing List Menu shows you the Maintain Mailing List Master panel. From this panel you can:

- Display, change, add, or delete records in mailing list master file
- Display the greater or equal (GE) value
- Do a name search

For options 1 to 5, you have to enter an account number; for option 6, enter a name in the Search Field.

```
                        Maintain Mailing List Master
Action . . . . . . . . .:                         1=Display  2=Change
                                                  3=Add      4=Delete
                                                  5=Display GE value
                                                  6=Name search
Account number . . . . .:                         Numeric 5.0
Search field . . . . . .:                         Char


For Options 1-5, enter an Account number
For Option 6, enter a Search field

F3=Exit
```

Figure 106. Maintain Mailing List Master Panel

Following is an example panel to change the mailing list master file.

```
                        Maintain Mailing List Master       ACTION -  Change
Account number . . . . .:   15902                 Name
Account type . . . . . .:   2                     1=Bus 2=Gov 3=Org 4=Sch 5=Pvt
                                                  9=Oth
Name . . . . . . . . . .:   Joseph Jones                    Char
Search name  . . . . . .:   JONES                           Char
Address  . . . . . . . .:   3008 Brook St                   Char
City . . . . . . . . . .:   Little Rock                     Char
State  . . . . . . . . .:   AR                    Valid state abbreviations
Zip code . . . . . . . .:   44877                 Numeric 5.0




F3=Exit
```

Figure 107. Change Mailing List Master Panel

Action **5** on the Maintain Mailing List Master panel allows you to search through the file (one record at a time) either from the account number entered or from the beginning of the file if no account number is entered. If no account number is entered, the search begins by displaying the first record in the file. You may page down to see the next record. If a value is entered, such as account number 50000, the search starts with account number 50000. If the specified account number does not exist, the first account number after 50000 is displayed. You can page up or page down. If you see a record you want to change or delete, function keys allow the display to be switched to change or delete mode.

```
                       Maintain Mailing List Master       ACTION -  GE value
Account number . . . . .:   15902              Name
Account type . . . . . .:   2                  1=Bus 2=Gov 3=Org 4=Sch 5=Pvt
                                               9=Oth
Name . . . . . . . . . .:   Joseph Jones                 Char
Search name  . . . . . .:   JONES                        Char
Address  . . . . . . . .:   3008 Brook St                Char
City . . . . . . . . . .:   Little Rock                  Char
State  . . . . . . . . .:   AR                 Valid state abbreviations
Zip code . . . . . . . .:   44877              Numeric 5.0




F3=Exit   F6=Change    F11=Delete       Rollup         Rolldown
```

Figure 108. Display GE Value

Action 6 on the Maintain Mailing List Master panel allows a method of determining the account number when only the account name is known. It allows you a search by entering one or more characters in the search field. For example, if you enter the name SMITH, all accounts with a search field of SMITH, SMITHERMAN, SMITHE, and so on are shown. This is based on the search field entered into every record.

```
                       Maintain Mailing List Master
Action . . . . . . . . .: 6             1=Display  2=Change
                                              3=Add      4=Delete
                                              5=Display GE value
                                              6=Name search
Account number . . . . .:                     Numeric 5.0
Search field . . . . . .: JONES            Char



For Options 1-5, enter an Account number
For Option 6, enter a Search field

F3=Exit
```

Figure 109. Name Search

```
                  Mailing List Name Search              Search JONES
Type options, press Enter
   1=Display details   2=Return with Acct Number
Opt Search      Name           St  City           Address       Typ  Account
2   JONES       Joseph Jones   AR  Little Rock    3008 Brook St  2    15902
    JONES       Philip Jones   CA  San Diego      365 Parkway    9    28903
    JONES       Samuel Jones   MN  Minneapolis    220 4 Ave NW   1    10057
    JONESA      Maria Jonesa   PA  Philadelphia   559 9th Ave S  5    38724





F3=Exit
```

Figure 110. Result JONES Search

You can respond by requesting a return with a specific account number. After the return, the record is displayed in display mode. You can press F6 to go into change mode on the record.

```
                         Maintain Mailing List Master       ACTION -  Display
Account number . . . . .:   15902                Name
Account type . . . . . .:   2                    1=Bus 2=Gov 3=Org 4=Sch 5=Pvt
                                                 9=Oth
Name . . . . . . . . . .:   Joseph Jones                   Char
Search name  . . . . . .:   JONES                          Char
Address  . . . . . . . .:   3008 Brook St                  Char
City . . . . . . . . . .:   Little Rock                    Char
State  . . . . . . . . .:   AR                   Valid state abbreviations
Zip code . . . . . . . .:   44877                Numeric 5.0




F3=Exit   F6=Change    F11=Delete      Rollup        Rolldown
```

Figure 111. Return with Account Number

D.3.3 Submit Mailing by Account Number

Option 3 on the Mailing List menu produces a listing of all of the records in the Mailing List Master file. See Figure 112 for an example.

```
 12/14/93  14:07:33                      Customer listing                          Page 1
Number     Name                 City                 State     Zip     Type   Name search
15902      Joseph Jones         Little Rock           AR       44877    2     JONES
28903      Philip Jones         San Diego             CA       66903    9     JONES
10057      Samuel Jones         Minneapolis           MN       55454    1     JONES
14477      Charles Hanley       Rochester             MN       55920    4     HANLEY
18890      Carol Larson         Rochester             MN       55920    5     LARSON
11458      James Grover         Trenton               NJ       08690    1     GROVER
26640      Daniel Benson        Syracuse              NY       13212    1     BENSON
38724      Maria Jonesa         Philadelphia          PA       22809    5     JONESA
24882      Kathryn Donty        Dallas                TX       75248    1     DONTY
       Count of records-        9
```

Figure 112. Example Report

D.3.4 Submit Special Analysis Report

Option 4 on the Mailing List menu produces a listing of all the records in the Mailing List Master file with a zip code of 55920. See Figure 113 for an example.

```
 12/14/93  14:29:41                      Customer listing                          Page 1
Number     Name                 City                 State     Zip     Type   Name search
14477      Charles Hanley       Rochester             MN       55920    4     HANLEY
18890      Carol Larson         Rochester             MN       55920    5     LARSON
       Count of records-        2
```

Figure 113. Example Report Zip Code 55920

D.3.5 Query Mailing List File

Option 5 on the Mailing List menu calls the STRQRY command and lets you create your own query against the Mailing List Master file.

D.4 Parts Structure

The following objects (parts) are part of the Mailing List Application:

- MLGREFP (PF) mailing list field reference file
- MLGMSTP (PF) mailing master physical file
- MLGMSTL (LF) mailing list label printing logical
- MLGMSTL2 (LF) mailing list by state, city, name
- MLGMSTL3 (LF) general purpose LF for querying MLGMSTP
- MLGNAML (LF) mailing list logical file by name, state, city
- MLGINQD (DSPF) mailing list inquiry display
- MLGMTND (DSPF) mailing maintenance display file
- MLGMNUD (DSPF) mailing list menu display file
- MLGNAMD (DSPF) mailing list name search display file
- MLGRPTC (CLLE) print one liner with MLGMSTL2 LF
- MLGRPTC2 (CLLE) general purpose query of MLGMSTP
- MLGMTNC (CLLE) mailing list maintenance
- MLGMNUC (CLLE) mailing list menu program
- MLGLBLR (RPGLE) mailing list label printing
- MLGRPTR (RPGLE) mailing list one line report per name
- MLGINQR (RPGLE) mailing list inquiry
- MLGMTNR (RPGLE) mailing list master maintenance
- MLGNAMR (RPGLE) mailing list name search.

We added one more part to this base application:

MLGMSGF is the mailing list message file for the MSGCON and ERRMSGID keywords in the display files. In this way we could easily create a National Language version of the application.

Note: For ease of use, we only translated the message file and some RPG source files. We did not translate the constants in the Field reference file.

For a detailed explanation of the Mailing List Application, please refer to the publication *Application Development By Examples*, SC41-9852.

Index

A

activation group
  call stack 92
  CALLER 85, 90
  control boundary 94
  default 89, 91
  DFTACTGRP(*YES) 87
  DSPJOB 92
  ending an application 113
  exception handling 91
  NAMED 85
  NEW 85
  non-default 91
  recommendations 91
  run unit 114
  scoping 91
  system-named 90
  termination 118
  user-named 90
API
  CEE4ABN 116, 118
  CEEDATE 47
  CEEDAYS 47
  CEEHDLR 169
  CEEMRCR 166
  CEETREC 115, 118
  ILE condition handlers 163
  QlgConvertCase 74
  QLGCNVCS 74
Application Development Manager/400 123
  binding directory 126, 147
  build report 130
  export 140
  import 140
  part list 141
  part relationships 126
  service program 126
  trigger relations 127

B

bind by copy 83
bind by reference 83
binding
  program 98
  service program 99
binding directory 86, 96, 125
  with ADM/400 147
binding language 84, 144
binding source 125
build option 125

C

C specification 23
  new layout 24
call bound procedure 79
call stack 92
CALLPRC 77, 79
case conversion
  QlgConvertCase 74
  QLGCNVCS 74
CCSID 59, 71
CEEDATE 47
CEEDAYS 47
circular references 103
COBOL 168
CODE/400 5
commitment control 116
compatibility mode 78, 87, 113
  RCLRSC 121
compile options 152
constants
  externalizing 72
control boundary 94, 115, 164
  example 95
  hard control boundary 95
  soft control boundary 95
conversion
  /COPY 61
  arrays 62
  correct manually 62
  example 64
  externally described data structure 63
  QARNCVTLG 60
  QRNCVTLG 60
  QRPGLESRC 57
  scanning tool 64
COPY 66
copyright 148
CRTBNDRPG 57
CRTRPGMOD 57
CVTRPGSRC 59, 128

D

D specification 12
  ALT 14
  ASCEND 14
  BASED 14
  CONST 14
  CTDATA 14
  DATFMT 14
  DESCEND 14
  DIM 14
  DTAARA 14
  examples 18

© Copyright IBM Corp. 1995

D specification *(continued)*
EXPORT 14
EXTFLD 14
EXTFMT 14
EXTNAME 14
FROMFILE 14
IMPORT 14
INZ 14
LIKE 14
NOOPT 14
OCCURS 15
OVERLAY 15
PACKEVEN 15
PERRCD 15
PREFIX 15
PROCPTR 15
TIMFMT 15
TOFILE 15
date
calculations 43
example 42
date and time formats 40
date data types 40
date and time APIs 47
external formats 41
initializing 41
move operations 46
DATEDIT 41
DBCS graphic data type 75

E

E specification 27
error handling 159
example
ADM/400 127
mailing list application 127
exception handlers
direct monitors 167
HLL-specific handlers 167
ILE condition handlers 167
priority 167
types of 167
Exception Handling 161
architecture 162
comparing OPM 174
example 170
File Exceptions 161
handle cursor 166
HLL-specific handlers 163
ILE condition handlers 163
ILE direct monitors 163
resume cursor 166
export 16, 140, 159
circular references 103
data structure 51
example 51, 100
procedure 99
unresolved references 103
export *(continued)*
what is 97
external described files 10
externally described files 20

F

F specification 9
COMMIT 10
DATFMT 10
DEVID 10
EXTIND 10
FORMLEN 10
FORMOFL 10
IGNORE 10
INCLUDE 10
INFDS 10
INFSR 11
KEYLOC 11
MAXDEV 11
OFLIND 11
PASS 11
PGMNAME 11
PLIST 11
PREFIX 11
PRTCTL 11
RAFDATA 11
RECNO 11
RENAME 11
SAVEDS 11
SAVEIND 11
SFILE 11
SLN 11
TIMFMT 11
USROPN 11
file information data structure (INFDS) 161
file input/output feedback area 161
file open feedback area 161
file open scope 107

H

H Specification 8, 9
ALTSEQ 9
CURSYM 9
DATEDIT 9
DATFMT 9
DEBUG 9
DECEDIT 9
DFTNAME 9
FORMSALIGN 9
FTRANS 9
TIMFMT 9

I

I specification 20
field layout 21
record layout 21

ILE 84, 161
  activation groups 84
  Application Development Manager/400 123
  bind by copy 83
  bind by reference 83
  binding 96
  binding directory 86
  binding language 84
  call stack 92
  compile and bind 87
  concepts 83
  control boundary 94
  copyright 148
  creation commands 68
  development process 123
  ending an application 113
  Exception Handling 161
  modules 83
  naming conventions 125
  performance 151
  procedures 83
  program activation 85
  program entry procedure 86
  programs 83
  run unit 114
  service program 84, 87
  static binding 83
  transparency 110
ILE C/400 4
ILE CL 53
  CALLPRC 77
  CRTBNDCL 77
  CRTCLMOD 77
  parameter passing 80
  source type CLLE 81
ILE COBOL/400 3
ILE program activation 85
ILE RPG/400 2
import 16, 140, 159
  circular references 103
  data structure 51
  example 51, 100
  procedure 99
  unresolved references 103
  what is 97
INFDS (file information data structure) 161
information status subroutine 162
INFSR 162, 168
initialization 99

J

job message queues 163
job trace 155

L

L specification 27
length
  name 37
limits
  changes 37
  character field size 37
  constant size 37
  data structure name 37
  data structure size 37
  field/array name 37
  file name 37
  named constant 38
  number of array elements 37
  number of decimal places 37
  number of files 37
  number of subroutines 38
  record format name 37
  size of program 38
local data area 16

M

module 83, 125
  binding directory 96
  create 68
MONMSG 79, 168

N

national language support
  case conversion 74
  date fields 73
  sort sequence 73
  usage of characters 71

O

O specification 25, 26
observability 153
operation code 34
  CALLB 33
  EVAL 35
  example 32
  EXTRCT 32
  TEST 32
Operation Codes 29, 30, 31
  ADDDUR 30
  Process Date and Time 30
  Renamed 29
  SUBDUR 31
override example 108
override rules 108
override scope 107
OVRDBF 107

P

percolation 166, 169
performance
  activation groups 156

performance *(continued)*
  benefits 159
  compile options 152
  compile time 151
  considerations 156
  exception handling 175
  runtime 155
  working memory requirements 153
  working memory size 155
pointer
  example 53
procedure 83
program 125
  binding directory 96
  compress observability 153
  create 68
  create bound 69
  ending 112
  initialization 99
  object size 153
  unresolved references 103
  updating 105
program described files 10, 21
program entry procedure 93
  for ILE C 86
  for ILE CL 86
  for ILE RPG/400 86
PRTCMDUSG 79
PSSR 168

Q

QARNCVTLG 60
QRNCVTLG 60

R

RCLACTGRP 116, 117
RCLRSC 78, 113, 121
RPG IV 2
  %ADDR 38
  %ELEM 39
  %SIZE 39
  %SUBST 39
  %TRIM 39
  %TRIML 39
  %TRIMR 39
  ALTSEQ 73
  built-in functions 38
  case conversion 74
  compatibility mode 87
  compile time 151
  DBCS 75
  example 53
  national language support 71
  pointers 53
  sort sequence 73
  usage of characters 71
RPG IV Specifications 7
RTVCLSRC 78
run unit 114

S

sample address monitor 155
scoping
  activation group 91
  file opens 106
  overrides 106
  RCLRSC 113
  resource 106
  transparency 110
service program 84, 125, 132, 138, 159
  binding directory 96
  create 87
  initialization 99
  recommendations 105
  signature 104
  unresolved references 103
  updating 105
signal 168
signature 138
  current 104
  mismatch 104
  previous 104
SNDPGMMSG 79
software
  copyright 148
sort sequence 73
Specification Sheets 7
static bind
  bind by copy 83
  bind by reference 83
static call
  CALLB 33
symbolic names 36
  name length 37

T

termination
  example 118
TFRCTL 78
time data types 40
  calculations 43
  date and time APIS 47
  example 42
  external formats 41
  initializing 41
  move operations 46
timestamp data type 40, 50
timing and paging statistics 155
TPST 155
transparency 110

U

unresolved references 103
UPDPGM 105, 126
UPDSRVPGM 105, 126
upper/lowercase 36

V

VRPG Client/2 5