

Application System/400



RPG/400 User's Guide

Application System/400



RPG/400 User's Guide

Note!

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

First Edition (June 1994)

This edition applies to the licensed program IBM* ILE* RPG/400* (Program 5763-RG1), Version 3 Release 0 Modification 5, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, you can address your comments to:

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See "Communicating Your Comments to IBM" for a description of the methods. This page immediately precedes the Readers' Comment Form at the back of this publication.

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 1994. 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.

Contents

Notices	xi
Programming Interface Information	xi
Trademarks and Service Marks	xi
About This Manual	xiii
Who Should Use This Manual	xiii
How to Interpret Syntax Diagrams	xiv
Chapter 1. An Introduction to RPG/400 and the AS/400 System	1
The OS/400 System	1
The AS/400 Control Language	1
Commonly Used Control Language Commands	2
System/38 Environment on the AS/400 System	3
AS/400 Utilities and Languages	3
The Source Entry Utility	3
The Screen Design Aid	4
The Structured Query Language	4
Restrictions	4
Designing Your RPG/400 Program	5
Designing the Output	5
Designing the Processing	6
Designing the Input	6
Structured Programming in the RPG/400 Programming Language	6
Sequential Operation	6
Conditional Branching	6
If Else Structure	6
SELEC Structure	8
Other Conditional Branching Structures	9
Repeating an Operation	9
Do Operation	9
Do While Operation	11
Do Until Operation	13
Summary of Structured Programming Operation Codes	14
Designing Applications	15
Single Program Design	15
Modular Program Design	16
Examples of Application Design	17
Chapter 2. Entering RPG/400 Specifications	21
The RPG/400 Specifications	21
The Control Specification	21
File Description Specifications	22
Extension Specifications	22
Line Counter Specifications	22
Input Specifications	22
Calculation Specifications	22
Output Specifications	23
Entering Your Program	23
Chapter 3. Compiling an RPG/400 Program	25

Create RPG400 Program (CRTRPGPGM) Command	26
Using the CRTRPGPGM Command	27
Elements of the CRTRPGPGM Command Lines	28
Entering Elements from the CRTRPGPGM Command Display	28
Entering Only Certain Parameters	28
Entering Only the Parameter Values	29
CRTRPGPGM Command	29
Compiling under the System/38 Environment	45
Chapter 4. Error Messages, Testing, and Debugging	47
Using, Displaying, and Printing Messages	47
Using Messages	47
Systems Application Architecture Flagging Messages	49
Displaying and Printing Messages	49
How to Run an RPG/400 Program	49
Save-While-Active Support	50
Using a Test Library	51
Using Breakpoints	54
Example of Using Breakpoints	54
Considerations for Using Breakpoints	57
Using a Trace	58
Example of Using a Trace	59
Considerations for Using a Trace	60
Using the DEBUG Operation Codes	60
Using the RPG/400 Formatted Dump	60
Exception/Error Handling	70
Chapter 5. General File Considerations	75
Device Independence/Device Dependence	75
Spooling	77
Output Spool	77
Externally Described and Program-Described Files	78
Level Checking	80
File Locking by an RPG/400 Program	81
Record Locking by an RPG/400 Program	81
Unblocking Input Records and Blocking Output Records	82
Sharing an Open Data Path	83
Using the Control Language Command RCLRSC	84
Specifications for Externally Described Files	84
File Description Specifications	85
Renaming Record-Format Names	85
Ignoring Record Formats	86
Floating-Point Fields	86
Overriding or Adding RPG/400 Functions to an External Description	87
Output Specifications	88
Program-Described Files	90
Printer Files	90
Page Overflow	91
Overflow Indicators	92
Fetch-Overflow Logic	94
PRTCTL (Printer Control) Option	96
Sequential File	99
Special File	101

Chapter 6. Commitment Control	107
Using Commitment Control	107
Starting and Ending Commitment Control	107
Specifying Files for Commitment Control	108
Commitment Control Operations	108
Commitment Control Locks	109
Commitment Control in the Program Cycle	109
Example of Using Commitment Control	110
Chapter 7. Using DISK Files	113
Externally Described Disk Files	113
Record Format Specifications	113
Access Path	114
Valid Keys for a Record or File	117
Valid Search Arguments	117
Referring to a Partial Key	118
Processing Methods for Externally Described DISK Files	118
Program-Described Disk Files	119
Indexed File	119
Valid Search Arguments	119
Sequential Files	122
Record Address File	122
Limits Records	122
Relative Record Numbers	123
Externally Described File as Program Described	123
Methods for Processing Disk Files	123
Relative-Record-Number Processing	123
Consecutive Processing	124
Sequential-by-Key Processing	124
Sequential-within-Limits Processing	132
Keyed Processing Examples	132
Valid File Operations	140
Chapter 8. Using WORKSTN Files	143
Intersystem Communications Function	143
Externally Described WORKSTN Files	143
Processing an Externally Described WORKSTN File	144
Function Key Indicators on Display Device Files	146
Command Keys on Display Device Files	147
Processing WORKSTN Files	147
EXFMT Operation	147
READ Operation	147
WRITE Operation	147
WORKSTN file	148
Subfiles	150
Use of Subfiles	152
Program-Described WORKSTN File	154
Program-Described WORKSTN File with a Format Name	154
Output Specifications	154
Input Specifications	155
Calculation Specifications	155
Additional Considerations	156
Program-Described WORKSTN File without a Format Name	156
Input File	156

Output File	156
Combined File	156
Multiple-Device Files	157
WORKSTN File Examples	158
Sample Program 1–Inquiry	159
Sample Program 2–Data Entry with Master Update	166
Sample Program 3–Maintenance	174
Sample Program 4–WORKSTN Subfile Processing	187
Sample Program 5–Inquiry by Zip Code and Search on Name	196
Sample Program 6–Program-Described WORKSTN File with a FORMAT Name on Output Specifications	206
Sample Program 7–Variable Start Line	208
Sample Program 8–Read Operation with Time-Out	211
Chapter 9. Data Field Formats and Data Structures	215
Format of Fields in Files	215
Packed-Decimal Format	215
Zoned-Decimal Format	216
Binary Format	217
Program-Described File	217
Externally Described File	218
Signs	219
External Formats	220
Internal Format	220
Data Structures	220
Format of Data Structure Subfields in Storage	221
Data Structure Statement Specifications	221
Rules for Specifying Data Structure Statements	222
Multiple Occurrence Data Structure	223
Special Data Structures	223
Data Area Data Structure	223
File Information Data Structure	224
Program-Status Data Structure	224
Data Structure-Subfield Specifications	224
Rules for Subfield Specifications	226
Data Structure Examples	226
Chapter 10. Named Constants, Initialization, and SAA Data Types	237
Named Constants	237
Rules for Named Constants	237
Initialization	240
Initialization Subroutine (*INZSR)	240
CLEAR and RESET Operation Codes	240
Data Structure Initialization	241
Special Considerations for Initializing Data Structures	241
Rules for Initializing Subfields	242
Initialization and the Program Cycle	243
Initialization Examples	243
SAA Data Types	247
Variable-Length Fields	247
Date, Time, and Timestamp Fields	250
DBCS-Graphic Data Type Support	251
Null Value Support	252
Error Handling for SAA Data Types	253

Chapter 11. Communicating with Objects in the System	255
Calling Other Programs	255
CALL (Call a Program)	258
PLIST (Identify a Parameter List) and PARM (Identify Parameters)	259
Rules for Specifying PLIST	259
Rules for Specifying PARM	260
OS/400 Graphics Support	260
FREE (Deactivate a Program)	261
Calling Special Subroutines	261
Message-Retrieving Subroutine (SUBR23R3)	261
SAA Common Programming Interface Support	263
Moving Bracketed Double-byte Data and Deleting Control Characters (SUBR40R3)	263
Moving Bracketed Double-byte Data and Adding Control Characters (SUBR41R3)	264
Returning from a Called Program	265
A Normal End	266
An Abnormal End	266
Return without an End	267
Data Areas	267
Program Initialization Parameters (PIP) Data Area	269
Chapter 12. Auto Report Feature	271
Group Printing	271
Specifications	271
Examples	271
/COPY Statement Specifications	275
Changing Copied Specifications	276
Changing File Description Specifications	277
Changing Input-Field Specifications	277
Report Format	280
Spacing and Skipping	280
Placement of Headings and Fields	281
Page Headings	281
Reformatting *AUTO Page Headings	282
Body of the Report	283
Overflow of the D/T-*AUTO Print Lines	283
Generated Specifications	284
Generated Calculations	285
Generated Output Specifications	285
Programming Aids	291
Using CRTRPTPGM to Compile an Auto Report Program	294
Using the CRTRPTPGM Command	295
CRTRPTPGM Command	295
Examples of Using Automatic Report	299
EXAMPLE 1 - Sales Report	299
EXAMPLE 2 - Sales Report with Three Levels of Totals	304
EXAMPLE 3 - Sales Report with Group Indication	307
EXAMPLE 4 - Sales Report with Cross-Column Totals	310
EXAMPLE 5 - Sales Report Using Copied Specifications	314
EXAMPLE 6 - Override Copied Input Specifications	317
Chapter 13. RPG/400 Sample Programs	321
Checklist of Program Examples	321

Database Design	324
Employee Master File	324
Project Master File	324
Reason-Code Master File	325
Transaction History Files	325
Data Area Control File	326
Master File Maintenance	326
Data Area Control File Maintenance	326
Time-File Entry	327
Weekly Time-File Update	328
Monthly Time-Entry File Reporting and Update	330
Database Field Definition	332
Database Reference Master File - REF MST	333
Data Area Control File - CTLFIL	336
Employee Master File - EMP MST	337
Project Master File - PRJ MST	338
Reason-Code Master File - RSN MST	339
Weekly Transaction Entry File - TR WEEK	340
Monthly Transaction Entry File - TR MNTH	341
Time Reporting Menu Design	343
Master File Maintenance	346
Master File Maintenance Display - PRG01FM	347
SELECT Format - Maintenance Selection	347
Employee Master Selection - EMPSEL Format	348
Employee Master Maintenance - EMPMNT Format	349
Project Master Selection - PRJSEL Format	350
Project Master Maintenance - PRJMNT Format	351
Reason Code Master Selection - RSNSEL Format	352
Reason Code Master Maintenance - RSNMNT Format	353
Master File Maintenance Data Descriptions - PRG01FM	354
Master File Maintenance RPG/400 program - PRG01	364
Control File Maintenance	382
Control File Maintenance - PRG02FM	383
Control File Maintenance Data Descriptions - PRG02FM	384
Control File Maintenance RPG/400 Program - PRG02	386
Time File Transaction Entry	394
Time Reporting Transaction Entry - PRG03FM	395
Employee Selection Display	395
Time Reporting Transaction Entry Data Descriptions - PRG03FM	397
Time Reporting Transaction Entry RPG/400 Program - PRG03	401
Weekly Time File Update	418
Time File Entry Edit RPG/400 Program - PRG05	421
Weekly Employee Transaction Report Layout - PRG09	427
Master File Update and Weekly Transaction Report - PRG09	428
Monthly Processing	444
Monthly Time File Update and Reporting	444
Time Reporting Employee Summary Report Layout - PRG06RP	448
Employee Summary Report Data Descriptions - PRG06RP	449
Employee Summary Report RPG/400 Program - PRG06	453
Time Reporting Project Summary Report Layout - PRG07RP	463
Project Summary Report Data Descriptions - PRG07RP	464
Project Summary Report RPG/400 Program - PRG07	468
Time Reporting Reason Code Summary Report Layout - PRG08RP	475
Reason Code Summary Report Data Descriptions - PRG08RP	476

Reason Code Summary Report RPG/400 Program - PRG08	479
Master File Monthly Update and Clear RPG/400 Program - PRG04	487
Year End Processing	491
Appendix A. RPG Compiler and Auto Report Program Service	
Information	493
Compiler Overview	493
Compiler Phases	494
Major Compiler Data Areas	496
Compiler Error Message Organization	496
Run-Time Subroutines	497
Compiler Debugging Options	498
*SOURCE Value for the OPTION Parameter	498
*XREF Value for the OPTION Parameter	498
*DUMP Value for the OPTION Parameter	498
*LIST Value for the GENOPT Parameter	498
*ATR Value for the GENOPT Parameter	498
*XREF Value for the GENOPT Parameter	498
*DUMP Value for the GENOPT Parameter	498
*PATCH Value for the GENOPT Parameter	498
*OPTIMIZE Value for the GENOPT Parameter	499
ITDUMP Parameter	499
SNPDUMP Parameter	499
CODELIST Parameter	499
PHSTRC Parameter	499
Examples of Using Compiler Debugging Options	500
IRP Layout	515
Auto Report Program	518
Appendix B. RPG/400 and AS/400 RPG II System/36-Compatible	
Functions	521
Language Enhancements	521
Appendix C. Data Communication	527
Exception and Error Handling with ICF Files	527
Communications Error Recovery	527
Appendix D. Distributed Data Management (DDM) Files	529
Appendix E. System/38 Environment Option of the RPG Compiler	531
Differences between System/38 RPG III and the System/38 Environment	
Option of the RPG Compiler	531
Differences between the System/38 Environment Option of the RPG Compiler	
and RPG/400 Compiler	531
File Types Supported by Each Compiler	535
Appendix F. Examples of Using Arrays	537
Appendix G. Glossary of Abbreviations	547
Bibliography	549
Index	551

Notices

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 product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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, 208 Harbor Drive, Stamford, Connecticut, USA 06904-2501.

Changes or addition to the text are indicated by a vertical line (|) to the left of the change or addition.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This RPG/400 User's Guide is intended to help you create RPG/400 programs. This RPG/400 User's Guide documents *general-use programming interfaces and associated guidance information* provided by the RPG/400 compiler.

General-use programming interfaces allow the customer to write programs that request or receive services of the RPG/400 compiler.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks of the IBM Corporation in the United States or other countries:

Application System/400	AS/400
IBM	ILE
Operating System/2	Operating System/400
OS/2	OS/400
RPG/400	SAA
Systems Application Architecture	SQL/400
400	

About This Manual

This manual is a guide for the RPG/400* programming language on the AS/400 system using the Operating System/400* (OS/400*) system. The RPG/400 compiler is a Systems Application Architecture* (SAA*) compiler that adheres to SAA conventions.

The topics covered in this manual include:

- Designing RPG/400 programs
- Coding RPG/400 programs
- Entering and compiling RPG/400 programs
- Testing and debugging RPG/400 programs
- Studying coded RPG/400 examples and sample programs.

This manual may refer to products that are announced but are not yet available.

You may need to refer to other IBM* manuals for more specific information about a particular topic. The *Publications Guide, GC41-9678*, provides information on all of the manuals in the AS/400* library. For a list of related publications, see the "Bibliography" on page 549.

Who Should Use This Manual

This manual is intended for people who have a basic understanding of data processing concepts and of the RPG/400 programming language. It is also designed to guide the programmer in the use of RPG/400 programs and compilers on the AS/400 system. RPG/400 specifications and operations are frequently mentioned. For a detailed description of RPG/400 specifications and operation codes, see the *RPG/400 Reference, SC09-1817*.

Before you use this manual, you should be familiar with certain information:

- You should know how to use data management support to work with files, display stations, printers, tapes, and diskettes, as well as spooling support. This information is contained in the *Data Management Guide*.
- You should be familiar with your display station (also known as a work station) and its controls. Some elements of its display and certain keys on the keyboard are standard regardless of the software system currently running at the display station or the hardware system the display station is connected to. Some of these keys are:
 - Cursor movement keys
 - Command keys
 - Field exit keys
 - Insert and delete keys
 - The Error Reset key.

This information is contained in the *New User's Guide, SC41-8211*.

- You should know how to operate your display station when it is connected to the IBM AS/400 system and running AS/400 software. This means knowing about the OS/400 system and the Control Language (CL) to perform the tasks of:

- Sign on and sign off of the AS/400 system
- Interact with displays
- Use Help
- Enter control commands
- Call utilities
- Respond to messages.

To find out more about control language, refer to these IBM AS/400 publications:

- *CL Programmer's Guide*
 - *Control Language Reference*
- You should be familiar with the RPG/400 program cycle, how indicators affect the program cycle, and how to code entries on the RPG/400 specification sheets.

The sample application programs contained in this manual are scaled in such a way that you can use the *RPG Debugging Template*, GX21-9129 to check the coding in the programs.

These general items about the RPG/400 programming language are taught in an RPG/400 coding class. Detailed information on the RPG/400 programming language can be found in the *RPG/400 Reference*.

How to Interpret Syntax Diagrams

The syntax diagrams in this book use the following conventions:

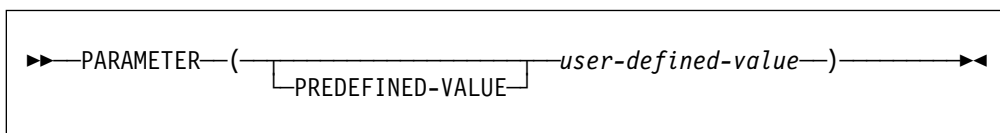


Figure 1. Structure of a Syntax Diagram

Read the syntax diagram from left to right, from top to bottom, following the path of the line.

The \blacktriangleright symbol indicates the beginning of the syntax diagram.

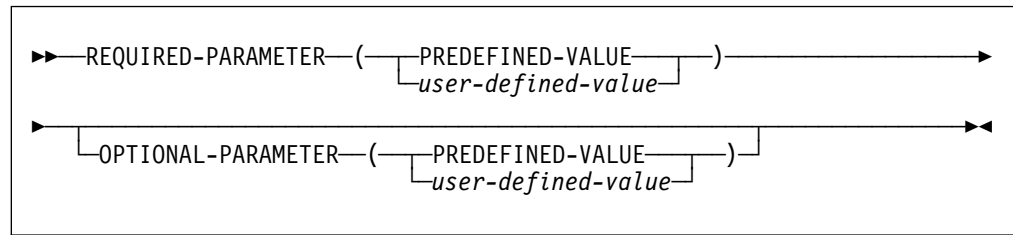
The \blacktriangleleft symbol indicates the end of the syntax diagram.

The \longrightarrow symbol indicates that the statement syntax is continued on the next line.

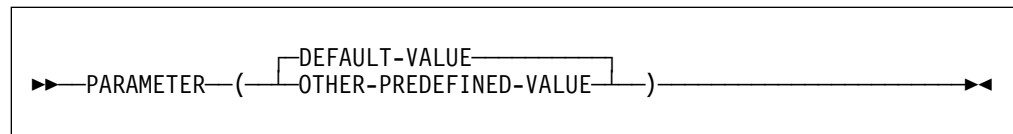
The \blacktriangleright symbol indicates that a statement is continued from the previous line.

The $\text{---}(\text{---})\text{---}$ symbol indicates that the parameter or value must be entered in parentheses.

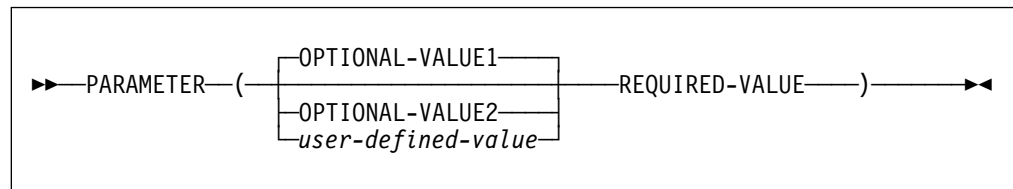
Required parameters appear on the base line and must be entered. **Optional parameters** appear below the base line and do not have to be entered. In the following sample, you must enter REQUIRED-PARAMETER and a value for it, but you do not have to enter OPTIONAL-PARAMETER or a value for it.



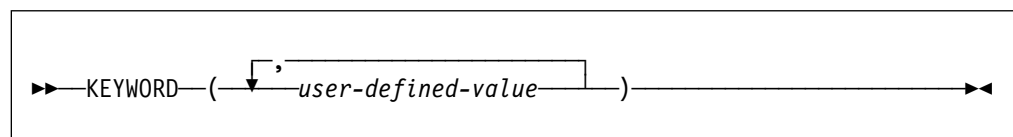
Default values appear above the base line and do not have to be entered. They are used when you do not specify a parameter. In the following sample, you can enter **DEFAULT-VALUE**, **OTHER-PREDEFINED-VALUE**, or nothing. If you enter nothing, **DEFAULT-VALUE** is assumed.



Optional values are indicated by a blank line. The blank line indicates that a value from the first group (**OPTIONAL-VALUE1**, **OPTIONAL-VALUE2**, *user-defined-value*) does not have to be entered. For example, based on the syntax below, you could enter: **KEYWORD**(**REQUIRED-VALUE**).



Repeated values can be specified for some parameters. The **,** in the following sample indicates that each *user-defined-value* must be separated by a comma.



Reading Syntax

Chapter 1. An Introduction to RPG/400 and the AS/400 System

The RPG/400 programming language is designed to make it easier for you to create business software applications.

RPG is a language under evolution. A slightly different version of RPG is available on each machine that supports it. The AS/400 system is the most recent of these computing systems. You should know that, as well as offering a new enhanced version of RPG, the AS/400 system also supports the previous versions of RPG available on System/38 and System/36. For more information, see Appendix B, "RPG/400 and AS/400 RPG II System/36-Compatible Functions," and Appendix E, "System/38 Environment Option of the RPG Compiler."

This chapter provides an overview of the following subjects:

- The OS/400 system and Control Language (CL)
- RPG/400 functions on the AS/400 system
- The System/38 environment on the AS/400 system
- Available languages and utilities
- The RPG/400 programming cycle
- RPG/400 program design
- Structured programming in RPG/400 programs
- Application design.

The OS/400 System

The operating system that controls all of your interactions with the AS/400 system is called the Operating System/400 (OS/400) system. From your work station, the OS/400 system allows you to:

- Sign on and sign off
- Interact with the displays
- Use the online help information
- Enter control commands and procedures
- Respond to messages
- Manage files
- Run utilities and programs.

Refer to the *Publications Guide* for a complete list of publications that discuss the OS/400 system.

The AS/400 Control Language

You can manipulate the OS/400 system with the CL. You interact with the system by entering or selecting CL commands. The AS/400 system often displays a series of CL commands or command parameters appropriate to the situation on the screen. You then select the desired command or parameters.

Commonly Used Control Language Commands

The following table lists some of the most commonly used CL commands, their function, and the reasons you might want to use them.

<i>Table 1. RPG/400 Functions and Associated CL Commands</i>		
RPG/400 Function	Associated Control Language Commands and their Uses	
Calling	CALL program-name CALL QCL	Run an RPG/400 program Access the System/38 environment
Commitment Control	CRTJRN CRTJRNCV ENDCMTCTL JRPNF STRCMTCTL	Prepare to use commitment control. Prepare to use commitment control. Notify the system you want to end commitment control. Prepare to use commitment control. Notify the system you want to begin commitment control.
Communications	CRTICFDEVE OVRICFDEVE	Create ICF Device Override ICF Device
Compiling	CRTRPGPGM CRTRPTPGM	Create RPG Program Create Auto Report Program
Consecutive Processing	OVRDBF	Override with Database file
Control Specification Data Area	CRTDTAARA DSPDTAARA	Create Data Area Display Data Area
Debugging	ADDBKP ADDTRC DSPBKP STRDBG	Add Breakpoint Add Trace Display Breakpoint Start Debug
Edit Codes	CRTEDTD DSPDTAARA	Create Edit Description (For User Defined Edit Code) Display Data Area
Printer Files	CRTPRTF OVRPRTF	Create Print File Override Print File
System Editor	STRSEU	Start Source Entry Utility

The Control Language and all of its commands are described in detail in the *CL Reference* manual.

System/38 Environment on the AS/400 System

The AS/400 system offers increased function over System/38. Because many RPG/400 language programs are written for the System/38, and because many programmers are already familiar with System/38, the AS/400 system also supports these programs under the System/38 environment. The CL command CALL QCL changes the AS/400 system display to appear to the user as a System/38 display. This is known as the System/38 environment. When you are in this environment, you can enter and compile RPG/400 programs as if you were using a System/38. The file naming conventions are the same as in System/38. You can also enter AS/400 CL commands in the System/38 environment. You can enter System/38 environment commands from the AS/400 system by library qualifying commands. The QSYS38/CRTRPGPGM command calls the System/38 environment RPG III compiler. For more information on the System/38 environment, see the *System/38 Environment Programmer's Guide/Reference*.

You can use the Source Entry Utility (SEU) to enter your RPG/400 source program interactively. Enter the CL command STRSEU to call SEU. If you specify the TYPE(RPG) parameter on this command, the RPG/400 syntax checker is called and detects RPG/400 syntax errors, statement by statement, while the source program is entered. Alternatively, you can enter a source program on diskettes and upload the program into a source file.

Note

To find out how to use RPG III in the System/38 environment, refer to the following:

- Appendix E, "System/38 Environment Option of the RPG Compiler" on page 531
- the *System/38 RPG III Reference Manual and Programmer's Guide* SC21-7725.

For information on System/38 devices and commands, refer to the appropriate manuals in the System/38 library.

AS/400 Utilities and Languages

The AS/400 system offers two utilities and a language that you may find useful for programming. They are the Screen Design Aid (SDA) utility, the Source Entry Utility (SEU), and the Structured Query Language (SQL).

The Source Entry Utility

You use the SEU to enter your code into the system. SEU also provides extensive syntax checking. For more information about SEU, refer to the *SEU User's Guide and Reference*.

The Screen Design Aid

The SDA utility makes it easier for you to create the displays your program requires. For more information about SDA, refer to the *SDA User's Guide and Reference*.

The Structured Query Language

The AS/400 system allows you to insert SQL/400 statements into RPG/400 programs. You enter SQL/400 statements on a calculation specification. The syntax is shown in Figure 2. You must observe the following rules:

- The starting delimiter /EXEC SQL must be entered into columns 7-15, with the slash in column 7.
- SQL/400 statements can be started on the same line as the starting delimiter.
- SQL/400 statements can be continued on any number of subsequent continuation lines. The continuation line delimiter is the + in column 7.
- SQL/400 statements cannot go past column 74.
- The ending delimiter /END-EXEC must be entered in columns 7-15, with the slash in column 7, on a separate line. This signals the end of the SQL/400 statements. It must be entered by itself, with no SQL/400 statements following it.

```
C  
C      |  
C      |  
C/EXEC SQL      (the starting delimiter)  
C+  
C+      (continuation lines containing SQL statements)  
C+  
.  
.  
.  
C/END-EXEC      (the ending delimiter)  
C      |  
C      |  
C      |
```

Figure 2. Syntax for Entering SQL/400 Statements into an RPG/400 Program

You must enter a separate command to process the SQL/400 statements.

Refer to the *SQL/400* Programmer's Guide* and the *Programming: Structured Query Language Reference* for the descriptions of how to code SQL/400 statements.

Restrictions

In the RPG/400 programming language, SQL/400 statements cannot be specified in the referred source member of a /COPY statement.

You should not use SQL/400 statements in an RPG automatic report program. Instead, you should use the CRTRPTPGM command to process your RPG automatic report programs and to save the generated RPG/400 source. Automatic report will generate RPG/400 source, to which you can add SQL/400 statements. To process your SQL/400 statements and generate an RPG object program, you should use the SQL/400 preprocessor. If SQL/400 statements are processed by the RPG/400 automatic report preprocessor, unpredictable results may occur.

Refer to the *SEU User's Guide and Reference* for information on how the SEU handles SQL/400 statement syntax checking, and to the *SQL/400* Programmer's Guide* and the *Programming: Structured Query Language Reference* for more information on the SQL/400 preprocessor.

Designing Your RPG/400 Program

Designing a program includes:

- Deciding what output you need from your program
- Deciding what processing will produce the output you need
- Deciding what input is required by and available to your program.

This sequence may seem backwards because it starts at the results (the output) and ends at the beginning (the input). Designing the output first is like knowing where you are going before you set out on a trip: it helps you decide the best way to get there.

Designing the Output

Your program produces output records. You must decide what to do with those records. In general, you have three choices (or any combination of the three choices):

- You can display them.
- You can print them.
- You can store them.

If you want to display the output records at your display station, you have to decide what information you want displayed and how you want it laid out. To define how you want your displays laid out, you use the display layout sheet. You can then use the SDA utility to create your own displays. For more information about SDA, refer to the *SDA User's Guide and Reference*.

If you want to print the output records, you have to decide what information you want printed (which fields from which records) and how you want that information laid out on the printed report. To indicate how you want the printed report laid out, use the printer layout sheet.

If you want to keep the output records in storage, you have to decide what information you want to keep and how you want to organize the fields in the output records.

After you design all your output records, you code those records on the RPG/400 file description specifications and output specifications.

Designing the Processing

Designing the processing means planning the calculations that produce the necessary output. When you design the processing, you must be aware of how the RPG/400 program cycle works. The RPG/400 program cycle controls certain read and write operations done on each record. As a result, the program cycle partly determines how you can process your data.

Designing the Input

After you decide what output you need and the calculations that produce the output, the next step is to determine where the input data for your program will come from. It might come from one or more files already on the system, from one or more display stations on your system, from one or more other systems, or from a combination of these sources. You have to know the names used for input files, the location of fields in the input records, the sequence of record types, the formats of numeric data, and the indicators used. When you know all these kinds of information, you can describe your input records on the RPG/400 input specifications.

Structured Programming in the RPG/400 Programming Language

Structured programming is an approach to design and coding that makes programs easy to understand, debug, and modify.

Three structures used in every computer program are:

- Sequential operation
- Conditional branching
- Repeating an operation based on a certain condition.

Ideally, a structured program is a hierarchy of modules that can have a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the structure.

The following discuss how the three structures can be accomplished in the RPG/400 programming language.

Sequential Operation

Sequential operation means any series of instructions that is processed one instruction after another, without transferring control to another part of the program.

Conditional Branching

If Else Structure

An example of an If-Then-Else conditional branching structure in simple English is:

```
IF the weather is cold,  
THEN I will wear my coat;  
ELSE, I will leave my coat at home.
```

Figure 3 is a flowchart of a conditional branch.

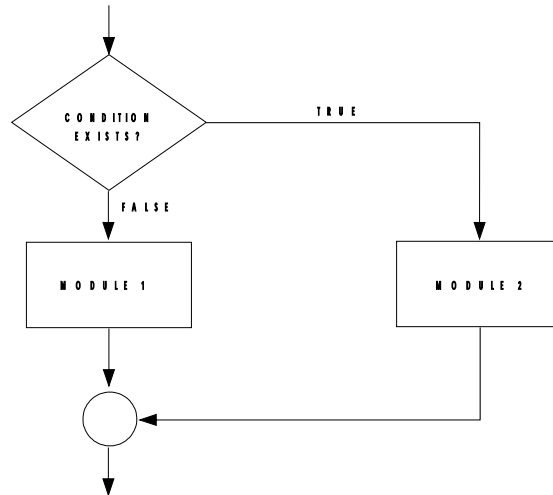


Figure 3. Flowchart of a Conditional Branch

In the RPG/400 programming language, the If-Then-Else structure is carried out through the operation codes IFxx, ELSE, and END. Figure 4 shows a design for a conditional branch using the IFxx, ELSE, and END operation codes.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* In this example, if CENTR equals Y or if CENTR equals N, then
C* indicator 52 is set off by moving '0' to *IN52. If CENTR equals
C* neither Y nor N, then indicator 52 is set on by moving '1' to
C* *IN52. The END statement ends the IF/THEN/ELSE group.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C      CENTR      IFEQ 'Y'
C      CENTR      OREQ 'N'
C              MOVE '0'          *IN52
C              ELSE
C              MOVE '1'          *IN52
C              END
  
```

Figure 4. Design for a Conditional Branch Using the IF/ELSE/END Operations

SELEC Structure

An example of a SELEC-WHEN-OTHER conditional branching structure in simple english is:

```
SELEC  
WHEN the weather is warm  
  I will wear my sunhat  
  I will go to the beach  
WHEN the weather is cool  
  I will wear my jacket  
OTHERwise, I will not go outside
```

Figure 5 is a flowchart of a SELEC-WHEN-OTHER conditional branch.

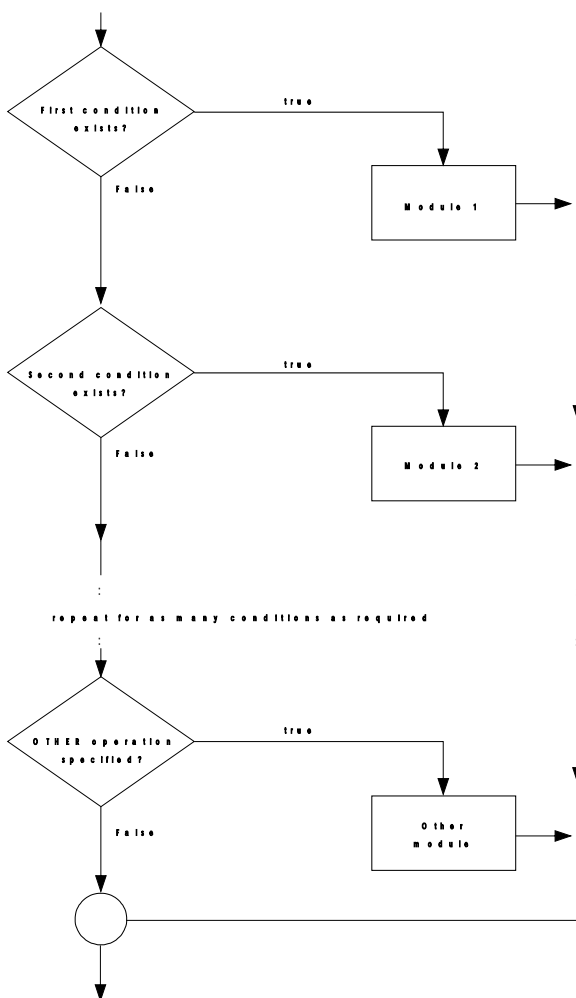


Figure 5. Flowchart of a SELEC-WHEN-OTHER Conditional Branch

In the RPG/400 programming language, the SELEC-WHEN-OTHER structure is carried out through the operation codes of SELEC, WHxx, and OTHER. Figure 6 shows conditional branching using the SELEC, WHxx, and OTHER operation codes.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C* If X equals 1 then do the operations in sequence 1; if
C* X does not equal 1, then if Y=2 and X<10 do the operations
C* in sequence 2. If neither condition is true, then do the
C* operations in sequence 3.
C*
C          SELEC
C      X      WHEQ 1
C          :
C          seq 1
C      Y      WHEQ 2
C      X      ANDLT10
C          :
C          seq 2
C          OTHER
C          :
C          seq 3
C          ENDSL
C*

```

Figure 6. Conditional Branching Using the SELEC/WHxx/OTHER Operations

Other Conditional Branching Structures

There are three other ways you can create conditional branches:

- The CASxx operation
- The GOTO operation and conditioning indicators
- The CABxx operation.

You can also create a branch to a subroutine with the EXSR operation and conditioning indicators.

Repeating an Operation

The RPG/400 programming language implements three repeat structures—Do, Do While, and Do Until—by means of the DOWxx, DOUxx, and D0 operation codes and the END operation code.

Do Operation

Figure 7 on page 10 is a flowchart of a Do operation, and Figure 8 on page 11 illustrates the coding.

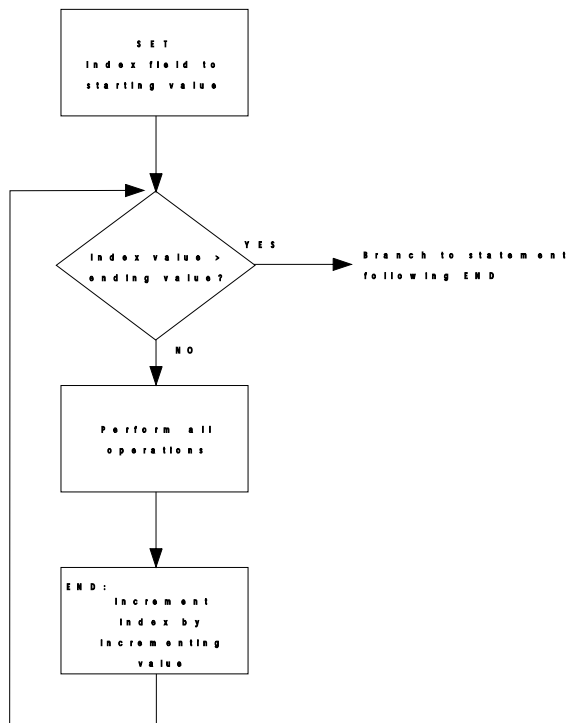


Figure 7. Flowchart of a Do Operation

This is how the Do operation works:

1. Set the index field (result field) to the starting value (factor 1).
2. Test if the index field value is greater than the ending value (factor 2).
If the index field value is greater than the ending value, control passes to the statement following the END statement.
3. If the index field value is not greater than the ending value, the operations between the DO statement and the END statement are processed.
4. At END, the index field value is increased by the increment value specified in factor 2 on the END statement, or by 1 if the increment is not specified.
5. Control passes to step 2 above.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following example illustrates a Do operation. Because factor
C* 1 of the DO statement is blank, the starting value of Y is 1, and
C* because factor 2 of the END statement is blank, the increment
C* value of Y is 1. Factor 2 of the DO statement contains the value
C* 10, which is the ending value for the DO routine.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          Z-ADD1          X          20
C          DO  10          Y          30
C          X          LOKUPTABA          TABR          50
C  50          TABR          MULT 1.04          RATE          72
C          MOVE '1'          *IN90
C          EXCPT
C          MOVE '0'          *IN90
C          ADD  1          X
C          END
    
```

Figure 8. Design for a Do Operation Using the DO and END Operation Codes

Do While Operation

If you test the condition first and then process the operations, the structure is called a Do While. An example of a Do While operation is:

1. Compare a sum with 5.
2. If the sum is less than 5, add 1 to the sum.
3. Repeat steps 1 and 2 until the sum is equal to or greater than 5.

Figure 9 is a flowchart of a Do While operation, and Figure 10 on page 12 illustrates the coding of a Do While operation.

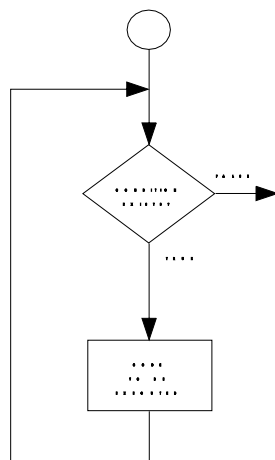


Figure 9. Flowchart of a Do While Operation

Structured Programming

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code determines if the subfile has been filled.
C* If indicator 32 is off (*IN32 equal 0), the DOWEQ operation
C* processes until the remainder of the subfile is filled with
C* blank records.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN32      DOWEQ '0'
C          MOVE *BLANKS  STATUS
C          MOVE *BLANKS  PRCDEX
C          MOVE *BLANKS  RSCDEX
C          Z-ADD0        EHWRKX
C          Z-ADD0        ACDATX
C          Z-ADD0        TFRRN
C          ADD 1         RECNO
C          WRITEEMPFIL           32
C          END
C* The preceding END denotes the end of the Do While operation.
```

Figure 10. Design for a Do While Operation Using the DOWxx Operation Code

Notice in Figure 10 (the Do While) that the program first tests if the condition is true. If it is true, the code between the DOW and the END operations is processed. The program then goes back to test again if the condition is still true, and the entire cycle is repeated. If the condition is no longer true, control passes to the instruction immediately following the END operation.

Do Until Operation

If you process the operations first and then test the condition, the structure is called a Do Until operation. An example of a Do Until operation is:

1. Add 1 to a sum.
2. Compare the sum with 5.
3. If the sum is less than 5, repeat steps 1 and 2 .

Figure 11 is a flowchart of a Do Until operation, and Figure 12 on page 14 illustrates the coding.

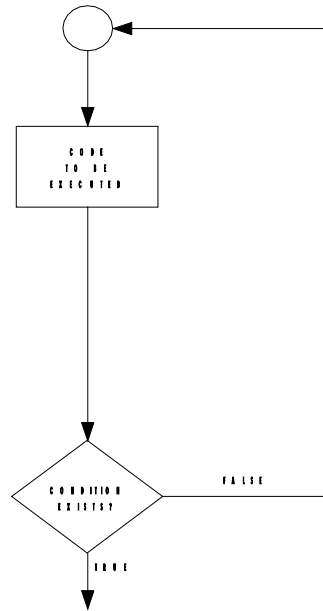


Figure 11. Flowchart of a Do Until Operation

Structured Programming

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C*
C* The following lines of code perform a Do Until condition. The
C* program loops between the DOUEQ statement and the END statement
C* until end of file (*IN50 equal 1) is reached.
C      EMPSR      BEGSR
C*
C      *IN50      DOUEQ'1'
C      READ RCEMP      50
C* The following lines of code add current month hours to the year-
C* to-date hours for the employee master file. Since factor 1 is
C* not specified in the statements, factor 2 is added to the result
C* field and the result is placed in the result field. If *INU4
C* is on, this session is being run for year end, and the current
C* year hours are moved to the prior year hours.
C      ADD EPHRC      EPHRY
C      ADD EPNRC      EPNRY
C U4      MOVE EPHRY      EPHRP
C U4      MOVE EPNRY      EPNRP
C* The following code clears the current month hours fields by
C* zeroing them and adding 0 to them. If *INU4 is on, this session
C* is being run for year end, and the current year hours must be
C* zeroed as well.
C      Z-ADD0      EPHRC
C      Z-ADD0      EPNRC
C U4      Z-ADD0      EPHRY
C U4      Z-ADD0      EPNRY
C* The following code updates the employee master file using the
C* RCEMP format.
C      UPDATRCEMP
C      END
C* The preceding END statement is associated with the DOUEQ
C* statement.
```

Figure 12. Design for a Do Until Operation Using the DOUxx Operation Code

Summary of Structured Programming Operation Codes

The structured programming operation codes are:

- IFxx (If/Then)
- ELSE (Else Do)
- ENDyy (End)
- DO (Do)
- DOWxx (Do While)
- DOUxx (Do Until)

- ANDxx (And)
- ORxx (Or)
- CASxx (Conditional Invoke Subroutine)
- SELEC (Select a module)
- WHxx (When)
- OTHER (Otherwise).

where xx can be:

- GT Factor 1 is greater than factor 2.
- LT Factor 1 is less than factor 2.
- EQ Factor 1 is equal to factor 2.
- NE Factor 1 is not equal to factor 2.
- GE Factor 1 is greater than or equal to factor 2.
- LE Factor 1 is less than or equal to factor 2.
- Blanks Factor 1 is not compared to factor 2 (unconditional processing). This is valid for CASxx operation only.

and where yy can be:

- CS End a CAS group.
- DO End a DO, DO UNTIL, or DO WHILE group.
- IF End an IF group.
- SL End a SELEC group.

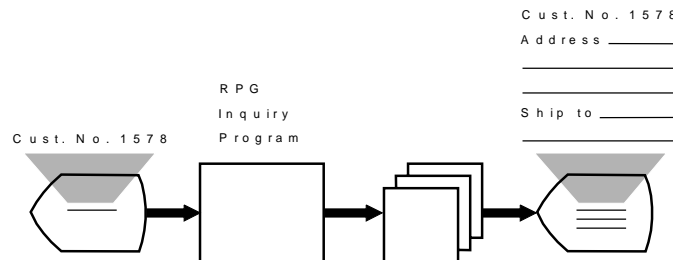
Designing Applications

Application design involves determining whether to create one program to do all of the required functions, or to create multiple programs to make up an application.

Single Program Design

In a single program design, all functions are done within one program. Single program design applies to both batch and interactive programs. It is best used when there are few, relatively simple functions.

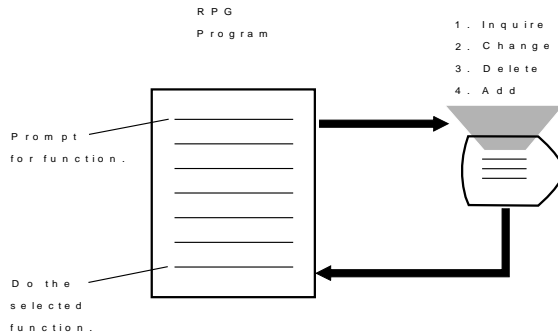
For example, an interactive inquiry program that accepts a customer number from an operator, finds the corresponding record in a customer master file, and displays the record as a simple program that could have a single program design.



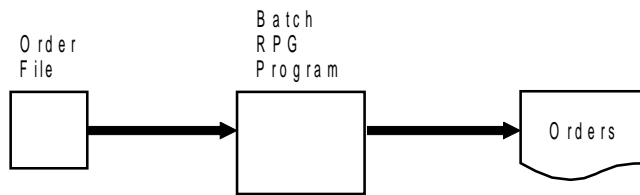
A slightly more complex program that might also have a single program design is a file maintenance program that allows an operator to:

- Inquire into a record
- Change a record
- Delete a record
- Add a record.

Designing Applications



An example of a batch program that has a single program design is a program that prints a list of orders that each operator entered during the day.



Modular Program Design

Modular program design includes using multiple programs to do multiple functions, one function per program. Modular program design can be applied to both batch and interactive programs. For example, the order entry application shown in Figure 13 is designed to have four programs:

- An RPG/400 or CL mainline program
- An RPG/400 program that prompts for the customer number and shows customer information on the display
- An RPG/400 program that accepts input of line items from the order
- An RPG/400 program that calculates totals for the order.

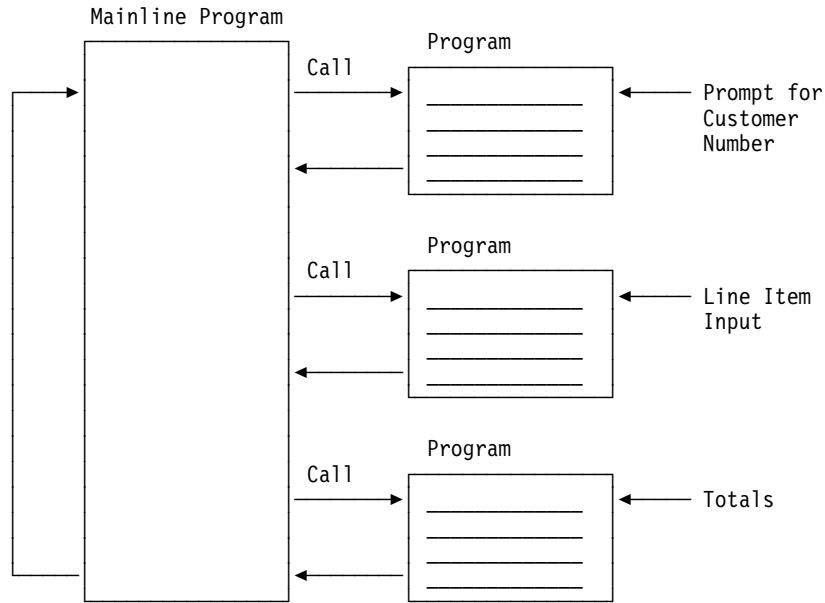


Figure 13. Modular Design for an Order Entry Application

A modular program design has several potential advantages:

- Designing, coding, testing, and maintaining several small programs can be easier than designing, coding, testing, and maintaining one large, complex program. This choice is a matter of personal preference, but it is often beneficial to keep your programs small and as simple as possible.
- CL functions can be requested from RPG/400 programs because the AS/400 system allows RPG/400 programs and CL programs to call one another.

A single, long-running program might have sections of code that run infrequently. A modular design could arrange to have the seldom-used code called only when needed.

A potential disadvantage of modular program design is the additional calling of programs that is required. These calls take time to code and might require additional system overhead for program processing.

Examples of Application Design

Following are descriptions of modular programs that illustrate some design approaches.

The order entry function shown in Figure 14 has three sub-functions:

- Accepting heading information about an order
- Accepting line item input from the order
- Calculating totals for the order.

One way to design this application is to have a CL mainline program call RPG/400 programs to do the functions.

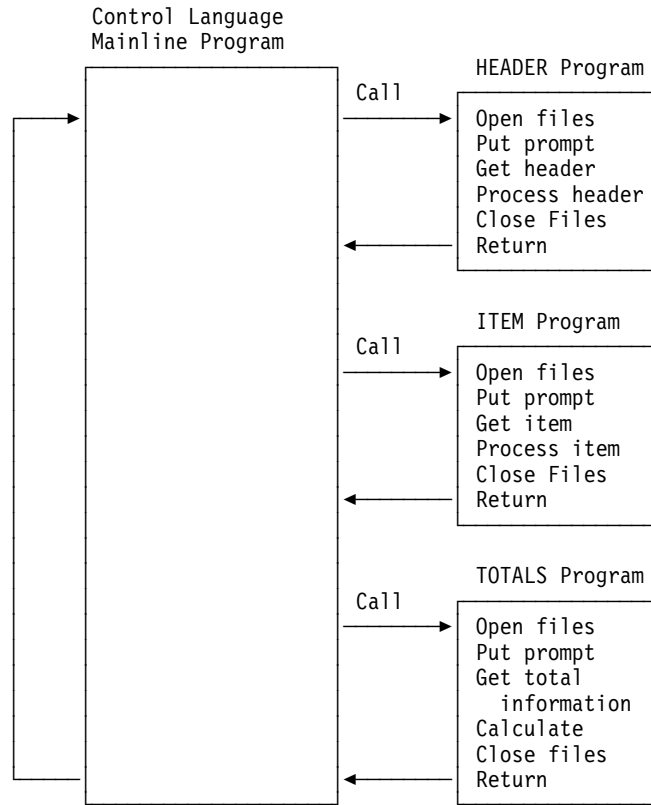


Figure 14. Example of Application Design for an Order Entry Function

Each of the RPG/400 programs:

- Opens files
- Displays a prompt for user information and input
- Accepts input from the user
- Processes the information
- Closes the files
- Returns to the mainline program.

The following events occur after a user enters input:

1. The input is processed.
2. Files are closed.
3. Control returns to the mainline program.
4. The mainline program calls the next program.
5. That program prompts for user input.

All processing of input and output from work stations and all opening and closing of files occurs in the RPG/400 programs. Therefore, the user might have to wait for a while after entering a display before seeing the next display.

A change in the previous design that might shorten response times and make more efficient use of system resources is shown in Figure 15.

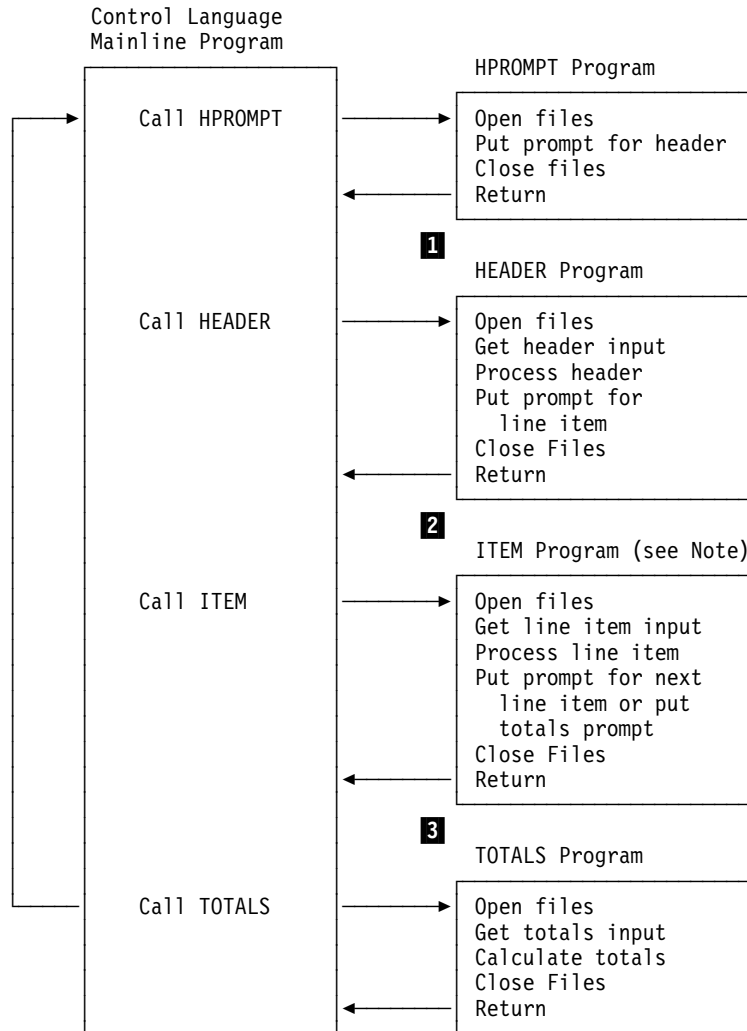


Figure 15. Example of Changed Application Design for an Order Entry Function

Note: Rather than returning unconditionally to the mainline program, the ITEM program could be designed to loop within itself as long as line items are being entered.

This modification allows user data entry to occur while programs are started and files are opened and closed. The overlap of data entry and AS/400 system processing occurs at points **1**, **2**, and **3**.

For the previous two examples of modular program design, all input from and output to work stations occurs in the programs. For the example in Figure 16, a series of operations occur in an RPG/400 mainline program.

Designing Applications

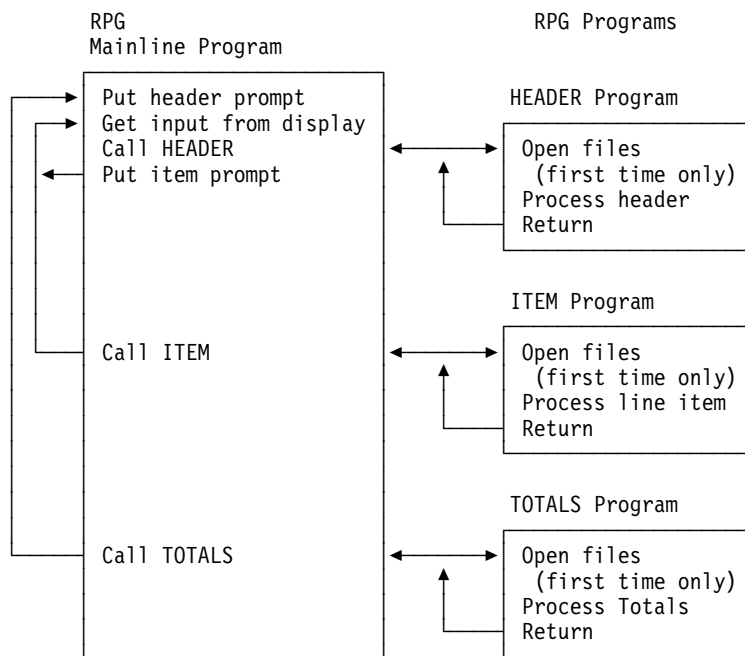


Figure 16. Example of Application Design with Input and Output in Mainline Program

The input from the display determines the program to call. If a header is read, HEADER is called and the header record is passed as a parameter. If a line item is read, ITEM is called and a line item record is passed as a parameter. If total information is read, TOTALS is called and a total record is passed as a parameter.

The programs leave files open until the job ends, thereby eliminating open and close processing time for the files. The programs do not end when they return to the mainline program.

Chapter 2. Entering RPG/400 Specifications

After designing your program, you must write the individual statements that you will combine into a source program. These statements are coded on RPG/400 specification sheets. Each line coded on a specification sheet represents a statement in the source program. Each specification sheet contains 80 columns. Column headings indicate the kind of information to code in particular columns.

This chapter describes the kinds of specifications you can enter when creating an RPG/400 source program. This chapter also describes how to use a text editor, such as SEU, to enter this information directly into the system and thus begin creating your source program online.

The RPG/400 Specifications

There are seven kinds of RPG/400 specifications. When your source program is compiled, these specifications must be in the following sequence:

1. Control specifications
2. File description specifications
3. Extension specifications
4. Line counter specifications
5. Input specifications
6. Calculation specifications
7. Output specifications.

Each of these specifications is described briefly in this chapter. The *RPG/400 Reference* provides detailed descriptions for these specifications.

RPG/400 programs do not have to use all specifications. A typical program may use file description, input, calculation, and output specifications.

The Control Specification

The control specification provides the RPG/400 compiler with information about your program and your system. This includes:

- Name of the program
- Date format for the program
- If an alternative collating sequence or file translation is used.

Note: The control specification is optional.

File Description Specifications

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

- Name of the file
- How the file is used
- Size of records in the file
- Input or output device used for the file
- If the file is conditioned by an external indicator.

Extension Specifications

Extension specifications describe all record address files, table files, and array files used in the program. The information includes:

- Name of the file, table, or array
- Number of entries in a table or array input record
- Number of entries in a table or array
- Length of the table or array entry.

Line Counter Specifications

Line counter specifications describe the page or form on which output is printed. The information includes:

- Number of lines per page
- Line of the page where overflow occurs.

Input Specifications

Input specifications describe the records, fields, data structures and named constants used by the program. The information in the input specifications includes:

- Name of the file
- Sequence of record types
- Whether record-identifying indicators, control-level indicators, field-record-relation indicators, or field indicators are used
- Whether data structures, lookahead fields, record identification codes, or match fields are used
- Type of each field (alphanumeric or numeric; packed-decimal, zoned-decimal, or binary format)
- Location of each field in the record
- Name of each field in the record
- Named constants.

Calculation Specifications

Calculation specifications describe the calculations to be done on the data and the order of the calculations. Calculation specifications can also be used to control certain input and output operations. The information includes:

- Control-level and conditioning indicators for the operation specified
- Fields or constants to be used in the operation
- The operation to be processed
- Whether resulting indicators are set after the operation is processed.

Output Specifications

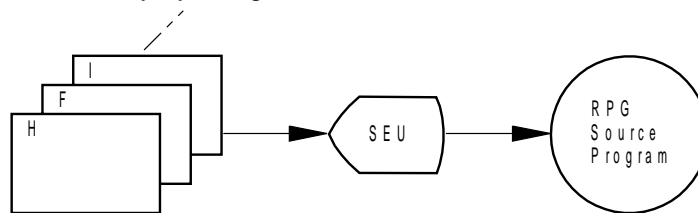
Output specifications describe the records and fields in the output files and the conditions under which output operations are processed. The information includes:

- Name of the file
- Type of record to be written
- Spacing and skipping instructions for PRINTER files
- Output indicators that condition when the record is to be written
- Name of each field in the output record
- Location of each field in the output record
- Edit codes and edit words
- Constants to be written
- Format name for a WORKSTN file.

Entering Your Program

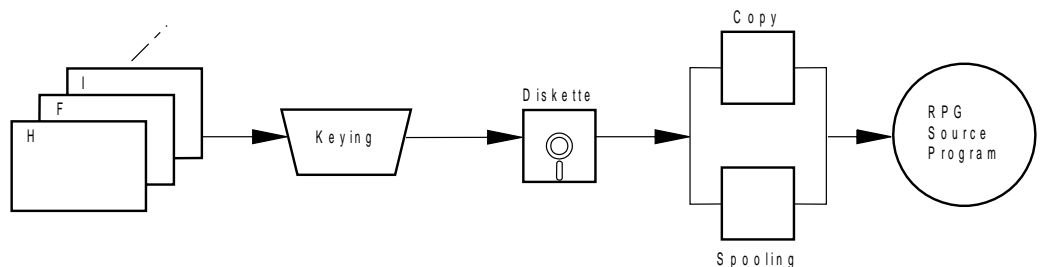
After you have written your RPG/400 program on the specifications forms, you must enter it into source files in the system. You can enter the source program in two ways:

- Interactively by using SEU:



The *SEU User's Guide and Reference* provides a complete description of how to enter or update an RPG/400 source program using SEU.

- In a batch manner (that is, from diskette) by using either the OS/400 system copy or spooling functions:



The *Data Management Guide* provides more information on how to use the copy or spooling function for batch entry of the source program.

Note: Whichever method of source entry you use, you can use lowercase letters only in literals, constants, comments, array data, and table data. All other information must be in uppercase letters.

Entering Your Program

Chapter 3. Compiling an RPG/400 Program

There are two environments that you can compile source programs from: the AS/400 system environment, and the System/38 environment. Consequently, there are two ways of compiling source programs. This chapter describes:

- Using the CL command CRTRPGPGM to compile an RPG/400 source program in AS/400 system environment
- Using the CL commands CALL QCL and CRTRPGPGM to compile an RPG/400 source program in the System/38 environment.

This chapter also contains information on interpreting a compiler listing.

To compile a program, you must ensure that the library QTEMP is in the library list. The CL command CRTRPGPGM calls the compiler to create an RPG/400 program object and a listing. (If externally described files are used in the program, the OS/400 system provides information about the files to the program during compilation.) The following figure shows an overview of the compilation process:

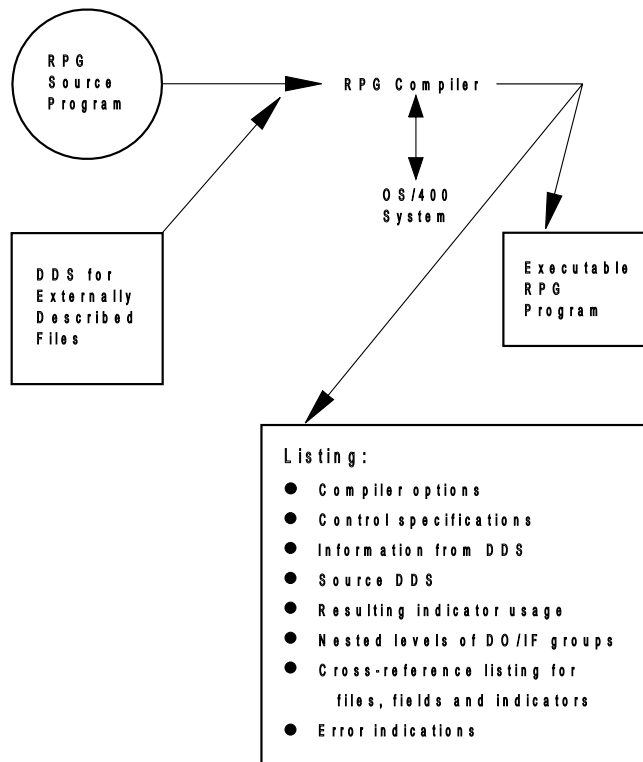


Figure 17. Overview of the Compilation Process

The compiler checks the syntax of the RPG/400 source program line by line and the interrelationships between the lines. For example, it checks that all field names are defined and, if a field is multiply defined, that each definition has the same attributes.

The RPG/400 compiler supports a source file record length of 102. In addition to the usual fields of sequence number (6 characters), last-changed date (6 characters), and the data (80 characters), a field of 10 characters that can contain addi-

Create RPG/400 Program (CRTRPGPGM) Command

tional information is placed at the end of the record (positions 93-102). This information is not used by the RPG/400 compiler but is placed on the extreme right of the compiler listing. You, the programmer, place information into this field. If you want to use the additional field, create a source file with a record length of 102. The AS/400 system has an IBM-supplied RPG/400 source file called QRPGRSRC, which has a record length of 92.

Create RPG400 Program (CRTRPGPGM) Command

To compile an RPG/400 source program into a program object, you must enter the CL command CRTRPGPGM (Create RPG/400 Program) to call the RPG/400 compiler. RPG/400 program objects are created with the public authority of *LIBCRTAUT. You may want to change this authority to maintain greater security on your system.

If the RPG/400 compiler stops because of errors, the escape message QRG9001 is issued. A CL program can monitor for this exception by using the CL command MONMSG (Monitor Message). See Chapter 4, "Error Messages, Testing, and Debugging."

The compiler creates and updates a data area with the status of the last compilation. This data area is named RETURNCODE, is 400 characters long, and is placed into library QTEMP. You can access the RETURNCODE data area by specifying RETURNCODE in factor 2 of an *NAMVAR DEFN statement. The data area RETURNCODE has the following format:

Table 2 (Page 1 of 2). Contents of the Data Area RETURNCODE

Byte	Content and Meaning
1	Character 1 means a program was created.
2	Character 1 means the compilation failed because of compiler errors.
3	Character 1 means the compilation failed because of source errors.
4	Character 1 means compiled from source generated by automatic report.
5	Character 1 means program resolution monitor was not called because *NOGEN option was selected on CRTRPGPGM command.
6-10	Number of source statements.
11-12	Severity level from command.
13-14	Highest severity on message diagnostic.
15-20	Number of errors found in program.
21-26	Compile date.
27-32	Compile time.
33-100	Not set.
101-110	Program name.
111-120	Program library name.
121-130	Source file name.

Table 2 (Page 2 of 2). Contents of the Data Area RETURNCODE

Byte	Content and Meaning
131-140	Source file library name.
141-150	Source file member name.
151-160	Compiler listing file name.
161-170	Compiler listing library name.
171-180	Compiler listing member name.
181-190	Automatic report source file name.
191-200	Automatic report library name.
201-210	Automatic report member name.
211-370	Not set.
371-378	Size of intermediate representation of program passed to program resolution monitor.
379	Not set.
380-384	Total compile time.
385	Not set.
386-390	Time used by compiler.
391-395	Time used by program resolution monitor.
396-400	Time used by translator.

All object names specified on the CRTRPGPGM command must be composed of alphanumeric characters, the first of which must be alphabetic. The full OS/400 system naming convention is allowed. The length of the names cannot exceed 10 characters. See the *CL Programmer's Guide* for a detailed description of OS/400 object naming rules and for a complete description of OS/400 command syntax.

It is unlikely that the system internal size limits for a program will be exceeded. However, if these limits are exceeded, the program must be rewritten, usually as multiple programs.

Using the CRTRPGPGM Command

You can call the RPG/400 compiler in one of three ways:

- Interactively from the CRTRPGPGM command display screen using prompts. You start the display, illustrated in Figure 19 on page 31 and Figure 20 on page 37, by typing the CL command CRTRPGPGM and then pressing F4.
- Entering CRTRPGPGM followed by only those parameters by keyword that override the default settings. This statement is entered on the command line interactively or as part of a batch input stream.

Create RPG/400 Program (CRTRPGPGM) Command

- Entering CRTRPGPGM followed only by the parameter values, in the proper sequence. This method is most often used when you are submitting the compiling request as part of a batch input stream, or if you are including the compiling request as part of a CL program. This method can also be used interactively, but you are limited by CL to entering only the first three parameter values.

Note: Any default on the CRTRPGPGM command or any other CL command can be changed using the CL command CHGCMDDFT (Change Command Default). Refer to the *CL Reference* for more information.

Elements of the CRTRPGPGM Command Lines

The descriptions that follow refer to the three elements of the compiler command line:

- The CL compiler command word CRTRPGPGM.
- The parameter, which is referred to by a keyword such as PGM, SRCFILE, GENOPT, and so on.
- The value for the parameter. This can be a predefined value or an object name.

All object names specified must consist of alphanumeric characters. The first character must be alphabetic, and the length of the name cannot exceed 10 characters. You can use the full OS/400 system naming convention.

Entering Elements from the CRTRPGPGM Command Display

Type CRTRPGPGM, and press F4. The CRTRPGPGM prompt screens appear. Press F10 to get additional parameters. These screens, and the values you can enter on them, are described later in this chapter.

Each parameter on the screen displays a default value. Move the cursor past items where you want the default value to apply. Type over any items where you want to set a different value or option. If you are not sure about what to set a particular parameter to, type a question mark (?) as the first character in that field and press Enter to receive more detailed information. The question mark must be followed by a blank.

When you have set all values to your satisfaction, press Enter.

Entering Only Certain Parameters

All of the CRTRPGPGM parameters have default values. Simply type CRTRPGPGM, followed only by those parameters (specified by keyword) whose default settings you want to override. Separate parameters by spaces; enter values for each parameter by enclosing the value or values in parentheses.

For example, to change the program and library name, and accept default values for all other parameters, enter:

```
CRTRPGPGM PGM(newlibrary/newname)
```

Entering Only the Parameter Values

You have the choice of entering only the parameter values without specifying the parameter keywords. Because there is no keyword to tell the system which value belongs to which parameter, you must enter all the values in the sequence shown below. You need not enter the entire set of options, but you must enter the options for all the parameters up to the one you want. The system uses the default values for the remaining parameters.

For example, to compile a source program in member ABC in file QRPGRSRC in library SRCLIB, enter:

```
CRTRPGPGM QTEMP/ABC SRCLIB/QRPGRSRC *PGM
```

Notice that you also had to enter names for the program and library for the compiled program. The system recognizes which option belongs to which parameter by the position of the value on the compiler command line. You can enter a maximum of three parameter values positionally.

For more information on AS/400 system commands, see the *CL Reference*.

CRTRPGPGM Command

The entire syntax diagram for the CRTRPGPGM command is shown in Figure 18 on page 30.

Read the syntax diagram from left to right, from top to bottom, following the path of the line.

Control Language (CL) commands, parameters, and keywords can be entered in either uppercase or lowercase characters. In this manual they are shown in uppercase (for example, PARAMETER, PREDEFINED-VALUE). Variables appear in lowercase italic letters (for example, *user-defined-value*). Variables are user-defined names or values.

For information on how to read syntax diagrams, see “How to Interpret Syntax Diagrams” on page xiv.

CRTRPGGM Command

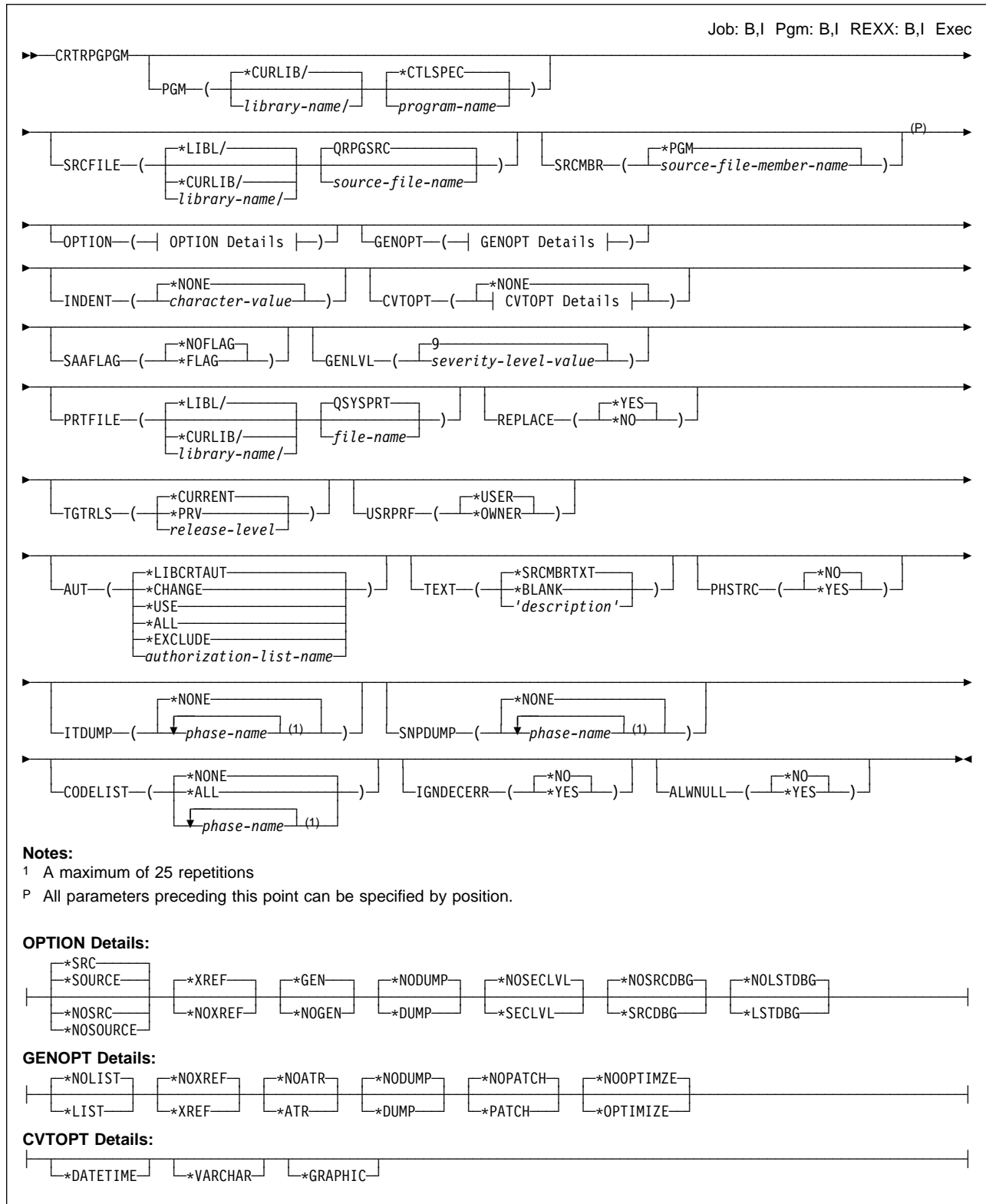


Figure 18. Syntax of the CRTRPGGM Command

Following are examples of the prompt screens for the CRTRPGGM command. The example screens are provided in sets. The first screen in the set describes the values you can enter, the second screen presents the keywords and defaults. You

can switch between the values and keywords screens by pressing F11. The text that follows the screens describes those keywords and defaults.

In the description of the parameters, all defaults are explained first and highlighted. The parameters are presented in sequence. Follow this sequence if you are entering only the parameter values without the corresponding parameter abbreviation.

Note: For a description of the differences between compiling RPG/400 and System/38 environment RPG III programs, see Appendix E, "System/38 Environment Option of the RPG Compiler."

```

                                Create RPG/400 Program (CRTRPGPGM)

Type choices, press Enter.

Program . . . . . *CTLSPEC   Name, *CTLSPEC
  Library . . . . . *CURLIB   Name, *CURLIB
Source file . . . . . QRPGRSRC Name, QRPGRSRC
  Library . . . . . *LIBL     Name, *LIBL, *CURLIB
Source member . . . . . *PGM   Name, *PGM
Generation severity level . . . . . 9      0-99
Text 'description' . . . . . *SRCMBRTXT

-----

                                Additional Parameters

Source listing options . . . . . _____ *SOURCE, *NOSOURCE, *SRC...
      + for more values          _____
Generation options . . . . . _____ *LIST, *NOLIST, *XREF...
      + for more values          _____
Source listing indentation . . . . . *NONE      Character value, *NONE
                                          More...

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

```

Figure 19 (Part 1 of 2). First Set of CRTRPGPGM Prompt Screens

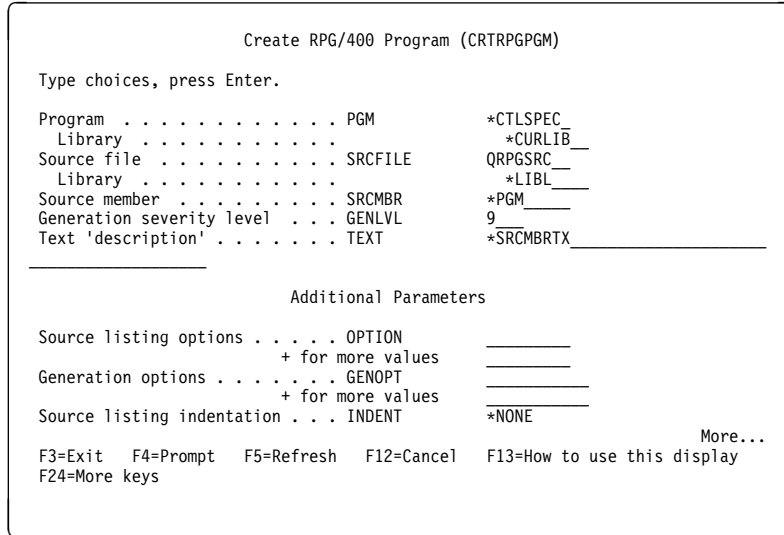


Figure 19 (Part 2 of 2). First Set of CRTRPGPGM Prompt Screens

PGM

Specifies the library and program name by which the compiled RPG/400 program is to be known. If no library is specified, the created program is stored in the current library.

***CTLSPEC**

The program name specified in positions 75 through 80 of the control specification is used.

If the program name is not specified on the control specification, but the source program is from a database file, the member name, specified by the SRCMBR parameter, is used as the program name. If the source is not from a database file, the program name defaults to RPGOBJ.

program-name

Enter the name by which the program is to be known.

***CURLIB**

The compiled program is stored in the current library. If you have not specified a current library, QGPL is used.

library-name

Enter the name of the library where the compiled program is to be stored.

SRCFILE

Specifies the name of the source file that contains the RPG/400 source program to be compiled and the library where the source file is located.

QRPGSRC

The default source file QRPGSRC contains the RPG/400 source program to be compiled.

source-file-name

Enter the name of the source file that contains the RPG/400 source program to be compiled.

***LIBL**

The system searches the library list to find the library where the source file is located.

***CURLIB**

The current library is used to find the source file. If you have not specified a current library, QGPL is used.

library-name

Enter the name of the library where the source file is stored.

SRCMBR

Specifies the name of the member of the source file that contains the RPG/400 source program to be compiled. This parameter can be specified only if the source file named in the SRCFILE parameter is a database file.

***PGM**

Use the name specified by the *PGM parameter as the source file member name. The compiled program will have the same name as the source file member. If no program name is specified by the *PGM parameter, the command uses the first member created in or added to the source file as the source member name.

source-file-member-name

Enter the name of the member that contains the RPG/400 source program.

GENLVL

Specifies whether or not a program object is generated, depending on the severity of the errors encountered. A severity-level value corresponding to the severity level of the messages produced during compilation can be specified with this parameter. If errors occur in a program with a severity value less than 30, and if a severity-level greater than that of the program is specified for this parameter the program is compiled; however, the program may contain errors

that cause unpredictable results when the program is run. For program errors equal to or greater than severity 30, the compilation of the program may be ended or the program object may not be generated, regardless of the value of this parameter. Specifying a value greater than 30 is not recommended for this parameter.

9

A program object will not be generated if you have messages with a severity-level greater than or equal to 9.

severity-level-value:

Enter a number, 0 through 99.

Note: The severity-level value of RPG/400 compile messages does not exceed 50.

TEXT

Lets the user enter text that briefly describes the program and its function. The text appears whenever program information is displayed.

***SRCMBRTXT**

The text of the source member is used.

***BLANK**

No text appears.

'description'

Enter the text that briefly describes the program and its function. The text can be a maximum of 50 characters and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string. Apostrophes are not required if you are entering the text on the prompt screen.

OPTION

Specifies the options to use when the source program is compiled. You can specify any or all of the options in any order. Separate the options with a blank space.

***SOURCE**

Produces a source listing, consisting of the RPG/400 program input and all compile-time errors.

***NOSOURCE**

A source listing is not produced. If *NOSOURCE is specified, the system

assumes that you also don't want a cross-reference listing and *NOXREF is also specified.

The acceptable abbreviation for *SOURCE is *SRC, and for *NOSOURCE is *NOSRC.

***XREF**

Produces a cross-reference listing and key-field-information table (when appropriate) for the source program.

Note: If you also want to specify *NOSOURCE or *NOSRC, you must explicitly specify *XREF or else *NOXREF is assumed.

***NOXREF**

A cross-reference listing is not produced.

Note: If either *NOSOURCE or *NOSRC is also specified, the usual default (*XREF) is overridden and *NOXREF is the default.

***GEN**

Creates a program object that can be run after the program is compiled.

***NOGEN**

Do not create a program object.

***NODUMP**

Do not dump major data areas when an error occurs during compilation.

***DUMP**

Dump major data areas when an error occurs during compilation.

***NOSECLVL**

Do not print second-level message text on the line following the first-level message text.

***SECLVL**

Print second-level message text.

***NOSRCDBG**

Do not generate source level error and debug information.

***SRCDBG**

Generate source level error and debug information. Produce an event file even if the compiler completes without error.

***NOLSTDBG**

Do not generate error and debug information.

***LSTDBG**

Generate a listing view and error and debug information required for the listing view.

Note: You can only use the *NOSRCDBG, *SRCDBG, *NOLSTDBG and *LSTDBG options if you are using the AD/Cycle CoOperative Development Environment/400 product to compile your program. If you specify one or more of these options but do not have the AD/Cycle CODE/400 product installed, the RPG/400 compiler will not continue processing and an error message is issued. For more information on these options, see the *CODE Debug Tool User's Guide and Reference*, SC09-1622.

GENOPT

Specifies the options to use to create the program object: the printing of the intermediate representation of a program (IRP), a cross-reference listing of objects defined in the IRP, an attribute listing from the IRP, and the program template. You can also specify options in the GENOPT parameter to reserve a program patch area, and to improve a program for more efficient running. These results may be useful if a problem occurs when you are trying to run the compiled program. You can specify any or all of the options in any order. Separate the values with a blank. For a description of the GENOPT parameter and the information it provides, see "Compiler Debugging Options" on page 498 in Appendix A, "RPG Compiler and Auto Report Program Service Information."

***NOOPTIMIZE**

Do not process program optimization.

***OPTIMIZE**

Process program optimization. With *OPTIMIZE, the compiler generates a program for more efficient processing and one that will possibly require less storage. Specifying *OPTIMIZE can substantially increase the time required to create a program. Existing program objects can be optimized with the CL command CHGPGM.

INDENT

Specifies whether or not the compiled RPG/400 program's source listing is generated with the indentation of structured operations for enhanced readability. Also specifies the characters that are used to mark the structured operation clauses.

***NONE**

A listing without indentation will be produced by the compiler.

character-value

The source listing is indented for structured operation clauses. Alignment of statements and clauses are marked using the characters you choose. You can choose any character string up to 2 characters in length. If you want to use a blank in your character string, you must enclose it in single quotation marks.

Note: The indentation may not appear as expected if there are errors in the RPG/400 program.

The second set of prompt screens shown in Figure 20 provides more values and keywords that you can enter for the CRTRPGPGM command.

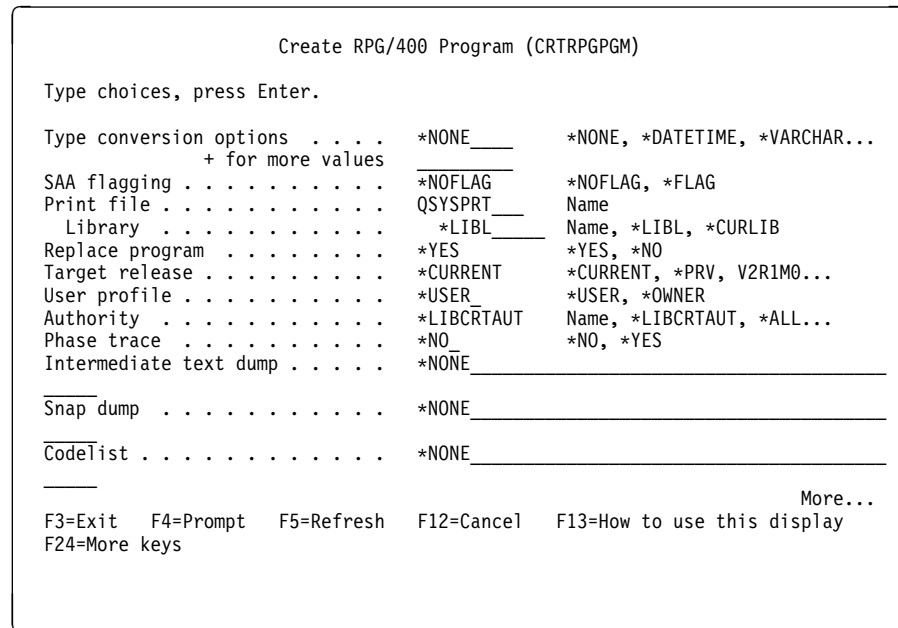


Figure 20 (Part 1 of 2). Second Set of CRTRPGPGM Prompt Screens

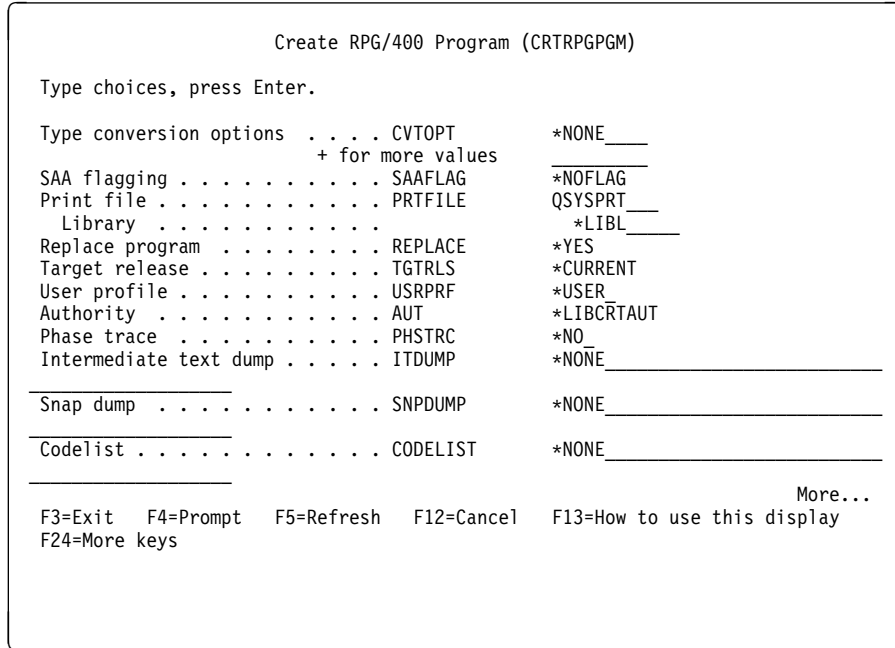


Figure 20 (Part 2 of 2). Second Set of CRTRPGPGM Prompt Screens

CVTOPT

Specifies how the RPG/400 compiler handles date, time, and timestamp database data types, and variable-length data types which are retrieved from externally-described files. See “SAA Data Types” on page 247 for a detailed description of how the RPG/400 compiler supports SAA data types.

***NONE**

Date, time, timestamp and variable-length database data types are ignored and not accessible in the RPG/400 program. A severity 0, compile-time informational message is issued when a record format contains ignored fields.

***DATETIME**

Specifies that date, time, and timestamp database data types are to be declared as fixed length character fields and are accessible in the RPG/400 program.

***VARCHAR**

Specifies that variable-length database data types are to be declared as fixed length character fields and are accessible in the RPG/400 program.

***GRAPHIC**

Specifies that double-byte character set (DBCS) graphic data types are to be declared as fixed length character fields and are accessible in the RPG/400 program.

Note: Choose both of the parameters *VARCHAR and *GRAPHIC if you want variable-length DBCS graphic data types to be declared in your program.

SAAFLAG

Specifies if there will be flagging of specifications not supported by SAA RPG. For more information on SAA flagging, how and why to use it, see “Systems Application Architecture Flagging Messages” on page 49.

***NOFLAG**

No flagging will be performed.

***FLAG**

Flagging will be performed. Messages will be listed under the heading of SAA Message Summary. No SAA message will be issued for a specification if a message of severity 30 or above is issued for that specification.

PRTFILE

Specifies the name of the file where the compiler listing is to be placed, and the library where the file is located. If you specify a file whose record length is less than 132, information will be lost.

QSYSPRT

If a file name is not specified, the compiler listing is placed in the IBM-supplied file, QSYSPRT. The file QSYSPRT has a record length of 132.

file-name

Enter the name of the file where the compiler listing is to be placed.

***LIBL**

The system searches the library list to find the library where the file is located.

***CURLIB**

The current library will be used to find the file. If you have not specified a current library, QGPL will be used.

library-name

Enter the name of the library where the file is located.

REPLACE

Specifies whether or not a new program object is to be created if a program with the same name already exists in the specified library.

***YES**

A new program object will be created and any existing program object of the same name in the specified library will be moved to library QRPLOBJ.

CRTRPGPGM Command

***NO**

A new program object will not be created if a program object of the same name already exists in the specified library.

TGTRLS

Specifies the release level of the operating system on which you intend to use the object being created.

You can specify an exact release level in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level.

Note: To use the object on the target system, you must save the object to the target release level specified on the create command and then restore it on the target system.

***CURRENT**

The object is to be used on the release of the operating system currently running on your system.

***PRV**

The object is to be used on the previous release with modification level 0 of the operating system.

release-level

Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release.

USRPRF

Specifies the user profile the compiled RPG/400 program runs under. This profile controls which objects can be used by the program (including what authority the program has for each object). If the program already exists, the USRPRF parameter will not be updated to a new profile.

***USER**

The program runs under the user profile of the program's user.

***OWNER**

The program runs under the user profiles of both the program's owner and user. The collective sets of object authority in both user profiles are used to find and access objects while the program is running. Any objects that are created during the program are owned by the program's user.

Note: The USRPRF parameter reflects the security requirements of your system. The security facilities available on the AS/400 system are described in detail in the *Security Reference* and the *CL Reference*.

AUT

Specifies the authority given to users who do not have specific authority to the object, who are not on the authorization list, and whose user group has no specific authority to the object. The authority can be altered for all or for specified users after program creation with the CL commands Grant Object Authority (GRTOBJAUT) or Revoke Object Authority (RVKOBJAUT). For further information on these commands, see the *CL Reference*.

*LIBCRTAUT

The public authority for the object is taken from the CRTAUT keyword of the target library (the library that contains the object). The value is determined when the object is created. If the CRTAUT value for the library changes after the create, the new value will not affect any existing objects.

*CHANGE

The public has object operational authority and all the data authorities for the compiled program. Any user can run, debug, change and perform basic functions on the program.

*USE

The public can run the compiled program, but cannot debug or change it.

*ALL

The public has complete authority for the program.

*EXCLUDE

The public cannot use the program.

authorization-list name

Name of an authorization list to which the program is added. For a description of the authorization list and how to create it, see the *CL Reference*.

Note: Use the AUT parameter to reflect the security requirements of your system. The security facilities available are described in detail in the *Security Reference* manual.

PHSTRC

Specifies if information about compiler phases is provided on the listing. See

CRTRPGPGM Command

Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NO

Do not provide information about compiler phases.

*YES

Provide information about compiler phases.

ITDUMP

This parameter specifies if a dynamic listing of intermediate text for one or more specified phases is to be printed at compile time as each IT record is being built. This parameter also specifies if a flow of the major routine runs in one or more specified phases is to be printed. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NONE

No intermediate text dump is produced.

phase-name

Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

SNPDUMP

Specifies if the major data areas are to be printed after the running of one or more specified phases. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NONE

No snap dump is produced.

phase-name

Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

CODELIST

Specifies if a dynamic listing of the IRP is to be printed during compilation of

one or more specified phases of the source program. See Appendix A, “RPG Compiler and Auto Report Program Service Information” for a detailed explanation of this parameter.

***NONE**

Do not produce a code listing for each of the code generating phases run.

***ALL**

Produce a code listing for each of the code generating phases run.

phase-name

Enter the last two characters of phase name. See Appendix A, “RPG Compiler and Auto Report Program Service Information” for a detailed explanation of this parameter.

The third set of prompt screens shown in Figure 21 provides more values and keywords that you can enter for the CRTRPGPGM command.

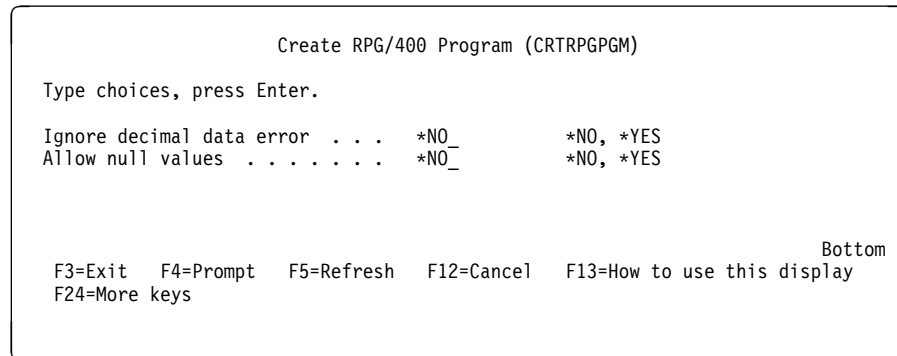


Figure 21 (Part 1 of 2). Third Set of CRTRPGPGM Prompt Screens

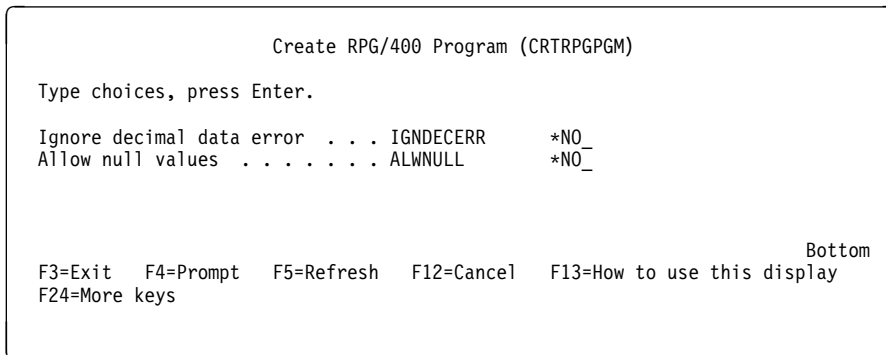


Figure 21 (Part 2 of 2). Third Set of CRTRPGPM Prompt Screens

IGNDECERR

Specifies if decimal data errors detected by the system are ignored by the program.

*NO

Do not ignore decimal data errors. When a numeric operation is attempted on a numeric field that contains decimal data that is not valid, a program exception is raised. Decimal data errors will be detected only for fields defined in packed decimal format. For more information on packed decimal format, see Chapter 11, “Communicating with Objects in the System” on page 255.

*YES

Ignore decimal data errors. The effect of decimal data errors on processing is not readily predicted. The compiler only generates an error message on the compiler listing to notify the user that this option was specified. When this option is specified, incorrect results that occur while a program is running are the user’s responsibility.

ALWNULL

Specifies whether an RPG/400 program will accept null values from null-capable fields in an externally described input-only file. A severity 0, compile-time message is issued when a record format contains null-capable fields. See “Null Value Support” on page 252 for a detailed description of how the RPG/400 compiler supports null-capable fields.

*NO

Specifies that the RPG/400 program will not process null value fields from externally-described files. If you attempt to retrieve a record containing null values, no data in the record is accessible to the RPG/400 program and a data-mapping error occurs.

*YES

Specifies that an RPG/400 program will accept null value fields for an externally-described input-only file. When a record containing null values is retrieved from an externally-described input-only file, no data mapping

errors occur and the database default values are placed into any fields which contain null values.

Compiling under the System/38 Environment

You can also compile an RPG/400 source program from the System/38 environment. You call the compiler with the same commands as you use in the AS/400 system environment (CRTRPGPGM to call up the RPG/400 compiler, and CRTRPTPGM to call up the automatic report function). To compile a program from the System/38 environment, use the CL command CALL QCL to call up the System/38 environment before you enter the CRTRPGPGM command. You can also enter System/38 environment commands from the native environment by library qualifying commands. The QSYS38/CRTRPGPGM command calls the System/38 environment RPG III compiler.

For more information on the differences between the RPG/400 program in the AS/400 environment and in the System/38 environment, see Appendix E, "System/38 Environment Option of the RPG Compiler."

For further information about programming in the System/38 environment, refer to the *System/38 RPG III Reference Manual and Programmer's Guide*.

Compiling under the System/38 Environment

Chapter 4. Error Messages, Testing, and Debugging

This chapter describes error messages you may receive from RPG/400 compiler, explains their meaning, and how you can display and print them. This chapter also describes testing and debugging an RPG/400 program using functions provided by the RPG/400 compiler and OS/400 system.

OS/400 System	RPG
<ul style="list-style-type: none"> • Test library • Breakpoints • Traces 	<ul style="list-style-type: none"> • DEBUG operation code • DUMP operation code

The OS/400 system functions allow you to use CL commands to test programs while protecting your production files, and let you observe and debug operations as a program runs. See the *CL Reference* for more information on using CL commands.

No special source code is required to use the OS/400 system functions. The RPG/400 compiler functions can be used independently of the OS/400 system functions or in combination with them either to:

- Debug a program
- Produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables.

Special source code is required to use the RPG/400 DEBUG and DUMP operation codes. You can also obtain a formatted dump in response to a run-time message.

A file information data structure and a program status data structure can provide additional debugging information. These data structures are described later in this chapter. Following this is information on exception/error handling.

Using, Displaying, and Printing Messages

Using Messages

This manual refers to the messages you receive during compilation and run-time. These messages are displayed on your screen or printed on your compiler listing. This product has no message manuals.

Using, Displaying, and Printing Messages

Each message contains a minimum of three parts as illustrated in the following sample message:

A 10
B **Message:** Syntax of Program-Identification entry is not valid. Defaults to RPGOBJ.
C **Cause:** The Program-Identification entry (positions 75-80) of a control specification has a not valid syntax: the first character is not alphabetic or it is not left-justified, or it contains embedded blanks or special characters. Defaults to RPGOBJ.
Recovery: Specify RPGOBJ or a valid entry (positions 75-80) for the Program-Identification option. Recompile.

A A number indicating the severity of the error. The severity-level value of the RPG/400 compile-time messages does not exceed 50.

Severity Meaning

- | | |
|-----------|--|
| 00 | An informational message displayed during entering, compiling, and running. No error has been detected and no corrective action is necessary. |
| 10 | A warning message displayed during entering, compiling, and running. This level indicates that an error is detected but is not serious enough to interfere with the running of the program. The results of the operation are assumed to be successful. |
| 20 | An error message displayed during compiling. This level indicates an error, but the compiler is attempting a recovery that might yield the desired code. The program may not work as the author intends. |
| 30 | A severe error message displayed during compiling. This level indicates that an error too severe for automatic recovery is detected. Compilation is complete, but the program does not run. |
| 40 | An abnormal end-of-program or function message displayed during compiling or running. This level indicates an error that forces cancelation of processing. The operation ended either because it could not handle valid data, or because the user canceled it. |
| 50 | An abnormal end-of-job message displayed during compiling or running. This level indicates an error that forces cancelation of job. The job ended either because a function failed to perform as required, or because the user canceled it. |
| 99 | A user action to be taken during running. This level indicates that some manual action is required of the operator, such as entering a reply, changing diskettes, or changing printer forms. |

B The text you see online or on a listing. This text is a brief, generally one-sentence, description of the problem.

C This text is printed on your listing if you specify *SECLVL in your compile-time options. It contains an expanded description of the message and a section detailing the correct user response. The IBM-supplied default for this option is *NOSECLVL.

At run time, you can enter D to obtain an RPG/400 formatted dump, S to obtain system dump, C to cancel, G to continue processing at *GETIN, or F to obtain a RPG/400 full-formatted dump.

Systems Application Architecture Flagging Messages

In addition to the messages described above, the RPG/400 compiler also has a set of messages that flag those RPG/400 compiler features not supported by SAA RPG. These messages are requested with a compiler option, SAAFLAG, described in “CRTRPGPGM Command” on page 29. The default value for this option is *NOFLAG. If you select *FLAG, these messages are printed separately under the heading SAA Message Summary.

The SAA flagging messages are to help the programmer when writing portable code. If you are seeking maximum portability, you should eliminate the flagged codes from your program. A program that has only SAA messages will compile and run correctly on the AS/400 system. SAA messages are informational messages only. Severe error messages may suppress SAA messages.

SAA messages are divided in the same way as the other messages described here. A sample message is:

```
A 0
B Message: SAA RPG does not support numeric fields with more than 15 digits.
C Cause: Systems Application Architecture
Common Programming
Interface RPG does not support numeric fields with more than 15 digits.
Recovery: If SAA RPG adherence is required, change the program
and recompile.
```

These messages flag RPG/400 compiler specific functions only.

Displaying and Printing Messages

To display or print particular messages, use the DSPMSGF or DSPMSGD commands. The compile-time messages are stored in a file called QRPMSG in library QRP. The run-time messages are stored in a file called QRPMSG in library QSYS.

In the System/38 environment, all the compile-time messages are in file QRP3MSG in library QRP38. The run-time messages are in file QRP3MSG in library QSYS.

Note: If you have any comments or suggestions concerning the messages, please use the Reader Comment Form included with this manual to send them to us.

How to Run an RPG/400 Program

There are many ways to run an RPG/400 program, depending on how the program is written and who is using the program. See the *CL Programmer's Guide* for the various ways to run an RPG/400 program. The three most common ways of running an RPG/400 program are through:

- A high-level language CALL statement or operation

How to Run an RPG/400 Program

- An application-oriented menu
- A user-created command.

The CL statement CALL can be part of a batch job, be entered interactively by a work station user, or be included in a CL program. An example is CALL PAYROLL. PAYROLL is the name of either a CL program or an RPG/400 program that is called and then run. An RPG/400 program can call another program with the CALL operation code. See Chapter 11, “Communicating with Objects in the System.”

Another way to run an RPG/400 program is through an application-oriented menu. You can request an application-oriented menu and then select an option that will call the appropriate program. Figure 22 is an example of an application-oriented menu:

```
                                PAYROLL DEPARTMENT MENU

1.  Inquire into employee master
2.  Change employee master
3.  Add new employee
4.  Return

Option: __
```

Figure 22. Example of an Application-Oriented Menu

This menu is normally written as a CL program where each option calls a separate RPG/400 program. When an RPG/400 program ends, the system returns control to the calling program or to the user. This could be a work station user, a CL program (such as the menu handling program), or another RPG/400 program.

You can also create a command yourself to run an RPG/400 program by using a command definition. See the *CL Programmer's Guide* for a description of how to define a command. For example, you can create a PAY command that calls a PAYROLL program. A user-created command can be entered into a batch job, or it can be entered interactively by a workstation user.

Save-While-Active Support

Application programs that change objects or data may run while the objects or data are being saved. Refer to the *Advanced Backup and Recovery Guide* for possible programming considerations related to save-while-active support.

Using a Test Library

The basic concept of testing and debugging is that of a separate testing environment. Programs running in a normal operating environment or in a test environment can read, update, and write records that are in either test or production libraries. To prevent database files in production libraries from being changed unintentionally, you can specify the UPDPROD(*N0) option on the CL command STRDBG (Start Debug).

On the AS/400 system, you can copy production files into the test library or you can create special files for testing in this library. A test copy of a file and its production copy can have the same name if the files are in different libraries. You can use the same file name in the program for either testing or normal processing.

Figure 23 shows an example of using a separate test environment.

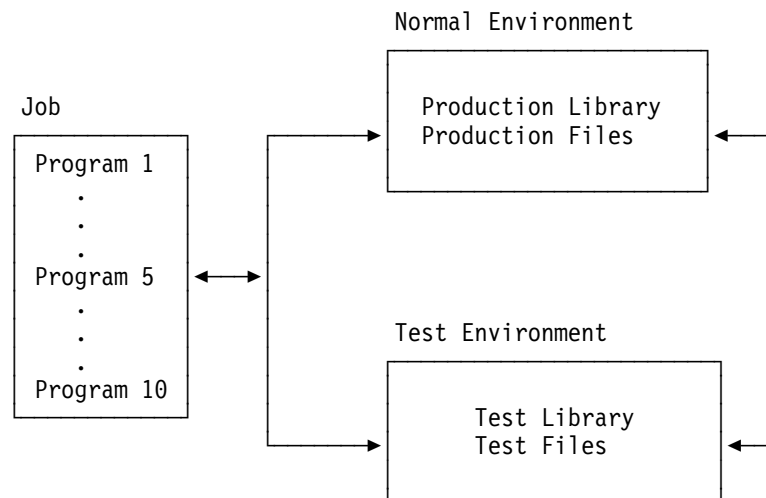


Figure 23. Using a Separate Test Environment

Using a Test Library

For testing, you must place the test library name ahead of the production library name in the library list for the job that contains the program to be tested as shown in Figure 24.

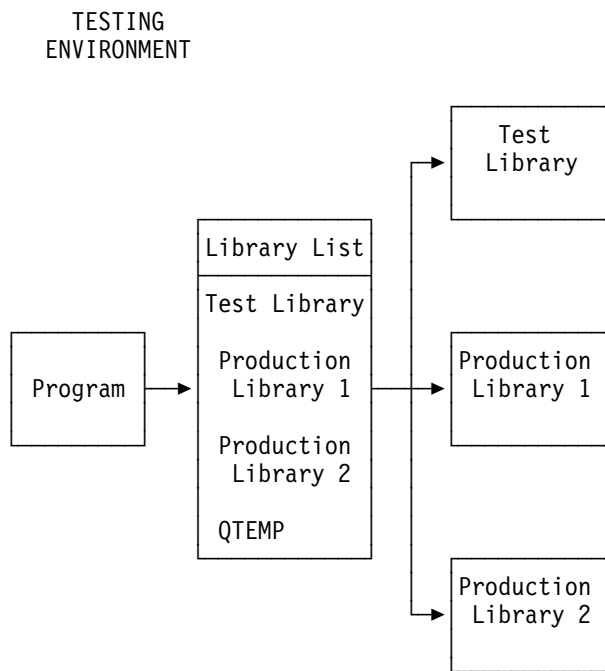


Figure 24. Testing Environment

For normal program running, the production library should be the only library named in the library list for that job. (That is, the test library should not be named.) See Figure 25 below.

NORMAL ENVIRONMENT

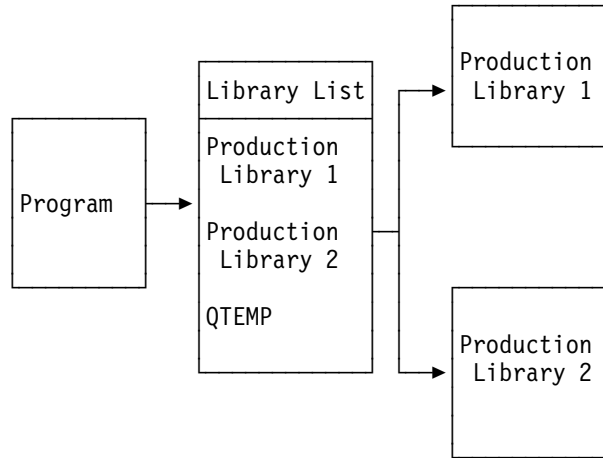


Figure 25. Normal Environment

No special statements for testing are contained within the program being tested. The same program being tested can be run normally without modifications. All testing functions are specified within the job that contains the program and not within the program.

JOB

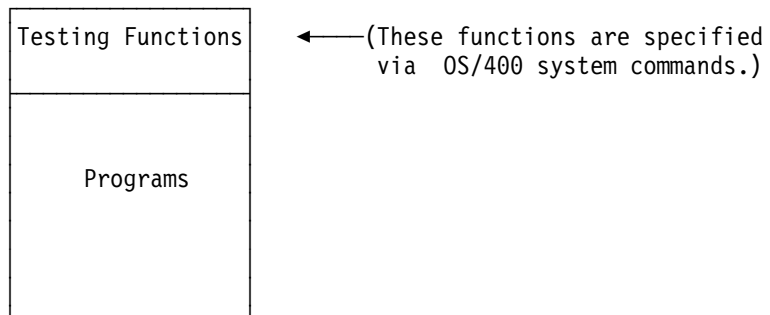


Figure 26. Testing Functions

Using Breakpoints

Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another job that is in a normal processing environment.

The OS/400 system testing functions let you interact with a program while it is running so as to observe its processing. These functions include using breakpoints and traces.

Using Breakpoints

You can use breakpoints to stop your program at a specified point. A breakpoint can be a statement number or a label in your program. If you use a label as a breakpoint rather than a statement number, the label can be:

- On a TAG statement in the program
- Associated with a step in the RPG/400 program cycle. For example, *T0TC indicates the beginning of total calculations, and *T0TL indicates the beginning of total output.
- Associated with a function done by your RPG/400 program. For example, SQRT indicates the square root function.

When a breakpoint is encountered in an interactive job, the system displays the breakpoint at which the program stops and, if requested, the values of program variables. After getting this information (displayed), you can go to a Command Entry Screen and enter CL commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace).

When a breakpoint is encountered in a batch job, a breakpoint program can be called. You must create this breakpoint program to handle the breakpoint information.

Example of Using Breakpoints

Figure 27 shows a source listing of a sample RPG/400 program, DBGPGM, and the CL commands that add breakpoints at statements 1200 and 1500. The value of variable *IN is displayed when the breakpoint at statement 1200 is reached, and the value of variables FLD1 and PART are displayed when the breakpoint at statement 1500 is reached.

CL Commands

```
STRDBG PGM(EXAMPLES/DBGPGM)
ADDBKP STMT(1200) PGMVAR((*IN))
ADDBKP STMT(1500) PGMVAR((FLD1) (PART)) OUTFM(*HEX)
```



```

5763RG1 V3R0M5 940125          IBM RPG/400          QGPL/DBGPGM          01/25/94 13:42:19      Page 2
SEQUENCE                      IND DO LAST PAGE PROGRAM
NUMBER *...1...+...2...+...3...+...4...+...5...+...6...+...7...* USE NUM UPDATE LINE ID

          S o u r c e   L i s t i n g

H
100 FTESTX IF F 5          DISK          01/01/94
200 FTESTA UF F 10        DISK          01/01/94
300 ITESTX NS 01          01/01/94
400 I                      1 5 PART     01/01/94
500 ITESTA NS 02          01/01/94
600 I                      1 5 FLD1    01/01/94
700 *****
800 *      MAINLINE      01/01/94
900 *****
1000 C          LOOP     TAG          01/01/94
1100 C          READ TESTX          66          3          01/01/94
1200 C 66          GOTO ENDPGM          01/01/94
1300 C          READ TESTA          67          3          01/01/94
1400 C N67          MOVE PART      FLD1          01/01/94
1500 C N67          EXCPTMAST          01/01/94
1600 C N66          GOTO LOOP          01/01/94
1700 C          ENDPGM   TAG          01/01/94
1800 C          SETON          LR          1          01/01/94
1900 OTESTA E          MAST          01/01/94
2000 O          FLD1      5          01/01/94

          * * * * *   E N D   O F   S O U R C E   * * * * *

```

Figure 27. Sample RPG/400 Program DBGPGM

The first breakpoint shows you where you are in the program. Figure 28 shows the two displays as a result of reaching the first breakpoint.

Using Breakpoints

```
Display Breakpoint
Statement/Instruction . . . . . : 1200 /004A
Program . . . . . : DBGPGM
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *CHAR
Length . . . . . : *DCL

Variable . . . . . : *IN
Lower/upper bounds . . . . . : (1:99)
Type . . . . . : CHARACTER
Length . . . . . : 1
Element ----- Values .
  1  '1' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 11  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 21  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 31  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'

Press Enter to continue.
F3=Exit program  F10=Command entry

More...
```

```
Display Breakpoint
Statement/Instruction . . . . . : 1200 /004A
Program . . . . . : DBGPGM
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *CHAR
Length . . . . . : *DCL

41  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
51  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
61  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
71  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
81  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
91  '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'

Press Enter to continue.
F3=Exit program  F10=Command entry
```

Figure 28. First Breakpoint Display for DBGPGM

Figure 29 shows the two displays as a result of reaching the second breakpoint.

```

                                Display Breakpoint
Statement/Instruction . . . . . : 1500 /0060
Program . . . . . : DBGPGM
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *HEX
Length . . . . . : *DCL

Variable . . . . . : FLD1
Type . . . . . : CHARACTER
Length . . . . . : 5
* . . . +. . . . 1 . . . + . . . *...+...1...+.
404000063F

Variable . . . . . : PART
Type . . . . . : CHARACTER
Length . . . . . : 5

Press Enter to continue.
F3=Exit program  F10=Command entry
More...

```

```

                                Display Breakpoint
Statement/Instruction . . . . . : 1500 /0060
Program . . . . . : DBGPGM
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *HEX
Length . . . . . : *DCL

* . . . +. . . . 1 . . . + . . . *...+...1...+.
404000063F

Press Enter to continue.
F3=Exit program  F10=Command entry

```

Figure 29. Second Breakpoint Display for DBGPGM

At this point, you can change the value of one of these variables to alter how your program runs. After getting to the Command Entry Screen by pressing F10, you can use the CL command CHGPGMVAR (Change Program Variable) to change the value of a variable.

Considerations for Using Breakpoints

You should know the following characteristics of breakpoints before using them:

- If a breakpoint is part of a conditional statement, that breakpoint request is processed even if the condition is not met.

Using a Trace

- If a breakpoint is bypassed by a GOTO operation, that breakpoint request is not processed.
- Some statements that are not processed do not represent a definite position in the logic flow of your program. Avoid putting breakpoints on PLIST, PARM, KLIST, KFLD, and DEFN operations.
- When a breakpoint is requested for a statement, the breakpoint occurs before that statement is run.
- When a breakpoint is requested for a statement that is not processed, such as a TAG operation, the breakpoint is set on the next statement.
- Breakpoint functions are specified using CL commands. You can use CL commands to add breakpoints to programs, display breakpoint information, remove breakpoints from programs and start a program after a breakpoint has been displayed. Refer to the *CL Reference* for descriptions of these commands and for a further description of breakpoints.
- Input fields not used in your program cannot be specified in the PGMVAR parameter of the debug commands. You can display the entire input or output buffer for a record by using the variable name ZZnnBIN (input buffer) or ZZnnBOUT (output buffer). The nn value is the sequence number corresponding to the order in which the files are defined in your specifications. This number also appears in the cross reference section of the compiler listing. Thus you can display the input buffer for the second file in your program by specifying PGMVAR (ZZ02BIN).

Using a Trace

You can use a trace to record the statements that are run in a program and the values of the variables used in the statements.

To use a trace, you specify what statements and variables the system should trace. You can also specify that variables be traced only when their values change. You can specify a trace of one statement, a group of statements, or an entire program. You must request a display of the traced information. The display shows the sequence in which the statements were run and, if requested, the values of variables used in the statements. Figure 30 on page 59 shows the setup of a trace for program statements and their order of processing.

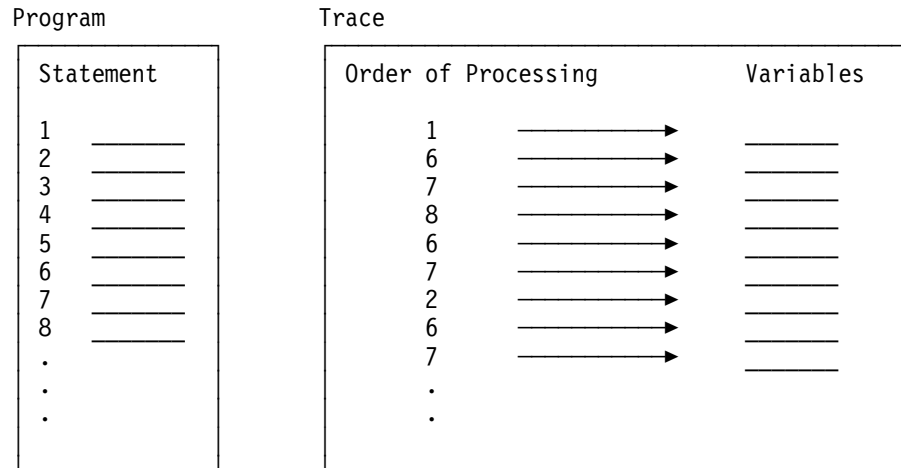


Figure 30. Program Statements and Order of Processing

Example of Using a Trace

Figure 27 on page 55 shows a portion of a listing of RPG/400 program DBGPGM. The CL command that adds a trace of statements 1000 through 1800 in that program is:

```
ADDTRC STMT((1000 1800))
```

Figure 31 is an example of a display of the traced information. The CL command to display this information is:

```
DSPTRCDTA OUTPUT(*)
```

Display Trace Data			
Program	Statement/ Instruction	Recursion Level	Sequence Number
DBGPGM	1000	1	1
DBGPGM	1200	1	2
DBGPGM	1300	1	3
DBGPGM	1400	1	4
DBGPGM	1500	1	5
DBGPGM	1600	1	6
DBGPGM	1000	1	7
DBGPGM	1200	1	8
DBGPGM	1800	1	9

Press Enter to continue.
F3=Exit F12=Cancel

Figure 31. Trace Data Display for DBGPGM

Considerations for Using a Trace

You should know the following characteristics of traces before using them:

- A conditional statement is recorded in the trace even if the condition is not met.
- Statements bypassed by GOTO operations are not included in the trace.
- Trace functions are specified with CL commands in the job that contains the traced program. These functions include adding trace requests to a program, removing trace requests from a program, removing data collected from previous traces, displaying trace information, and displaying the traces that have been specified for a program.
- You cannot display a variable that is not referenced in your RPG/400 program.

Using the DEBUG Operation Codes

You can code one or more DEBUG operation codes among your RPG/400 calculations to help you debug a program that is not working properly. Whenever the DEBUG operation is processed, one or two records with debugging information are provided. The first record contains a list of all indicators that are set on at the time the DEBUG operation was encountered. The second record is optional and shows the contents of the result files specified for the DEBUG operation.

The DEBUG operation can be coded at any point or at several points in the calculation specifications. The output records are written whenever the DEBUG operation occurs.

You should know the following characteristics of the DEBUG operation code before using it:

- The DEBUG operation runs (are active) only if position 15 of the control specification contains a 1.
- If the DEBUG operation is conditioned, it occurs only if the condition is met.
- If a DEBUG operation is bypassed by a GOTO operation, the DEBUG operation does not occur.

You can apply the OS/400 system testing and debugging functions to programs that use DEBUG operations; a breakpoint can be on a DEBUG operation, and a DEBUG operation can be traced.

Using the RPG/400 Formatted Dump

To obtain an RPG/400 formatted dump (printout of storage) for a program while it is running, you can code one or more DUMP operation codes in your calculations, or you can respond to a run-time message with a D or F option. It is also possible to automatically reply to make a dump available. Refer to the "System Reply List" discussion in the *CL Programmer's Guide*.

The formatted dump includes field contents, data structure contents, array and table contents, the file information data structure, and the program status data structure. The dump is written to the file called QPPGMDMP. (A system abnormal dump is written to the file QPSRVDMP.)

If you respond to an RPG/400 run-time message with an F option, the dump also includes the hexadecimal representation of the open data path (ODP, a data management control block). If position 15 of the control specification contains a 1, the F option also provides a list of compiler-generated fields.

Information from the file information data structure (INFDS) is provided for each file in the program. Not all the information that is contained in the INFDS is printed in the dump. Remember that, to use any information from the INFDS in your program, you must define the INFDS in your program.

The same characteristics as described for the DEBUG operation apply to the DUMP operation.

Figure 32 shows an example of an RPG/400 formatted dump.

Note

Only selected pages of an RPG/400 formatted dump are presented below.

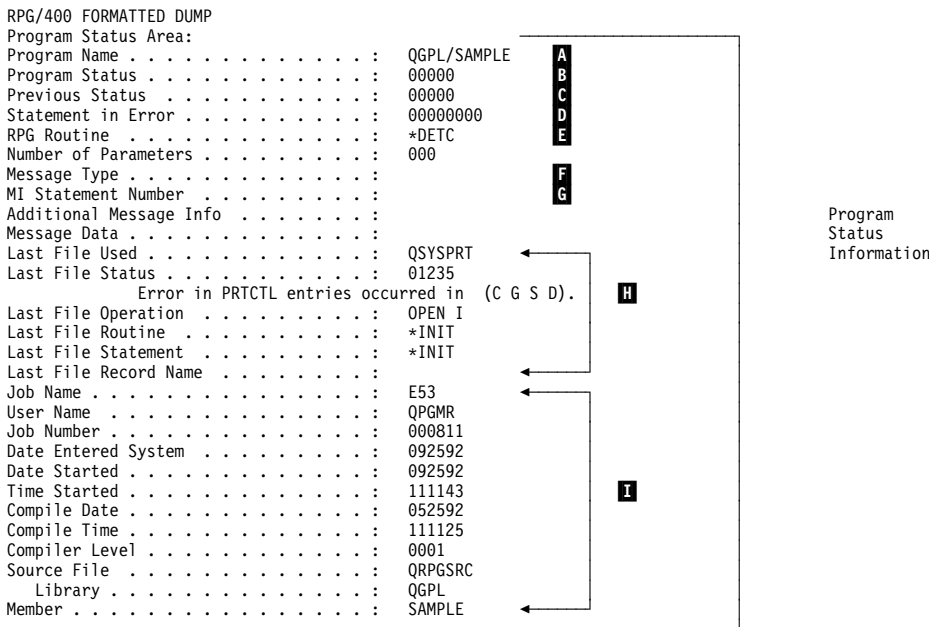


Figure 32 (Part 1 of 8). RPG/400 Formatted Dump

Using the RPG/400 Formatted Dump

RPG/400 FORMATTED DUMP

```

File . . . . . : FILEIN1
File Open . . . . . : YES
File at EOF . . . . . : YES
Commit Active . . . . . : NO
File Status . . . . . : 00011
                    RPG0011 End of file (input).
File Operation . . . . . : READ R
File Routine . . . . . : *DETC
Statement Number . . . . . : 2500
Record Name . . . . . : FILEA
Message Identifier . . . . . :
MI Instruction Number . . . . . :
ODP type . . . . . : DB
File Name . . . . . : FILEIN1
Library . . . . . : QGPL
Member . . . . . : FILEIN1
Record Format . . . . . :
Primary Record Length . . . . . : 45
Secondary Record Length . . . . . : 0
Input Block Length . . . . . : 4125
Output Block Length . . . . . : 0
Device Class . . . . . : '0000'X
Lines per Page . . . . . : 0
Columns per Line . . . . . : 0
Number of Records in File . . . . . : 0
Access Type . . . . . : ARRIVAL SEQ
Allow Duplicate Keys . . . . . : NO
Source File . . . . . : NO
UFCB Parameters . . . . . : 'A2000000000000500000'X
UFCB Overrides . . . . . : '00000000000000000000'X
Records to Transfer . . . . . : 74
Number of Puts . . . . . : 0
Number of Gets . . . . . : 0
Number of Put/Gets . . . . . : 0
Number of other I/O . . . . . : 0
Current Operation . . . . . : '4040'X
Device Class . . . . . : '4040'X
Device Name . . . . . :
Length of Last Record . . . . . : 0
DDS Information . . . . . :
Relative Record Number . . . . . : 0
Records Transferred . . . . . : 0
Current Line Number . . . . . : 0
Input Buffer:
0000  80000000  00000000  0007C00D  BD000880  0000004A  0037002D  40404040  40404040  *
0020  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0060  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0080  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
00A0  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
00C0  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
00E0  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0100  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0120  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0140  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *
0160  40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040  *

```

Figure 32 (Part 2 of 8). RPG/400 Formatted Dump

RPG/400 FORMATTED DUMP

Open Data Path:

0000	64800000	000010A4	00001100	00000000	00000140	000001C6	00000280	000002C0	*		F	*	
0020	00000530	00000000	00000000	00000140	00000000	00000000	00000000	00000000	*			*	
0040	00000000	00000016	0007C00D	BA0019FF	00000000	00000000	00000000	00000000	*			*	
0060	80000000	00000A80	0007C001	DB001710	00000000	00000000	00000000	00000000	*			*	
0080	80000000	00000000	0007C00D	BA001120	01900000	00010000	00000084	00000000	*			*	
00A0	08000000	00000000	00000000	00100000	E2D7D8E2	E8E2D7D9	E3404040	D8E2E8E2	*		SPQSYSVRT	QSYS*	
00C0	40404040	4040D8F0	F4F0F7F9	D5F0F0F1	D8E2D7D3	40404040	40400013	00840000	*	Q04079N001QSPL		*	
00E0	D8F7F1F3	F7F8F4F7	F0F10000	00000000	00840002	00000000	0A008400	00000000	*	*Q713784701		*	
0100	0000D5A4	12100000	00000000	00000000	00000000	00000000	00000100	09000000	*	N		*	
0120	0005A000	5CD54040	40404040	40400001	00000000	00000000	00000000	00000000	*	N		*	
0140	00010001	5CD54040	40404040	40400000	06700000	00000000	00450045	00450045	*	N		*	
0160	07A10045	00450045	00700045	00450045	00450045	00450045	002F0030	00040005	*			*	
0180	5CD54040	40404040	40400208	00000000	20000000	00000000	00000000	00000000	*	N		*	
01A0	00000000	00000001	C2200000	00059A00	00000000	00000000	00000000	00000000	*		B	*	
01C0	00000000	00000009	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
01E0	00000000	02080000	00000000	00000000	00000084	00000000	00000000	00000000	*			*	
0200	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
0220	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
0240	00000000	00000000	00000000	00000000	00000000	00000001	00000001	00000000	*			*	
0260	00000000	00000000	00000000	00000000	00000000	00000000	F0F0F0F0	00000000	*		0000	*	
0280	00000001	00300000	00000000	00003000	00000000	0000001C	000502B5	D90019FF	*		R	*	
02A0	00000000	00000000	00000000	00000000	41100000	00000000	00000000	00000000	*			*	
02C0	80000000	0000004F	0007C00D	BB001824	80000000	00000000	0007C00D	BB000860	*			*	
02E0	80000000	00000000	0007C00D	BB000860	80000000	00000000	0007C00D	BB001838	*			*	
0300	00000000	00000700	000502DB	2700195F	00000000	00000000	0FEF0000	30700000	*			*	
0320	0FCC00D6	00E00000	0001DEC0	00000000	00000000	00000000	00000000	00010000	*	O		*	
0340	000186A0	C6000000	5CD1D6C2	40404040	40404040	40404040	40404040	5CE2E3C4	*	F	JOB	STD*	
0360	40404040	40400000	000001F0	80000000	000081F0	0007C00D	BB0003B0	BB0003B0	*		0	0	*
0380	80000000	00000000	0007C00D	BB000860	80000000	00000000	0007C00D	BB000860	*			*	
03A0	80000000	00000000	0007C00D	BB000460	80000000	00000000	0007C00D	BB000576	*			*	
03C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
03E0	00000000	00000000	00000000	00000000	D8F0F4F0	F7F9D5F0	F0F10048	D8E2D7D3	*		Q04079N001	QSPL*	
0400	40404040	40400049	00000000	00000000	00004040	40404040	40401010	00010100	*			*	
0420	F0F1F0F0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*	*0100		*	
0440	00010FEF	7FFF0001	00840004	00010001	D8E2E8E2	D7D9E340	4040FFF5	00000000	*		QSYSVRT	5	*
0460	00000000	00000000	00000000	00000000	FFF30000	00000000	00000000	00000000	*		3	*	
0480	00000000	0000000C	C0000A00	84000E00	06000F00	09001000	00110000	12C04000	*			*	
04A0	13007FFF	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
04C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
04E0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
0500	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
0520	00000000	00000000	00000000	00000000	006A0800	00000000	00000650	00000000	*			*	
0540	00000000	0000000C	000002C0	00005C00	0002FFFF	00010001	00015CC3	C8C1D5C7	*			CHANG*	
0560	C5404040	00000000	00010000	00000000	00000000	00000000	00000000	00000000	*	E		*	
0580	00000080	00000000	00000000	00000000	D8E2E8E2	D7D9E340	404000AE	0601000A	*		QSYSVRT	*	
05A0	00840009	0D405CC4	C5E5C440	40404040	40404040	40404040	40405CD7	D9E3C9D4	*	DEVD		PRTIM*	
05C0	C7404040	40404040	40404040	4040060A	060F100F	00000000	00000000	00000000	*	G		*	
05E0	00000000	009E0000	01001000	00C1D7D7	D3D7C7D4	40000000	00000000	00000000	*		APPLPGM	*	
0600	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	
0620	00000000	00000000	00000000	00000000	00000000	00000000	00100000	00015CC3	*			C*	
0640	D7C94040	40404040	00000000	00000000	A4121000	00000000	00000000	40000000	*	PI		*	
0660	00000000	00000000	00000000	00000000	80000000	00000018	0007C00D	BB000860	*			*	
0680	80000000	000001F0	00060382	3B000562	80000000	00000000	00060382	3B000A72	*	0		*	
06A0	80000000	00000000	00060382	3B00015C	80000000	00000000	0007C00D	BA001000	*			*	
06C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*			*	

Figure 32 (Part 5 of 8). RPG/400 Formatted Dump

Using the RPG/400 Formatted Dump

```

RPG/400 FORMATTED DUMP
NAME      OFFSET ATTRIBUTES VALUE
..MDFDEV 000980 CHAR(10)        '
..MDFDVP 0009A0 POINTER(SPP)   '00000000000000000000'X
          SPACE OFFSET 2480          '000009B0'X
          OBJECT      PSSA
..MDFNDI 001990 CHAR(1)        '0'
.ACTPTR  001A60 POINTER(SYP)   QGPL
          CONTEXT    SAMPLE
          OBJECT      '
.ACTPTRC 001A60 CHAR(16)      '00000000000000000000703AA08000238'X
.BINF1   000610 BINARY(4)      0
.BINF2   000614 BINARY(4)      0
.BINRF   000618 BINARY(4)      0
.BLANKS  0002B2 CHAR(140)     ' '
.BPCA    0004B0 CHAR(32767)   CANNOT DUMP - SPACE ADDRESSING OR BOUNDARY ALIGNMENT EXCEPTION
.BPCAPTR 000A00 POINTER(SPP)   1200          '000004B0'X
          SPACE OFFSET
          OBJECT      FILEIN2 QGPL FILEIN2
.BUFFER  0004D0 CHAR(148)     '
          0004D0 VALUE IN HEX '404040404040404040404040404040404040404040404040404040404040404040404040404000010000'X
          0004FD +46          '000100000000004040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
          00052A +91          '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
          000557 +136         '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
.BUFPTR  000980 POINTER(SPP)   1232          '000004D0'X
          SPACE OFFSET
          OBJECT      FILEIN2 QGPL FILEIN2
.CALLERR 001BF0 POINTER(IP)
          INSTR #      14
          CONTEXT    QGPL
          OBJECT      SAMPLE
.CALLSW  001B2D CHAR(1)        '1'
.CLOSASW 001B5A CHAR(1)        '0'
.CLOSPTR 0000A0 POINTER(SYP)   QSYS
          CONTEXT    QDMCLOSE
          OBJECT
.CO01001 001980 POINTER(SPP)   6480          '00001950'X
          SPACE OFFSET
          OBJECT      PSSA
.CO02002 0019C0 POINTER(SPP)   6545          '00001991'X
          SPACE OFFSET
          OBJECT      PSSA
.CURROP  000598 CHAR(5)        'WRITE'
.DBFIND  0003A7 CHAR(2)        DIMENSION(4)
          0003A7 (1)          'OV'
          0003A9 (2)          '1V'
          0003AB (3)          '2V'
          0003AD (4)          'LR'
.DBFIX   0003A7 CHAR(8)        'OV1V2VLR'
.DBICNT  0003AF BINARY(2)     4
.DEACTSW 001CC1 CHAR(1)        '0'
.DMCBF   000190 CHAR(2)        DIMENSION(256)
          000190 (1)          ' '          '2000'X
          000214 (2)          ' '          '0000'X
          000298 (3)          ' '          '0005'X
          00031C (4)          ' '          '3070'X
          0003A0 (5)          ' '          '8000'X

```

M

Figure 32 (Part 6 of 8). RPG/400 Formatted Dump

- A** Qualified program name and library.
- B** Current status code.
- C** Previous status code.
- D** RPG/400 source statement in error.
- E** RPG/400 routine in which the exception or error occurred.
- F** CPF or MCH for a machine exception.
- G** Machine instruction number.
- H** Information about the last file used in the program before an exception or error (RPG1235) occurred.
- I** Program information.
- J** Error in the file.
- K** The number of times the RPG/400 compiler requested I/O of the system (not the number of I/O operations requested by the program).
- L** The open data path is included in the dump if the user responds to an RPG/400 run-time message with an F option.
- M** A list of compiler-generated fields is also included in the dump if the user responds to an RPG/400 run-time message with an F option and if the program was compiled with a 1 in position 15 of the control specification.
- N** General indicators 1-99 and their current status (1 is on, 0 is off).
- O** Beginning of user fields.
- P** Incorrect zoned field printed in hexadecimal.
- Q** File information data structures for FILEIN1 and FILEIN2.
- R** Double-occurrence data structure.
- S** System date values.
- T** IGNDDECERR(*N0) was specified in the CRTRPGPGM command.
- U** Program status data area.
- V** Input buffer for file 02.
- W** This is the file number. See the cross-reference section of the compiler listing for the corresponding file name. The files are assigned a sequence number corresponding to the order in which they are defined in your specifications. Thus, the file number 03 corresponds to the FILEIN2 file described in this program.
- X** Output buffer for file 03.

Exception/Error Handling

The RPG/400 compiler handles two types of exception or errors: program exception or errors and file exception or errors. Some examples of program exception or errors are division by zero, not valid array index, or SQRT of a negative number. Some examples of file exception or errors are undefined record type or a device error.

Figure 33 shows an example of a file information data structure (INFDS) and a file exception/error subroutine. For further information on exception/error handling by the RPG/400 compiler, see the *RPG/400 Reference* manual.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* Three files are defined on the file description specifications.
F* You want to control the program logic if an exception or error
F* occurs on the TRNFIL file or on the MSTFIL file. Therefore, a
F* unique INFDS and a INFSR are defined for each file. They are
F* not defined for the AUDITFIL file.
F*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTRNFIL OF E K DISK KINFDS FILDS1
F KINFSR ERRRTN
FMSTFIL UF E K DISK KINFDS FILDS2
F KINFSR MSTERR
FAUDITFILOF E K DISK
```

Figure 33 (Part 1 of 4). Example of File Exception/Error Handling


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The location of the subfields in the file information data
I* structures is defined by special keywords in positions 44
I* through 51. To access these predefined subfields, you must
I* assign a name to each subfield in positions 53 through 58.
I* If an exception or error occurs, you can test the information
I* in the data structure to determine, for example, what exception
I* or error occurred (*STATUS) and on which operation it occurred
I* (*OPCODE). You can then use that information within the file
I* exception/error subroutine to determine the action to take.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IFILDS1      DS
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          *FILE    FIL1
I          *RECORD REC1
I          *OPCODE OP1
I          *STATUS STS1
I          *ROUTINE RTN1
IFILDS2      DS
I          *FILE    FIL2
I          *RECORD REC2
I          *OPCODE OP2
I          *STATUS STS2
I          *ROUTINE RTN2

```

Figure 33 (Part 2 of 4). Example of File Exception/Error Handling

Exception/Error Handling

.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..

C*

C* On the WRITE operation to the TRNREC record in the TRNFIL file,
 C* an exception/error indicator is specified in positions 56 and 57.
 C* This indicator is set on if an exception or error occurs on this
 C* operation. The ERRRTN subroutine (the file exception or error
 C* subroutine for the TRNFIL file) is explicitly called by the EXSR
 C* operation when indicator 71 is on. Because factor 2 of the ENDSR
 C* operation for the ERRRTN is blank, control returns to the next
 C* sequential instruction following the EXSR operation after the
 C* subroutine has run.

C*

CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*

C

WRITETRREC

71

C 71

EXSR ERRRTN

C

"

Calculations

C*

C*

C* No exception/error indicator is specified in positions 56 and 57
 C* of the WRITE operation to the AUDITREC record in the AUDITFIL
 C* file. No exception/error subroutine was defined for this file
 C* on the file description specifications. Therefore, any exception/
 C* errors that occur on this operation to the AUDITFIL file are
 C* handled by the default RPG default error handler.

C*

C

WRITEAUDITREC

C

"

Calculations

C*

Figure 33 (Part 3 of 4). Example of File Exception/Error Handling

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* No exception/error indicator is specified in positions 56 and 57
C* of the CHAIN operation to the MSTREC record in the MSTFIL file.
C* However, a file exception/error subroutine (MSTERR) is defined
C* for the file on the file description specifications. Therefore,
C* when an exception or error other than no record found occurs on
C* the CHAIN operation, RPG passes control to the MSTERR subroutine.
C* On the ENDSR operation for this subroutine, factor 2 contains a
C* field name. This allows the programmer to alter the return point
C* from the subroutine within the subroutine based on the exception
C* or error that occurred. The field must contain one of the values
C* described under File Exception/Error Subroutine earlier in
C* this chapter.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C      MSTKEY      CHAINMSTREC      61
C 61      GOTO NOTFND
C      "
C      "
C      MSTERR      BEGSR
C      "
C      "
C      ENDSRRTRPNT
C      "
C      "
C      ERRRTN      BEGSR
C      "
C      "
C      ENDSR

```

Figure 33 (Part 4 of 4). Example of File Exception/Error Handling

Figure 34 on page 74 shows an example of a program exception/error subroutine.

Exception/Error Handling

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IDsnme....NODsExt-file++.....OccrLen+.....*
I          SDS
I
I          *ROUTINE LOC
I          *STATUS  ERR
I          *PARMS   PARMS
I          *PROGRAM NAME

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *PSSR      BEGSR
C          ERR        COMP 102                20 DIV BY ZERO?
C  20          ADD 1          DIVSR
C  20          MOVE '*DETC'  RETURN  6
C  N20         MOVE '*CANCL' RETURN
C          ENDSRRETURN

```

Figure 34. Example of *PSSR Subroutine

The program-status data structure is defined on the input specifications. The pre-defined subfields *STATUS, *ROUTINE, *PARMS, and *PROGRAM are specified, and names are assigned to the subfields.

The *PSSR subroutine is coded on the calculation specifications. If a program exception/error occurs, the RPG/400 compiler passes control to the *PSSR subroutine. The subroutine checks to determine if the exception or error was caused by a divide operation in which the divisor is zero. If it was, indicator 20 is set on, 1 is added to the divisor (DIVSR), and the literal '*DETC' is moved to the field RETURN. Moving the literal into the RETURN field, which is specified in factor 2 of the ENDSR operation, allows you to control the return point within the subroutine. In this example, control returns to the beginning of the detail calculations routine, unless the exception or error was not a divide by zero. In that case, the literal '*CANCL' is moved into the RETURN field, and the program is ended.

Chapter 5. General File Considerations

This chapter describes:

- The device-independent and device-dependent characteristics of the RPG/400 program on the AS/400 system
- AS/400 spooling functions
- The extent to which externally described and program-described files are defined in the RPG/400 program
- Level checking functions
- File locking by the RPG/400 program
- Record locking by the RPG/400 program
- Unblocking and blocking records to improve performance
- Sharing an open data path
- General information about the use of externally described files and how this external description can be changed in the RPG/400 program
- Program-described files
- RPG/400 functions that relate specifically to an RPG/400 PRINTER device, SEQ device, and SPECIAL device.

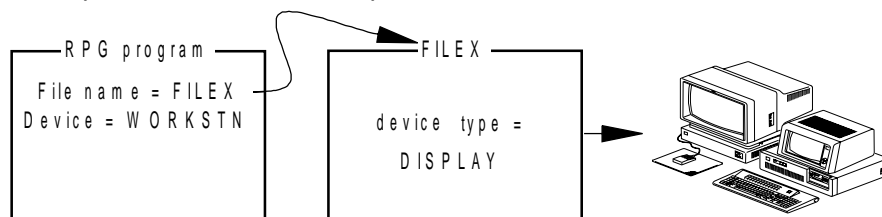
On the AS/400 system, files are made up of members. These files are organized into libraries. The convention for naming files is library-name/file-name.

Device Independence/Device Dependence

The key element for all input/output operations is the file. All files used on the system are defined to the OS/400 system. The OS/400 system maintains a description of each file that is accessed by a program when it uses the file.

The OS/400 file descriptions are kept online and serve as the connecting link between a program and the device used for I/O. The data is read from or written to the device when the file is used for processing. In some instances, this type of I/O control allows you to change the type of file (and, in some cases, change the device) used in a program without changing the program.

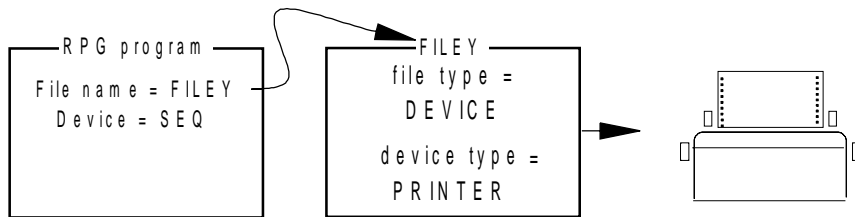
On the AS/400 system, the file name specified in positions 7 through 14 of the file description specification is used to point to the file, rather than the device name specified in positions 40 through 46. The file name points to the OS/400 file description that contains the specifications for the actual device:



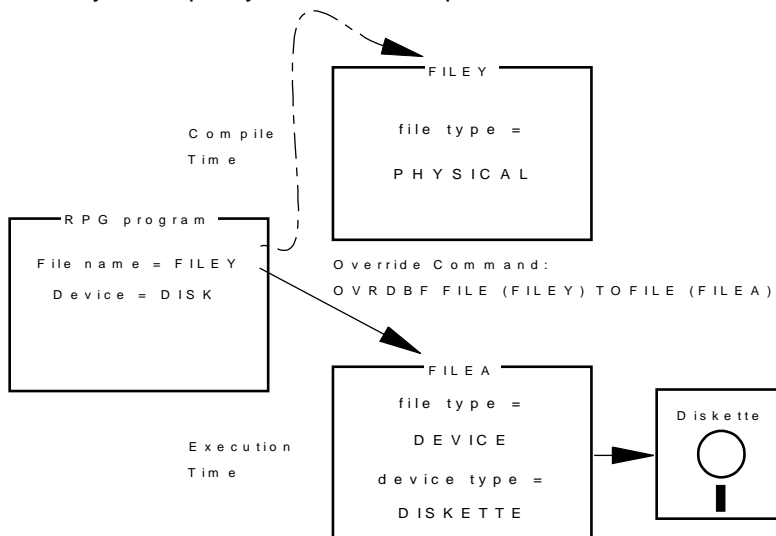
Device Independence/Device Dependence

The RPG/400 device name in positions 40 through 46 defines the RPG/400 functions that can be processed on the associated file. At compilation time, certain RPG/400 functions are valid only for a specific RPG/400 device name. In this respect, the RPG/400 function is device dependent. One example of device dependency is that the EXFMT operation code is valid only for a WORKSTN device.

For another example, assume that the file name FILEY is specified in the RPG/400 program with the SEQ device. The device SEQ is an independent device type. When the program is run, the actual I/O device is specified in the description of FILEY. For example, the device might be PRINTER.



OS/400 commands can be used to override a parameter in the specified file description or to redirect a file at compilation time or run time. File redirection allows you to specify one file at compilation time and another file at run time:



In the preceding example, the CL command OVRDBF (Override With Database File) allows the program to run with an entirely different device file than was specified at compilation time.

Not all file redirections or overrides are valid. At run time, checking ensures that the specifications within the RPG/400 program are valid for the file being processed. The OS/400 system allows some file redirections even if device specifics are contained in the program. For example, if the RPG/400 device name is PRINTER, and the actual file the program connects to is not a printer, the OS/400 system ignores the RPG/400 print spacing and skipping specifications. There are other file redirections that the OS/400 system does not allow and that cause the program to end. For example, if the RPG/400 device name is WORKSTN and the EXFMT operation is specified in the program, the program is stopped if the actual file the program connects to is not a display or ICF file.

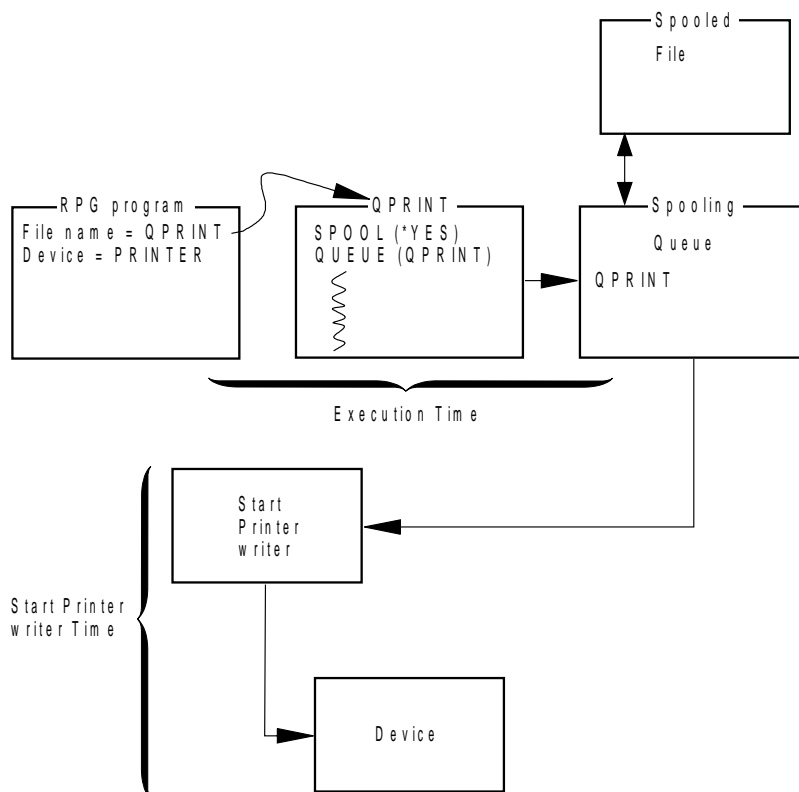
See the *Data Management Guide* for more detailed information on valid file redirections and file overrides.

Spooling

Spooling is a system function that puts data into a storage area to wait for processing. The AS/400 system provides for the use of input and output spooling functions. The RPG/400 program is not aware that spooling is being used. The actual physical device from which a file is read or to which a file is written is determined by the spool reader or the spool writer. For more detailed information on spooling, see the *Data Management Guide*.

Output Spool

Output spooling is valid for batch or interactive jobs. The description of the file that is specified in the RPG/400 program by the file name contains the specification for spooling as shown in the following diagram:



File override commands can be used at run time to override the spooling options specified in the file description, such as the number of copies to be printed. In addition, AS/400 spooling support allows you to redirect a file after the program has run. You can direct the same printed output to a different device such as a diskette.

Externally Described and Program-Described Files

All files on the AS/400 system are defined to the OS/400 system. However, the extent to which files can be defined differs:

- An *externally described file* is described to the OS/400 system at the field level. The description includes information about where the data comes from, such as the database or a specific device, and a description of each field and its attributes.
- A *program-described file* is described at the field level within the RPG/400 program on input/output specifications. The description of the file to the OS/400 system includes information about where the data comes from and the length of the records in the file.

An externally described file does not have to be redefined in an RPG/400 program on input/output specifications. In a program-described file, the fields and their attributes must be described on input/output specifications.

Externally described files offer the following advantages:

- Less coding in RPG/400 programs. If the same file is used by many programs, the fields can be defined once to the OS/400 system and used by all the programs. This practice eliminates the need to code input and output specifications for RPG/400 programs that use externally described files.
- Less maintenance activity when the file's record format is changed. You can often update programs by changing the file's record format and then recompiling the programs that use the files without changing any coding in the program.
- Improved documentation because programs using the same files use consistent record-format and field names.

If an externally described file (identified by an E in position 19) is specified for the devices SEQ or SPECIAL, the RPG/400 program uses the field descriptions for the file, but the interface to the OS/400 system is as though the file were a program-described file. Externally described files cannot specify device-dependent functions such as forms control.

You can choose to use an externally described file within the program by specifying the file as program-described (F in position 19 of the file description specifications). The compiler does not copy in the external field-level description of the file at compilation time. This approach is used in conversion where existing programs use program-described files and new programs use externally described files to refer to the same file.

Figure 35 shows some typical relationships between an RPG/400 program and files on the AS/400 system.

Externally Described and Program-Described Files

Program and Files on the AS/400 System

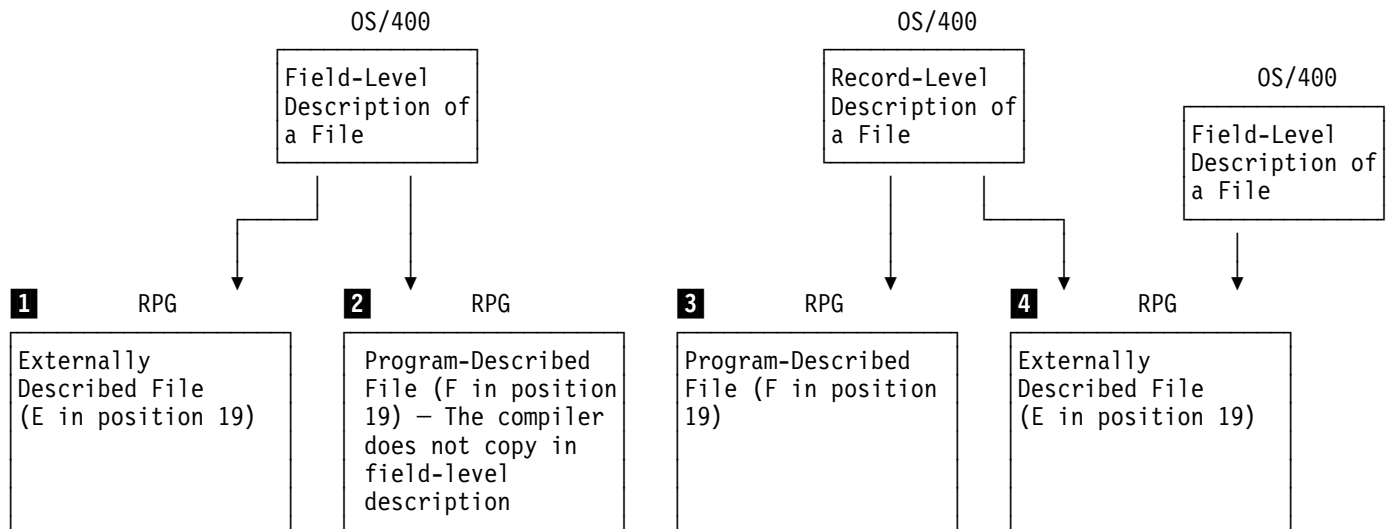


Figure 35. Typical Relationships between an RPG/400

- 1** The RPG/400 program uses the field-level description of a file that is defined to the OS/400 system. An externally described file is identified by an E in position 19 of the file description specifications. At compilation time, the compiler copies in the external field-level description.
- 2** An externally described file is used as a program-described file in the RPG/400 program. A program-described file is identified by an F in position 19 of the file description specifications. This entry tells the compiler not to copy in the external field-level descriptions. This file does not have to exist at compilation time.
- 3** A file is described only to the record level to the OS/400 system. The fields in the record are described within the RPG/400 program; therefore, position 19 of the file description specifications must contain an F. This file does not have to exist at compilation time.
- 4** A file name can be specified for compilation time, and a different file name can be specified for run time. The E in position 19 of the file description specifications indicates that the external description of the file is to be copied in at compilation time. At run time, a file override command can be used so that a different file is accessed by the program. To override a file at run time, you must make sure that record names in both files are the same. The RPG/400 program uses the record-format name on the input/output operations, such as a READ operation where it specifies what record type is expected.

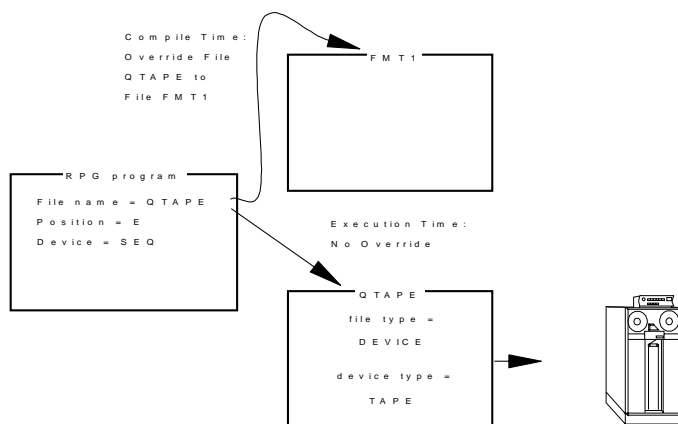
The following example shows the use of a file override at compilation time. Assume that you want to use an externally described file for a TAPE device that the system does not support. You must:

- Define a physical file named FMT1 with one record format that contains the description of each field in the record format. The record format is defined on the data description specifications (DDS). For a tape device, the externally described file should contain only one record format.
- Create the file named FMT1 with a Create Physical File CL command.

Level Checking

- Specify the file name of QTAPE (which is the IBM-supplied device file name for magnetic tape devices) in the RPG/400 program. This identifies the file as externally described (indicated by an E in position 19 of the file description specifications), and specifies the device name SEQ in positions 40 through 46.
- Use an override command—OVRDBF FILE(QTAPE) TOFILE(FMT1)—at compilation time to override the QTAPE file name and use the FMT1 file name. This command causes the compiler to copy in the external description of the FMT1 file, which describes the record format to the RPG/400 compiler.
- Create the RPG/400 program using the CRTRPGPGM command.
- Call the program at run time. The override to file FMT1 should not be in effect while the program is running. Use the CL command DLT0VR (Delete Override).

Note: You may need to use the CL command OVRTAPF before you call the program to provide information necessary for opening the tape file.



Level Checking

Because RPG/400 programs are dependent on receiving an externally described file whose format agrees with what was copied into the program at compilation time, the system provides a level-check function that ensures that the format is the same.

The RPG/400 program always provides the information required by level checking when an externally described DISK, WORKSTN, or PRINTER file is used. The level-check function can be requested on the create, change, and override file commands. The default on the create file command is to request level checking. Level checking occurs on a record-format basis when the file is opened unless you specify LVLCHK(*NO) when you issue an override command or create a file. If the level-check values do not match, the program is notified of the error. The RPG/400 program then handles the OPEN error as described in “Exception/Error Handling” on page 70.

The RPG/400 program does not provide level checking for program-described files or for files using the devices SEQ or SPECIAL.

For more information on how to specify level checking, see the *Data Management Guide*.

File Locking by an RPG/400 Program

The OS/400 system allows a lock state (exclusive, exclusive allow read, shared for update, shared no update, or shared for read) to be placed on a file used during a job. Programs within a job are not affected by file lock states. A file lock state applies only when a program in another job tries to use the file concurrently. The file lock state can be allocated with the CL command ALCOBJ (Allocate Object). For more information on allocating resources and lock states, see the *Data Management Guide*.

The OS/400 system places the following lock states on database files when it opens the files:

File Type	Lock State
Input	Shared for read
Update	Shared for update
Add	Shared for update
Output	Shared for update

The shared-for-read lock state allows another user to open the file with a lock state of shared for read, shared for update, shared no update, or exclusive allow read, but the user cannot specify the exclusive use of the file. The shared-for-update lock state allows another user to open the file with shared-for-read or shared-for-update lock state.

The RPG/400 program places an exclusive-allow-read lock state on device files. Another user can open the file with a shared-for-read lock state.

The lock state placed on the file by the RPG/400 program can be changed if you use the Allocate Object command.

Record Locking by an RPG/400 Program

When a record is read by a program, it is read in one of two modes: input or update. If a program reads a record for update, a lock is placed on that record. Another program cannot read the same record for update until the first program releases that lock. If a program reads a record for input, no lock is placed on the record. A record that is locked by one program can be read for input by another program.

In RPG/400 programs, you use an update file to read records for update. A record read from a file with a type other than update can be read for inquiry only. By default, any record is read from an update file will be read for update. For update files, you can specify that a record be read for input by using one of the input operations CHAIN, READ, READE, READP, or REDPE and specifying an N in column 53 of the calculation specification.

When a record is locked by an RPG/400 program, that lock remains until one of the following occurs:

- the record is updated.
- the record is deleted.
- another record is read from the file (either for inquiry or update).

Unblocking Input Records and Blocking Output Records

- a SETLL or SETGT operation is performed against the file
- an UNLCK operation is performed against the file.
- an output operation defined by an output specification with no field names included is performed against the file.

Note: An output operation that adds a record to a file does not result in a record lock being released.

If your program reads a record for update and that record is locked through another program in your job or through another job, your read operation will wait until the record is unlocked. If the wait time exceeds that specified on the WAITRCD parameter of the file, an exception occurs. If the default error handler is given control when a record lock timeout occurs, an RPG1218 error message will be issued. One of the options listed for this message is to retry the operation on which the timeout occurred. For programs compiled for version 2 (or higher) this will cause the operation on which the timeout occurred to be re-issued, allowing the program to continue essentially as if the record lock timeout had not occurred. Note that if the file has an INFSR specified in which an I/O operation is performed on the file before the default error handler is given control, unexpected results can occur if the input operation that is retried is a sequential operation, since the file cursor may have been modified.

With programs compiled using version 1 of the RPG/400 compiler, the retry option is still displayed, but it is not valid. If a retry is requested for a version 1 program, an additional error message (RPG1918) is issued indicating that a retry is not valid for programs compiled using version 1. In all cases, if control is returned to the program by specifying a return to *GETIN, the RPG STATUS value is set to 1218. If a retry is specified and the subsequent read operation is successful, the STATUS returns as if the record lock and subsequent retry did not occur.

If no changes are required to a locked record, you can release it from its locked state using the UNLCK operation or by processing output operations defined by output specifications with no field names included. These output operations can be processed by EXCPT output, detail output, or total output.

(There are exceptions to these rules when operating under commitment control. See Chapter 6, "Commitment Control" on page 107 for more information.)

Unblocking Input Records and Blocking Output Records

The RPG/400 compiler unblocks input records and blocks output records to improve run-time performance in SEQ or DISK files if:

- The file is an output-only file (0 is specified in position 15 of the file description specifications) and contains only one record format if the file is externally described.
- The file is a combined table file. (C is specified in position 15, and T in position 16 of the file description specifications.)
- The file is an input-only file. (I is specified in position 15 of the file description specifications.) It contains only one record format if the file is externally described, and uses only the OPEN, CLOSE, FEOD, and READ operation codes.

The RPG/400 compiler generates object program code to block and unblock records for all SEQ or DISK files that satisfy these conditions. Certain OS/400 system restrictions may prevent blocking and unblocking. In those cases, performance is not improved and the input/output feedback area is updated for each input/output operation.

The contents of positions 60 through 65 of the RECNO option in the file description specifications continuation line may not be valid when the RPG/400 compiler blocks and unblocks records.

The input/output and device-dependent sections of the file information data structure are not updated after each read or write for files in which the records are blocked and unblocked by the RPG/400 compiler. The feedback area is updated each time a block of records is transferred. (See the *RPG/400 Reference* for more information.)

Sharing an Open Data Path

An open data path is the path through which all input and output operations for a file are defined. Usually a separate open data path is defined each time a file is opened. If you specify SHARE(*YES) for the file creation or on an override, the first program's open data path for the file is shared by subsequent programs that open the file concurrently. The position of the current record is kept in the open data path for all programs using the file. If you read a record in one program and then read a record in a called program, the record retrieved by the second read depends on whether the open data path is shared. If the open data path is shared, the position of the current record in the called program is determined by the current position in the calling program. If the open data path is not shared, each program has an independent position for the current record.

If your program holds a record lock in a shared file and then calls a second program that reads the shared file for update, you can release the first program's lock by performing a READ operation on the update file by the second program, or by using the UNLCK or the read-no-lock operations.

Sharing an open data path improves performance because the OS/400 system does not have to create a new open data path. However, sharing an open data path can cause problems. For example, an error is signaled in the following cases:

- If a program sharing an open data path attempts file operations other than those specified by the first open (for example, attempting input operations although the first open specified only output operations)
- If a program sharing an open data path for an externally described file tries to use a record format that the first program ignored
- If a program sharing an open data path for a program described file specifies a record length that exceeds the length established by the first open.

When several files in one RPG/400 program are overridden to one shared file at run time, the file opening order is important. In order to control the file opening order, you should use a programmer-controlled open or use a CL program to open the files before calling the RPG/400 program.

Specifications for Externally Described Files

If a program shares the open data path for a primary or secondary file, the program must process the detail calculations for the record being processed before calling another program that shares that open data path. Otherwise, if lookahead is used or if the call is at total time, sharing the open data path for a primary or secondary file may cause the called program to read data from the wrong record in the file.

You must make sure that when the shared file is opened for the first time in a job, all of the open options that are required for subsequent opens of the file are specified. If the open options specified for subsequent opens of a shared file are not included in those specified for the first open of a shared file, an error message is sent to the program.

Table 3 details the system open options allowed for each of the open options you can specify.

RPG User Open Options	System Open Options
INPUT	INPUT
OUTPUT	OUTPUT (program created file)
UPDATE	INPUT, UPDATE
DELETE	DELETE
ADD	OUTPUT (existing file)

For additional information about sharing an open data path, see the *Database Guide*.

Using the Control Language Command RCLRSC

The Reclaim Resources (RCLRSC) CL command is designed for use in the controlling program of an application. It frees the static storage and closes any files that were left open by other programs in the application that are no longer active. This command will not always free program static storage or close all files. Using RCLRSC may close some files but keep their static storage. When this occurs, static storage indicates that these files are open, but their open data path (ODP) does not exist. When I/O is attempted with these files, an error occurs. For additional information, refer to the *CL Reference*.

Specifications for Externally Described Files

You can use the DDS to describe files to the OS/400 system. Each record type in an externally described file is identified by a unique record-format name.

The following text describes the special entries that you can use on the file description, input, and output specifications for externally described files. Remember that input and output specifications for externally described files are optional.

File Description Specifications

An E entry in position 19 of the file description specifications identifies an externally described file. The E entry indicates to the compiler that it is to retrieve the external description of the file from the system when the program is compiled.

The information in this external description includes:

- File information, such as file type, and file attributes, such as access method (by key or relative record number)
- Record-format description, which includes the record format name and field descriptions (names, locations, and attributes).

The information the RPG/400 compiler retrieves from the external description is printed on the compiler listing when the program is compiled.

Renaming Record-Format Names

Many of the functions that you can specify for externally described files (such as the CHAIN operation) operate on either a file name or a record-format name. Consequently, each file and record-format name in the program must be a unique symbolic name.

To rename a record-format name, use the RENAME option on the file description specifications continuation line for the externally described file as shown in Figure 36. You cannot specify the RENAME option on the main file description specifications line. The RENAME option is generally used if the program contains two identical record-format names or if the record-format name exceeds eight characters, which is the maximum length allowed in an RPG/400 program.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FITMMSTL IP E K DISK
F ITEMFORMAT KRENAMEMSTITM
F*
```

Figure 36. RENAME Option for Record Format Names in an Externally Described File

To rename a record format in an externally described file, use a file description specifications continuation line to specify the RENAME option. (The RENAME option cannot be specified on the main file description line because the external name positions overlap some of the entries on the main file description line.) On the continuation line, enter the external name of the record format, left-adjusted, in positions 19 through 28. Specify K in position 53, RENAME in positions 54 through 59, and the program name for the record format, left-adjusted, in positions 60 through 67. The remaining positions of the continuation line must be blank.

In this example, the record format ITEMFORMAT in the externally described file ITMMSTL is renamed MSTITM for use in this program.

Ignoring Record Formats

If a record format in an externally described file is not to be used in a program, you can use the IGNORE option to make the program run as if the record format did not exist in the file. For logical files, this means that all data associated with that format is inaccessible to the program. Use the IGNORE option on a file description specifications continuation line for the externally described file as shown in Figure 37.

The file must have more than one record format, and not all of them can be ignored; at least one must remain. Except for that requirement, any number of record formats can be ignored for a file.

If a record-format name is specified on a continuation line for the IGNORE option, it cannot be specified on a continuation line for any other option (SFILE, RENAME, or PLIST), or on a continuation line for another IGNORE.

Ignored record-format names appear on the cross-reference listing, but they are flagged as ignored.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AIOvKlocEdevice+.....KExit++Entry+A....U1.*
FITMMSTL UF E K DISK
F NOTUSED KIGNORE
F*
```

Figure 37. IGNORE Option for Record Formats in an Externally Described File

To ignore a record format from an externally described file, use a file description specifications continuation line to specify the IGNORE option. (The IGNORE option cannot be specified on the main file description line because the external name positions overlap some of the entries on the main file description line.) On the continuation line, enter the external name of the record format, left-adjusted, in positions 19 through 28, K in position 53, and IGNORE in positions 54 through 59. The remaining positions of the continuation line must be blank.

In this example, the record format NOTUSED in the externally described file ITMMSTL is ignored.

Floating-Point Fields

The RPG/400 program does not support the use of floating-point fields. If you process an externally described file with floating-point fields, the following happens:

- You cannot access the floating-point fields.
- When you create a new record, the floating-point fields in the record have the value zero.
- When you update existing records, the floating-point fields are unchanged.
- If you attempt to use a floating-point field as a key field, your program receives a compile-time error.

Overriding or Adding RPG/400 Functions to an External Description

You can use the input specifications to override certain information in the external description of an input file or to add RPG/400 functions to the external description. On the input specifications, you can:

- Assign record identifying indicators to record formats as shown in Figure 38.
- Rename a field as shown in Figure 38.
- Assign control level indicators to fields as shown in Figure 38.
- Assign match-field values to fields for matching record processing as shown in Figure 39 on page 88.
- Assign field indicators as shown in Figure 39 on page 88.

You cannot use the input specifications to override field locations in an externally described file. The fields in an externally described file are placed in the records in the order in which they are listed in the data description specifications. Also, device-dependent functions such as forms control, are not valid in an RPG/400 program for externally described files.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IRcdname+....In.....*
IMSTR1IEM      01 1
I.....Ext-field+.....Field+L1M1..P1MnZr...*
I              ITEMNUMB 2          ITEM  L1 3
I*
IMSTRWHSE      02
I              ITEMNUMB          ITEM  L1
I*
I*
```

Figure 38. Overriding and Adding RPG/400 Functions to an External Description

- 1 To assign a record identifying indicator to a record in an externally described file, specify the record-format name in positions 7 through 14 of the input specifications and assign a valid record identifying indicator in positions 19 and 20. A typical use of input specifications with externally described files is to assign record identifying indicators.

In this example, record identifying indicator 01 is assigned to the record MSTRITEM and indicator 02 to the record MSTRWHSE.

- 2 To rename a field in an externally described record, specify the external name of the field, left-adjusted, in positions 21 through 30 of the field-description line. In positions 53 through 58, specify the name that is to be used in the program.

In this example, the field ITEMNUMB in both records is renamed ITEM for this program because ITEMNUMB exceeds the maximum length of six characters that is allowed for an RPG/400 field name.

- 3 To assign a control-level indicator to a field in an externally described record, specify the name of the field in positions 53 through 58 and specify a control level indicator in positions 59 and 60.

Specifications for Externally Described Files

In this example, the ITEM field in both records MSTRITEM and MSTRWHSE is specified to be the L1 control field.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IMSTREC      01 1
I.....PFromTo++DField+L1M1FrP1MnZr...*
I                                CUSTNO M1 1
I*
IWKREC      02
I                                CUSTNO M1
I                                BALDUE                98 2
I*
```

Figure 39. Adding RPG/400 Functions to an External Description

- 1** To assign a match value to a field in an externally described record, specify the record-format name in positions 7 through 14 of the record identification line. On the field-description line specify the name of the field in positions 53 through 58 and assign a match-level value in positions 61 and 62.

In this example, the CUSTNO field in both records MSTREC and WKREC is assigned the match-level value M1.

- 2** To assign a field indicator to a field in an externally described record, specify the record-format name in positions 7 through 14 of the record identification line. On the field-description line, specify the field name in positions 53 through 58, and specify an indicator in positions 65 through 70.

In this example, the field BALDUE in the record WKREC is tested for zero when it is read into the program. If the field's value is zero, indicator 98 is set on.

Output Specifications

Output specifications are optional for an externally described file. The RPG/400 program supports file operation codes such as WRITE and UPDAT that use the external record-format description to describe the output record without requiring output specifications for the externally described file.

You can use output specification to control when the data is to be written, or to specify selective fields that are to be written. The valid entries for the field-description line for an externally described file are output indicators (positions 23 through 31), field name (positions 32 through 37), and blank after (position 39). Edit words and edit codes for fields written to an externally described file are specified in the DDS for the file. Device-dependent functions such as fetch overflow (position 16) or space/skip (positions 17-22) are not valid in an RPG/400 program for externally described files. The overflow indicator is not valid for externally described files either. For a description of how to specify editing in the DDS, see the *DDS Reference*.

If output specifications are used for an externally described file, the record-format name is specified in positions 7 through 14 instead of the file name.

If all the fields in an externally described file are to be placed in the output record, enter *ALL in positions 32 through 37 of the field-description line. If *ALL is specified, you cannot specify other field description lines for that record.

If you want to place only certain fields in the output record, enter the field name in positions 32 through 37. The field names you specify in these positions must be the field names defined in the external record description, unless the field was renamed on the input specifications. See Figure 40.

You should know about these considerations for using the output specifications for an externally described file:

- In the output of an update record, only those fields specified in the output field specifications and meeting the conditions specified by the output indicators are placed in the output record to be rewritten. Fields not specified in the output specifications are rewritten using the values that were read. This technique offers a good method of control as opposed to the UPDAT operation code that updates all fields.
- In the creation of a new record, the fields specified in the output field specifications are placed in the record. Fields not specified in the output field specifications or not meeting the conditions specified by the output indicators are written as zeros or blanks depending on the data format specified in the external description.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
ONAME++++DFBASbSaN01N02N03Excnam.....*
OITMREC  D          20
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0                                *ALL 1
0*
0*
OSLSREC  D          30
0                                SLSNAM 2
0                                COMRAT
0                                15  BONUS
0*
0*
```

Figure 40. Output Specifications for an Externally Described File

- 1** For an update file, all fields in the record are written to the externally described record ITMREC using the current values in the program for all fields in the record.
For the creation of a new record, all fields in the record are written to the externally described record ITMREC using the current values in the program for the fields in the record.
- 2** To update a record, the fields SLSNAM and COMRAT are written to the externally described record SLSREC when indicator 30 is on. The field BONUS is written to the SLSREC record when indicators 30 and 15 are on. All other fields in the record are written with the values that were read.

Printer Files

To create a new record, the fields SLSNAM and COMRAT are written to the externally described record SLSREC when indicator 30 is on. The field BONUS is written when indicators 30 and 15 are on. All other fields in the record are written as zeros or blanks, depending on whether the field is numeric or character.

Program-Described Files

Program-described files are files whose records and fields are described on input/output specifications in the program that uses the file.

Figure 41 shows how to specify sequence checking when your input data must contain exactly one record of the first type (01 in positions 15 and 16), followed by at least one record of another type (02 through 04 in positions 15 and 16) in each group of records read. When the specifications shown in Figure 41 are used and two consecutive records of the first type are read, a run-time error occurs.

If each group of input records must contain exactly one record of a particular type, but that record need not be followed by any records of other types, specify no sequence checking (alphabetic entry in positions 15 and 16).

Write operations to a program-described file require a data structure name in the result field.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IINPUT 011 10 1 CA
I.....PFromTo++DField+L1M1FrP1MnZr...*
I 2 60TYPE
I 02N020 1 CB
I 2 11 KEY
I 03N030 1 CC
I 2 21 NAME
I 04N040 1 CD
I 2 6 NUMBER
I*
```

Figure 41. Input Specifications for Sequence Checking

Printer Files

The PRINTER file allows you to print the output file. A maximum of eight printer files is allowed per program. You must assign PRINTER as the device name for the file, and each file must have a unique file name. You can use the CL command CRTPRTF (Create Print File) to create a printer file (see the *CL Reference* for further information on the CRTPRTF command); or you can also use the IBM-supplied file names. See the *Data Management Guide* for more information on these file names.

The file operation codes that are valid for a PRINTER file are WRITE, OPEN, CLOSE, and FEOD. For a complete description of these operation codes, see the *RPG/400 Reference*.

PRINTER files can be either externally described or program described. Overflow indicators 0A-0G and 0V, fetch overflow, space/skip entries, and the PRTCTL option are not allowed for an externally described PRINTER file. See the *RPG/400 Reference* for the valid output specification entries for an externally described file. See the *DDS Reference* for information on the DDS for externally described printer files.

For an externally described PRINTER file, you can specify the DDS keyword INDARA. If you try to use this keyword for a program-described PRINTER file, you get a run-time error.

Page Overflow

An important consideration when you use a PRINTER file is page overflow. For an externally described PRINTER file, you are responsible for handling page overflow. Do one of the following:

- Count the number of output lines per page.
- Check for a file exception/error by specifying an indicator in positions 56 and 57 of the calculation specifications that specify the output operation, or by specifying an INFSR that can handle the error. The INFDS has detailed information on the file exception/error. See “Exception/Error Handling” on page 70 for further information on exception and error handling.
- Specify an indicator 01 through 99 as the overflow indicator in positions 33 and 34 of the file description specifications.
- Check INFDS for line number and page overflow. Refer to the *RPG/400 Reference* for more information.

For either a program-described or an externally described file, you can specify an indicator 01 through 99 in positions 33 and 34 of the file description specifications. This indicator is set on when a line is printed on the overflow line, or the overflow line is reached or passed during a space or skip operation. Use the indicator to condition your response to the overflow condition. The indicator does not condition the RPG/400 overflow logic as an overflow indicator (0A through 0G, 0V) does. You are responsible for setting the indicator off.

For both program-described and externally described files, the line number and page number are available in the file's INFDS. To access this information, specify an INFDS for the file using a file continuation specification. On the specification, define the line number in positions 367-368 and define the page number in positions 369-372 of the data structure. Both the line number and the page number fields must be defined as binary with no decimal positions. Because the INFDS will be updated after every output operation to the printer file, these fields can be used to determine the current line and page number without having line-count logic in the program.

For a program-described PRINTER file, the following sections on overflow indicators and fetch overflow logic apply.

Overflow Indicators

An overflow indicator (0A through 0G, 0V) is set on when the last line on a page has been printed or passed. An overflow indicator can be used to specify the lines to be printed on the next page. Overflow indicators can be specified only for program-described PRINTER files and are used primarily to condition the printing of heading lines. An overflow indicator is defined in positions 33 and 34 of the file description specifications and can be used to condition operations in the calculation specifications (positions 9 through 17) and output specifications (positions 23 through 31). If an overflow indicator is not specified, the RPG/400 compiler assigns the first unused overflow indicator to the PRINTER file. Overflow indicators can also be specified as resulting indicators on the calculation specifications (positions 54 through 59).

The RPG/400 compiler sets on an overflow indicator only the first time an overflow condition occurs on a page. An overflow condition exists whenever one of the following occurs:

- A line is printed past the overflow line.
- The overflow line is passed during a space operation.
- The overflow line is passed during a skip operation.

Table 4 on page 93 shows the results of the presence or absence of an overflow indicator on the file description and output specifications.

The following considerations apply to overflow indicators used on the output specifications:

- Spacing past the overflow line sets the overflow indicator on.
- Skipping past the overflow line to any line on the same page sets the overflow indicator on.
- Skipping past the overflow line to any line on the new page does not set the overflow indicator on unless a skip-to is specified past the specified overflow line.
- A skip to a new page specified on a line not conditioned by an overflow indicator sets the overflow indicator off after the forms advance to a new page.
- If you specify a skip to a new line and the printer is currently on that line, a skip does not occur. The overflow indicator is set to off, unless the line is past the overflow line.
- When an 0R line is specified for an output print record, the space and skip entries of the preceding line are used. If they differ from the preceding line, enter space and skip entries on the 0R line.
- Control level indicators can be used with an overflow indicator so that each page contains information from only one control group. See Figure 42 on page 94.
- For conditioning an overflow line, an overflow indicator can appear in either an AND or an OR relationship. For an AND relationship, the overflow indicator must appear on the main specification line for that line to be considered an overflow line. For an OR relationship, the overflow indicator can be specified on either the main specification line or the OR line. Only one overflow indicator can be associated with one group of output indicators. For an OR relationship, only the

conditioning indicators on the specification line where an overflow indicator is specified is used for the conditioning of the overflow line.

- If an overflow indicator is used on an AND line, the line is *not* an overflow line. In this case, the overflow indicator is treated like any other output indicator.
- When the overflow indicator is used in an AND relationship with a record identifying indicator, unusual results are often obtained because the record type might not be the one read when overflow occurred. Therefore, the record identifying indicator is not on, and all lines conditioned by both overflow and record identifying indicators do not print.
- An overflow indicator conditions an exception line (E in position 15), and conditions fields within the exception record.

Table 4. Results of the Presence or Absence of an Overflow Indicator

File Description Specifications Positions 33-34	Output Specifications Positions 23-31	Action
No entry	No entry	First unused overflow indicator used to condition skip to next page at overflow.
No entry	Entry	Error at compile time; overflow indicator dropped from output specifications. First unused overflow indicator used to condition skip to next page at overflow.
Entry	No entry	Continuous printing; no overflow recognized.
Entry	Entry	Processes normal overflow.

The first part of the following figure is an example of the coding necessary for printing headings on every page: first page, every overflow page, and each new page to be started because of a change in control fields (L2 is on). The first line allows the headings to be printed at the top of a new page (skip to 06) only when an overflow occurs (0A is on and L2 is not on).

The second line allows printing of headings on the new page only at the beginning of a new control group (L2 is on). This way, duplicate headings caused by both L2 and 0A being on at the same time do not occur. The second line allows headings to be printed on the first page after the first record is read because the first record always causes a control break (L2 turns on) if control fields are specified on the record.

The second part of the figure is the necessary coding for the printing of certain fields on every page; a skip to 06 is done either on an overflow condition or on a change in control level (L2). The NL2 indicator prevents the line from printing and skipping twice in the same cycle.

Printer Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OPRINT H 306 OANL2
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0 OR L2
0 8 'DATE'
0 18 'ACCOUNT'
0 28 'NAME'
0 46 'BALANCE'
0*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OPRINT D 306 OANL2
0 OR L2
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0 ACCT 8
0*
```

Figure 42. Using Control Level Indicators with an Overflow Indicator

Fetch-Overflow Logic

When there is not enough space left on a page to print the remaining detail, total, exception, and heading lines conditioned by the overflow indicator, the fetch overflow routine can be called. This routine causes an overflow. To determine when to fetch the overflow routine, study all possible overflow situations. By counting lines and spaces, you can calculate what happens if overflow occurs on each detail, total, and exception line.

The fetch-overflow routine allows you to alter the basic RPG/400 overflow logic to prevent printing over the perforation and to let you use as much of the page as possible. During the regular program cycle, the RPG/400 compiler checks only once, immediately after total output, to see if the overflow indicator is on. When the fetch overflow function is specified, the RPG/400 compiler checks overflow on each line for which fetch overflow is specified. See Figure 43 on page 95.

Specify fetch overflow with an F in position 16 of the output specifications on any detail, total, or exception lines for a PRINTER file. The fetch overflow routine does not automatically cause forms to advance to the next page.

During output, the conditioning indicators on an output line are tested to determine if the line is to be written. If the line is to be written and an F is specified in position 16, the RPG/400 compiler tests to determine if the overflow indicator is on. If the overflow indicator is on, the overflow routine is fetched and the following operations occur:

1. Only the overflow lines for the file with the fetch specified are checked for output.
2. All total lines conditioned by the overflow indicator are written.

3. Forms advance to a new page when a skip to a line number less than the line number the printer is currently on is specified in a line conditioned by an overflow indicator.
4. Heading, detail, and exception lines conditioned by the overflow indicator are written.
5. The line that fetched the overflow routine is written.
6. Any detail and total lines left to be written for that program cycle are written.

Position 16 of each 0R line must contain an F if the overflow routine is to be used for each record in the 0R relationship. Fetch overflow cannot be used if an overflow indicator is specified in positions 23 through 31 of the same specification line. If this is the case, the overflow routine is not fetched.

Figure 44 on page 96 shows the use of fetch overflow.

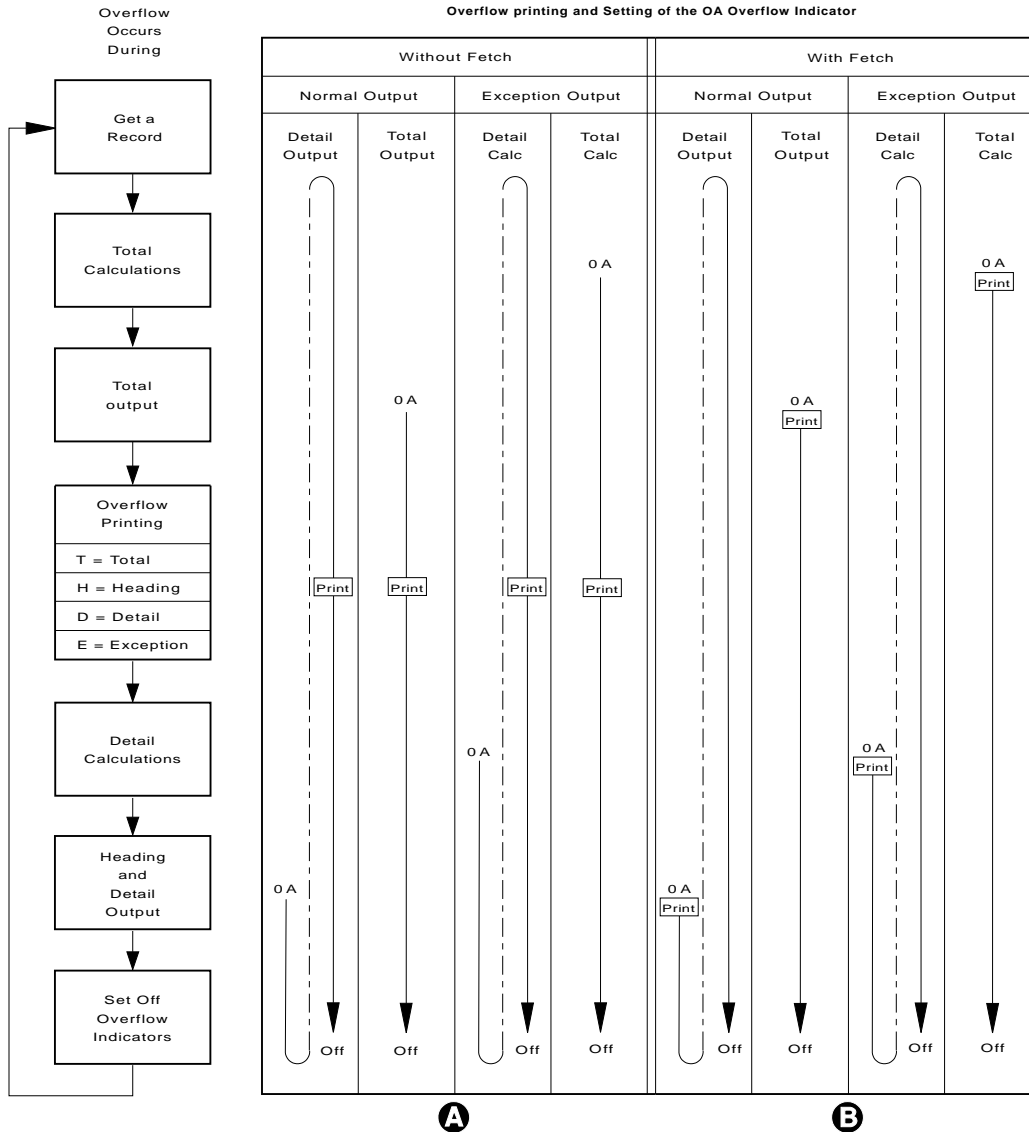


Figure 43. Overflow Printing: Setting of the Overflow Indicator

Printer Files

- A** When fetch overflow is not specified, the overflow lines print after total output. No matter when overflow occurs (OA is on), the overflow indicator OA remains on through overflow output time and is set off after heading and detail output time.
- B** When fetch overflow is specified, the overflow lines are written before the output line for which fetch overflow was specified, if the overflow indicator OA is on. When OA is set on, it remains on until after heading and detail output time. The overflow lines are not written a second time at overflow output time unless overflow is sensed again since the last time the overflow lines were written.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINTER H 305 OA
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0
0 15 'EMPLOYEE TOTAL'
0 TF 1 L1
0 EMPLTOT 25
0 T 1 L1
0 EMPLTOT 35
0 T 1 L1
0 EMPLTOT 45
0 TF 1 L1
0 EMPLTOT 55
0 T 1 L1
0 EMPLTOT 65
0 T 1 L1
0 EMPLTOT 75
0*
0*
```

Figure 44. Use of Fetch Overflow

The total lines with an F coded in position 16 can fetch the overflow routine. They only do so if overflow is sensed prior to the printing of one of these lines. Before fetch overflow is processed, a check is made to determine whether the overflow indicator is on. If it is on, the overflow routine is fetched, the heading line conditioned by the overflow indicator is printed, and the total operations are processed.

PRTCTL (Printer Control) Option

The PRTCTL (printer control) option allows you to change forms control information and to access the current line value within the program for a program-described PRINTER file.

Specify the PRTCTL option on a file description specifications continuation line for the PRINTER file with the following:

Note: If the file has a share ODP or user-controlled open, the line count value may be incorrect.

Position	Entry
6	F
7-52	Blank if the information is specified on a separate continuation line
53	K (indicates a continuation line)
54-59	PRTCTL
60-65	Name of the data structure used to contain the printer control and line count information
66-74	Blank if the information is specified on a separate continuation line.

The data structure specified in positions 60 through 65 of the file description specifications continuation line must be specified on the input specifications and must contain at least the following five subfields specified in the following order:

Data Structure Positions	Subfield Contents
1	A one-position character field that contains the space-before value
2	A one-position character field that contains the space-after value
3-4	A two-position character field that contains the skip-before value
5-6	A two-position character field that contains the skip-after value
7-9	A three-digit numeric field with zero decimal positions that contains the current line count value.

The values contained in the first four subfields of the data structure are the same as those allowed in positions 17 through 22 (space and skip entries) of the output specifications. If the space/skip entries (positions 17 through 22) of the output specifications are blank, and if subfields 1 through 4 are also blank, the default is to space 1 after. If the PRTCTL option is specified, it is used only for the output records that have blanks in positions 17 through 22. You can control the space and skip value (subfields 1 through 4) for the PRINTER file by changing the values in these subfields while the program is running. See Figure 45 on page 98.

Subfield 5 contains the current line count value. The RPG/400 compiler does not initialize subfield 5 until after the first output line is printed. The RPG/400 compiler then changes subfield 5 after each output operation to the file.

Printer Files

Figure 46 on page 99 is a processing chart for PRINTER files.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRINT 0 F 132 PRINTER KPRTCTLLINE
F*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IDname....NODsExt-file+.....OccrLen+.....*
ILINE DS
I.....Ext-field+.....PFromTo++DField+.....*
I 1 1 SPBEFR
I 2 2 SPAFTR
I 3 4 SKBEFR
I 5 6 SKAFTR
I 7 90CURLIN
I*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C 01 CURLIN COMP 10 49
C 01 49 MOVE '3' SPAFTR
C*
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OPRINT 01
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
0 DATA 25
O*
```

Figure 45. Example of the PRTCTL Option

On the file description specifications, the PRTCTL option is specified for the PRINT file. The name of the associated data structure is LINE.

The LINE data structure is defined on the input specifications as having only those subfields that are predefined for the PRTCTL data structure. The first four subfields in positions 1 through 6 are used to supply space and skip information that is generally specified in positions 17 through 22 of the output specifications. The PRTCTL option allows you to change these specifications within the program.

In this example, the value in the SPAFTR subfield is changed to 3 when the value in the CURLIN (current line count value) subfield is equal to 10. (Assume that indicator 01 was set on as a record identifying indicator.)

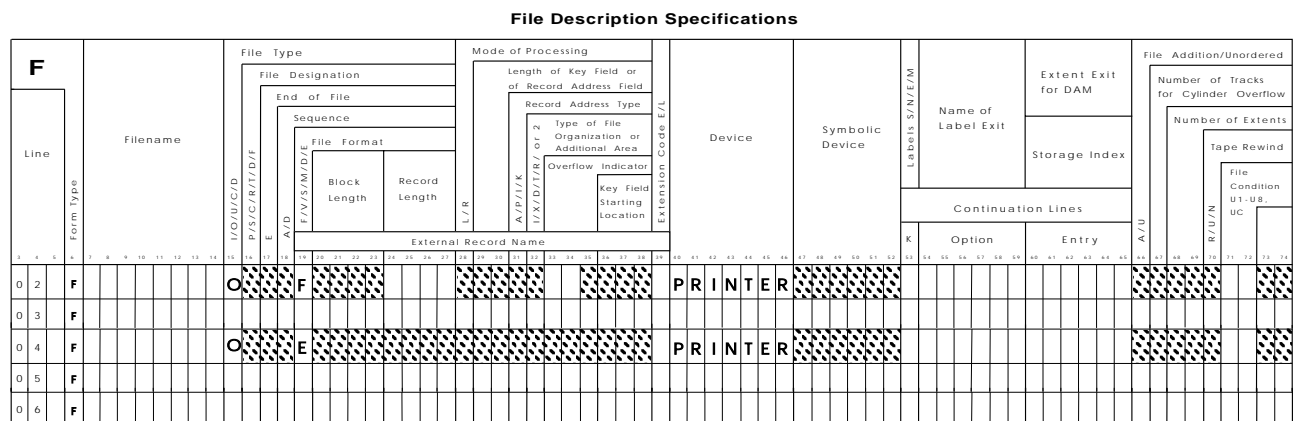


Figure 46. Processing Chart for PRINTER Files

Valid File Operations:

- 1. WRITE, OPEN, CLOSE, FEOD

Note: Shaded positions must be blank. Positions without entries are program dependent.

Sequential File

A sequential (SEQ) device specification, positions 40 through 46 in the file description specification, indicates that the input or output is associated with a sequentially organized file. Refer to Figure 47. The actual device to be associated with the file while running the RPG/400 program can be specified by a OS/400 override command (see the *CL Reference* for further information), or by the file description that is pointed to by the file name.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTIMECDS IP E DISK
FPAYOTIMEO F 132 SEQ
F*
F*
```

Figure 47. SEQ Device

A SEQ device is specified for the PAYOTIME file. When the program is run, you can use a OS/400 override command to specify the actual device (such as printer, tape, or diskette) to be associated with the file while the program is running. For example, diskette can be specified for some program runs while printer can be specified for others. The file description, pointed to by the file name, can specify the actual device, in which case an override command need not be used.

Any sequentially organized file, such as diskette, tape, database, savefile, or printer, can be associated with the SEQ device. If SEQ is specified in an RPG/400 program, no device-dependent functions such as space/skip, or CHAIN can be specified.

Use the SEQ device specifications whenever you write to a tape file. To write variable-length records to a tape file, also use the RCDBLKFMF parameter of either

Sequential File

the CL command CRTTAPF or OVRTAPF. (See the *CL Reference*.) When you use the RCDBLKFMFMT parameter, the length of each record that your program writes to the tape file is determined by the highest end position specified in positions 40 through 43 of the output specifications for that record. If you do not specify an end position, the RPG/400 compiler calculates the record length from the length of the fields in the record.

Read variable-length tape records as you would read records from any sequentially organized file. Be sure the record length specified on the file description specification accommodates the longest record in the file.

The following figure shows the operation codes allowed for a SEQ file.

File Description Specifications		Calculation Specifications
Positions 15	16	Positions 28-32
I	P/S	CLOSE, FEOD
I	F	READ, OPEN, CLOSE, FEOD
O		WRITE, OPEN, CLOSE, FEOD

Note: No print control specifications are allowed for a sequential file.

Figure 48. Valid File Operation Codes for a Sequential File

Figure 49 on page 101 is a processing chart for SEQ files.

File Description Specifications

Line	Form Type	Filename	File Type					Mode of Processing						Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered				
			File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator							Key Field Starting Location	Extension Code E/L	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind
			M/D/N/C/D	P/S/C/N/T/D/F	E	A/D	T/N/S/M/D/E	L/R	A/B/K	X/D/T/R/Z or 3	Overflow Indicator	Key Field Starting Location	Extension Code E/L						A/U	R/U/N	U1-UB, UC			
02	F		I	P	F									S	E	Q								
03	F		I	S	F									S	E	Q								
04	F																							
05	F		I	F	F									S	E	Q								
06	F																							
07	F		I	R	F									E	S	E	Q							
08	F																							
09	F		I	R	F					T				E	S	E	Q							
10	F																							
11	F		C	T	F									E	S	E	Q							
12	F																							
13	F		O		F									S	E	Q								
14	F																							
15	F		I	P	E									S	E	Q								
16	F		I	S	E									S	E	Q								
17	F																							
18	F		I	F	E									S	E	Q								
19	F																							
20	F		O		E									S	E	Q								
21	F																							

Figure 49. Processing Chart for SEQ Files

Valid File Operations:

1. CLOSE, FEOD
2. READ, OPEN, CLOSE, FEOD
3. OPEN, CLOSE, FEOD
4. WRITE, OPEN, CLOSE, FEOD

Note: Shaded positions must be blank. Positions without entries are program dependent.

Special File

SPECIAL in positions 40 through 46 of the file description specifications allows you to specify an input and/or output device that is not directly supported by the RPG/400 functions. The input and output operations for the file are controlled by a user-written routine. Positions 54 through 59 of the file description specifications line that contains SPECIAL in positions 40 through 46 must contain the name of the user-written routine.

RPG/400 calls this user-written routine to open the file, read and write the records, and close the file. RPG/400 creates a parameter list for use by the user-written routine. The parameter list contains an option code parameter (option), a return status parameter (status), an error-found parameter (error), and a record area parameter (area). This parameter list is accessed by the RPG/400 program and by the user-written routine; it cannot be accessed by the RPG/400 program that contains the SPECIAL file.

The following describes the parameters in this RPG-created parameter list:

- **Option:** The option parameter is a one-position character field that indicates the action the user-written routine is to process. Depending on the operation being processed on the SPECIAL file (OPEN, CLOSE, READ, WRITE, DELET, UPDAT), one of the following values is passed to the user-written routine from RPG/400:

Value Passed	Description
O	Open the file.
C	Close the file.
R	Read a record and place it in the area defined by the area parameter.
W	The RPG/400 program has placed a record in the area defined by the area parameter; the record is to be written out.
D	Delete the record.
U	The record is an update of the last record read.

- **Status:** The status parameter is a one-position character field that indicates the status of the user-written routine when control is returned to the RPG/400 program. Status must contain one of the following return values when the user-written routine returns control to the RPG/400 program:

Return Value	Description
0	Normal return. The requested action was processed.
1	The input file is at end of file, and no record has been returned. If the file is an output file, this return value is an error.
2	The requested action was not processed; error condition exists.

- **Error:** The error parameter is a five-digit zoned numeric field with zero decimal positions. If the user-written routine detects an error, the error parameter contains an indication or value representing the type of error. The value is placed in the first five positions of location *RECORD in the INFDS when the status parameter contains 2.
- **Area:** The area parameter is a character field whose length is equal to the record length associated with the SPECIAL file. This field is used to pass the record to or receive the record from the RPG/400 program.

You can add additional parameters to the RPG-created parameter list. Specify PLIST in positions 54 through 59 and the name of the PLIST in positions 60 through 65 of a file description specifications continuation line for the SPECIAL file. See Figure 50. Then use the PLIST operation in the calculation specifications to define the additional parameters.

The user-written routine, the name that is specified in positions 54 through 59 of the file description specifications for the SPECIAL file, must contain an entry parameter list that includes both the RPG-created parameters and the user-specified parameters.

If the SPECIAL file is specified as a primary file, the user-specified parameters must be initialized before the first primary read. You can initialize these parameters with a factor 2 entry on the PARM statements or by the specification of a compile-time array or an array element as a parameter.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FEXCPTN 1 F                                SPECIAL          USERIO
F                                           KPLIST SPCL
F*
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C                SPCL          PLIST
C                  PARM          FLD1
C                  PARM          FLD2
C                  PARM          FLD3
C*
C*
```

Figure 50. SPECIAL Device

The file EXCPTN is assigned to the device SPECIAL. The I/O operations for the SPECIAL device are controlled by the user-written routine USERIO. The parameters specified for the programmer-defined PLIST SPCL are added to the end of the RPG-created parameter list for the SPECIAL device. The programmer-specified parameters can be accessed by the user RPG/400 program and the user-written routine; whereas the RPG-created parameter list can be accessed only by internal RPG/400 logic and the user-written routine.

Special File

Figure 51 shows the file operation codes that are valid for a SPECIAL file.

File Description Specifications		Calculation Specifications
Positions 15	16	Positions 28-32
I	P/S	CLOSE, FEOD
C	P/S	WRITE, CLOSE, FEOD
U	P/S	UPDAT, DELET, CLOSE FEOD
O		WRITE, OPEN, CLOSE, FEOD
I	F	READ, WRITE, OPEN, CLOSE, FEOD
C	F	READ, WRITE, OPEN, CLOSE, FEOD
U	F	READ, UPDAT, DELET, OPEN, CLOSE, FEOD

Figure 51. Valid File Operations for a SPECIAL File

Valid File Operations:

1. CLOSE, FEOD
2. WRITE, CLOSE, FEOD
3. UPDAT, DELET, CLOSE, FEOD
4. READ, OPEN, CLOSE, FEOD
5. READ, WRITE, OPEN, CLOSE, FEOD
6. READ, DELET, UPDAT, OPEN, CLOSE, FEOD
7. WRITE, OPEN, CLOSE, FEOD

Notes:

1. Shaded positions must be blank. Positions without entries are program dependent.
2. Positions 54 through 59 must contain the name of the user-written routine that controls the input/output operations for the file.

Chapter 6. Commitment Control

This chapter describes how to use commitment control to process file operations as a group. With commitment control, you ensure one of two outcomes for the file operations: either all of the file operations are successful or none of the file operations has any effect. In this way, you process a group of operations as a unit.

Using Commitment Control

To use commitment control, you do the following:

- Use the CL commands CRTJRN (Create Journal), CRTJRNCV (Create Journal Receiver) and STRJRNPf (Journal Physical File) to prepare for using commitment control, and the CL commands STRCMTCTL (Start Commitment Control) and ENDCMTCTL (End Commitment Control) to notify the system when you want to start and end commitment control. See the *CL Reference* for information on these commands.
- Specify commitment control on the file-description specifications of the files you want under commitment control.
- Use the COMMIT (Commit) operation code to apply a group of changes to files under commitment control, or use the ROLBK (Roll Back) operation code to eliminate the pending group of changes to files under commitment control.

Starting and Ending Commitment Control

The CL command STRCMTCTL notifies the system that you want to process files under commitment control.

The LCKLVL (Lock Level) parameter allows you to select the level at which records are locked under commitment control. See “Commitment Control Locks” on page 109 and the *CL Programmer’s Guide* for further details on lock levels.

When you complete a group of changes with a COMMIT operation, you can specify a label to identify the end of the group. In the event of an abnormal job end, this identification label is written to a file, message queue, or data area so that you know which group of changes is the last group to be completed successfully. You specify this file, message queue, or data area on the STRCMTCTL command.

Before you call any program that processes files specified for commitment control, issue the STRCMTCTL command. If you call a program that opens a file specified for commitment control before you issue the STRCMTCTL command, the opening of the file will fail.

The CL command ENDCMTCTL notifies the system that your routing step has finished processing files under commitment control. See the *CL Reference* for further information on the STRCMTCTL and ENDCMTCTL commands.

Specifying Files for Commitment Control

On the file-continuation specifications, enter a K in position 53 and the word `COMIT` in positions 54 through 59. On the file-description specifications, describe the file as having device `DISK` in positions 40 through 46.

When a program specifies commitment control for a file, the specification applies only to the input and output operations made by this program for this file. Commitment control does not apply to operations other than input and output operations. It does not apply to files that do not have commitment control specified in the program doing the input or output operation.

When more than one program accesses a file as a shared file, all or none of the programs must specify the file to be under commitment control.

Commitment Control Operations

The `COMIT` (Commit) operation tells the system that you have completed a group of changes to the files under commitment control. The `ROLBK` (Roll Back) operation eliminates the current group of changes to the files under commitment control. For information on how to specify these operation codes and what each operation does, see the *RPG/400 Reference*.

If the system fails, it implicitly issues a `ROLBK` operation. You can check the identity of the last successfully completed group of changes using the label you specify in factor 1 of the `COMIT` operation code, and the notify-object you specify on the `STRCMTCTL` command.

At the end of a routing step, or when you issue the `ENDCMTCTL` command, the OS/400 system issues an implicit `ROLBK`, which eliminates any changes since the last `ROLBK` or `COMIT` operation that you issued. To ensure that all your file operations have effect, issue a `COMIT` operation before ending a routing step operating under commitment control.

The `OPEN` operation permits input and output operations to be made to a file and the `CLOSE` operation stops input and output operations from being made to a file. However, the `OPEN` and `CLOSE` operations do not affect the `COMIT` and `ROLBK` operations. A `COMIT` or `ROLBK` operation affects a file, even after the file has been closed. For example, your program may include the following steps:

1. Issue `COMIT` (for files already opened under commitment control).
2. Open a file specified for commitment control.
3. Perform some input and output operations to this file.
4. Close the file.
5. Issue `ROLBK`.

The changes made at step 3 are rolled back by the `ROLBK` operation at step 5, even though the file has been closed at step 4. The `ROLBK` operation could be issued from another program in the same routing step.

A program does not have to operate all its files under commitment control, and to do so may adversely affect performance. The `COMIT` and `ROLBK` operations have no effect on files that are not under commitment control.

Note: When multiple devices are attached to an application program, and commitment control is in effect for the files this program uses, the COMMIT or ROLBK operations continue to work on a file basis and not by device. The database may be updated with partially completed COMMIT blocks or changes that other users have completed may be eliminated. It is your responsibility to ensure this does not happen.

Commitment Control Locks

On the STRCMTCTL command, you specify a level of locking, either LCKLVL (*ALL) or LCKLVL (*CHG). When your program is operating under commitment control and has processed an input or output operation on a record in a file under commitment control, the record is locked by commitment control as follows:

- Your program can access the record.
- Another program in your routing step, with this file under commitment control, can read the record. If the file is a shared file, the second program can also update the record.
- Another program in your routing step that does not have this file under commitment control cannot read or update the record.
- Another program in a separate routing step, with this file under commitment control, can read the record if you specified LCKLVL (*CHG), but it cannot read the record if you specified LCKLVL (*ALL). With either lock level, the next program cannot update the record.
- Another program that does not have this file under commitment control and that is not in your routing step can read but not update the record.
- Commitment control locks are different than normal locks, depend on the LCKLVL specified, and can only be released by the COMMIT and ROLBK operations.

The COMMIT and ROLBK operations release the locks on the records. The UNLCK operation will not release records locked using commitment control. See the *CL Reference* for details on lock levels.

The number of entries that can be locked under commitment control before the COMMIT or ROLBK operations are required may be limited. For more information, see the *Advanced Backup and Recovery Guide*

Note: The SETLL and SETGT operations lock a record where a read operation (not for update) would lock a record for commitment control.

Commitment Control in the Program Cycle

Commitment control is intended for full procedural files, where the input and output is under your control. Do not use commitment control with primary and secondary files, where input and output is under the control of the RPG/400 program cycle.

The following are some of the reasons for this recommendation:

- You cannot issue a COMMIT operation for the last total output in your program.
- It is difficult to program within the cycle for recovery from a locked-record condition.
- Level indicators are not reset by the ROLBK operation.
- After a ROLBK operation, processing matching records may produce a sequence error.

Example of Using Commitment Control

The following is an example of the specifications and CL commands for a program operating under commitment control.

To prepare for using commitment control, you issue the following CL commands:

- CRTJRNRCV JRNRCV (RECEIVER)

The above command creates a journal receiver named RECEIVER.

- CRTJRN JRN(JOURNAL) JRNRCV(RECEIVER)

The above command creates a journal named JOURNAL and attaches the journal receiver named RECEIVER.

- STRJRNPF FILE(MASTER) JRN(JOURNAL)

The above command directs journal entries for the file MASTER to the journal JOURNAL.

In your program, you specify COMIT for the file MASTER:.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FMASTER UF E K DISK KCOMIT
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* In the calculation specifications, use the COMIT operation to
C* complete a group of operations.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C KEY CHAINMASTER 50
C N50 UPDATRECORD 99
C N99 COMIT
C*
C* If an operation within a group fails, use the ROLBK operation
C* to eliminate the entire group of operations.
C*
C 99 ROLBK
C*
```

Figure 53. Example of Using Commitment Control

To operate your program (named REVISE) under commitment control, you issue the commands:

- STRCMTCTL LCKLVL(*ALL)

The above command starts commitment control, with the highest level of locking.

- CALL REVISE

The above command calls your program (named REVISE).

- ENDCMTCTL

The above command ends commitment control, and causes an implicit Roll Back operation.

Chapter 7. Using DISK Files

Database files, which are associated with the RPG/400 device DISK in positions 40 through 46 of the file description specifications, can be:

- Externally described files, whose fields are described to the OS/400 system through the data description specifications (DDS)
- Program-described files, whose fields are described on input/output specifications in the program that uses the file.

All database files are created by the OS/400 create file commands. See the *CL Reference* for a description of the OS/400 commands that relate to database files.

Externally Described Disk Files

Externally described DISK files are identified by an E in position 19 of the file description specifications. The E indicates that the compiler is to retrieve the external description of the file from the system when the program is compiled. Therefore, you must create the file before the program is compiled.

The external description for a DISK file includes:

- The record-format specifications that contain a description of the fields in a record
- Access path specifications that describe how the records are to be retrieved.

These specifications result from the DDS for the file and the OS/400 create file command that is used for the file.

Record Format Specifications

The record-format specifications allow you to describe the fields in a record and the location of the fields in a record. The fields are located in the record in the order specified in the DDS. The field description generally includes the field name, the field type (character, binary, zoned decimal, or packed decimal), and the field length (including the number of decimal positions in a numeric field). Instead of specifying the field attributes in the record format for a physical or logical file, you can define them in a field-reference file.

In addition, the DDS keywords can be used to:

- Specify that duplicate key values are not allowed for the file (UNIQUE)
- Specify a text description for a record format or a field (TEXT).

For a complete list of the DDS keywords that are valid for a database file, see the *Database Guide*.

Externally Described Disk Files

Figure 54 shows an example of the DDS for a database file, and Figure 55 on page 115 for a field-reference file that defines the attributes for the fields used in the database file. See the *DDS Reference* for more information on a field-reference file.

Access Path

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (non-keyed) access path or on a keyed-sequence access path.

The arrival sequence access path is based on the order in which the records are stored in the file. Records are added to the file one after another.

For the keyed-sequence access path, the sequence of records in the file is based on the contents of the key field that is defined in the DDS for the file. For example, in the DDS shown in Figure 54, CUST is defined as the key field. The keyed-sequence access path is updated whenever records are added, deleted, or when the contents of a key field change.

For a complete description of the access paths for an externally described database file, see the *Database Guide*.

```
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
A.....T.Name+++++.Len++TDpB.....Functions+++++*****
A** LOGICAL  CUSMSTL    CUSTOMER MASTER FILE
A
A          R CUSREC          UNIQUE
A          PFIL(CUSMSTP)
A          TEXT('Customer Master Record')
A          CUST
A          NAME
A          ADDR
A          CITY
A          STATE
A          ZIP
A          SRHCOD
A          CUSTYP
A          ARBAL
A          ORDBAL
A          LSTAMT
A          LSTDAT
A          CRDLMT
A          SLSYR
A          SLSLYR
A          K CUST
```

Figure 54. Example of the Data Description Specifications for a Database File

The sample DDS are for the customer master logical file CUSMSTL. The file contains one record format CUSREC (customer master record). The data for this file is contained in the physical file CUSMSTP, which is identified by the keyword PFILE. The UNIQUE keyword is used to indicate that duplicate key values are not allowed for this file. The CUST field is identified by a K in position 17 of the last line as the key field for this record format.

The fields in this record format are listed in the order they are to appear in the record. The attributes for the fields are obtained from the physical file CUSMSTP. The physical file, in turn, refers to a field-reference file to obtain the attributes for the fields. The field-reference file is shown in Figure 55.

```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A**FLDRED   DSTREF   DISTRIBUTION APPLICATION FIELD REFERENCE
A           R DSTREF           TEXT('Distribution Field Ref')
A* COMMON FIELDS USED AS REFERENCE
A           BASDAT           6 0           EDTCDE(Y) 1
A           TEXT('Base Date Field')
A* FIELDS USED BY CUSTOMER MASTER FILE
A           CUST             5           CHECK(MF) 2
A           COLHDG('Customer' 'Number')
A           NAME             20          COLHDG('Customer Name')
A           ADDR             R           REFFLD(NAME) 3
A           COLHDG('Customer Address')
A           CITY             R           REFFLD(NAME) 3
A           COLHDG('Customer City')
A           STATE            2           CHECK(MF) 2
A           COLHDG('State')
A           SRHCOD            6           CHECK(MF) 2
A           COLHDG('Search' 'Code')
A           TEXT('Customer Number Search +
A           Code')
A           ZIP              5 0          CHECK(MF) 2
A           COLHDG('Zip' 'Code')
A           CUSTYP            1 0          RANGE(1 5) 4
A           COLHDG('Cust' 'Type')
A           TEXT('Customer Type 1=Gov 2=Sch+
A           3=Bus 4=Pvt 5=0th')
A           ARBAL            8 2          COLHDG('Accts Rec' 'Balance') 5
A           EDTCDE(J) 6
A           ORDBAL           R           REFFLD(ARBAL)
A           COLHDG('A/R Amt in' 'Order +
A           File')

```

Figure 55 (Part 1 of 2). Example of a Field Reference File

Externally Described Disk Files

```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A          LSTAMT      R          REFFLD(ARBAL)
A          COLHDG('Last' 'Amount' 'Paid')
A          TEXT('Last Amount Paid in A/R')
A          LSTDAT      R          REFFLD(BASDAT)
A          COLHDG('Last' 'Date' 'Paid')
A          TEXT('Last Date Paid in A/R')
A          CRDLMT      R          REFFLD(ARBAL)
A          COLHDG('Credit' 'Limit')
A          TEXT('Customer Credit Limit')
A          SLSYR       R+    2     REFFLD(ARBAL)
A          COLHDG('Sales' 'This' 'Year')
A          TEXT('Customer Sales This Year')
A          SLSLYR     R+    2     REFFLD(ARBAL)
A          COLHDG('Sales' 'Last' 'Year')
A          TEXT('Customer Sales Last Year') 7

```

Figure 55 (Part 2 of 2). Example of a Field Reference File

This example of a field-reference file shows the definitions of the fields that are used by the CUSMSTL (customer master logical) file as shown in Figure 54 on page 114. The field-reference file normally contains the definitions of fields that are used by other files. The following text describes some of the entries for this field-reference file.

- 1 The BASDAT field is edited by the Y edit code, as indicated by the keyword EDTCDE(Y). If this field is used in an externally described output file for an RPG/400 program, the edit code used is the one specified in this field-reference file; it cannot be overridden in the RPG/400 program. If the field is used in a program-described output file for an RPG/400 program, an edit code must be specified for the field in the output specifications.
- 2 The CHECK(MF) entry specifies that the field is a mandatory fill field when it is entered from a display work station. Mandatory fill means that all characters for the field must be entered from the display work station.
- 3 The ADDR and CITY fields share the same attributes that are specified for the NAME field, as indicated by the REFFLD keyword.
- 4 The RANGE keyword, which is specified for the CUSTYP field, ensures that the only valid numbers that can be entered into this field from a display work station are 1 through 5.
- 5 The COLHDG keyword provides a column head for the field if it is used by the Interactive Database Utilities (IDU).
- 6 The ARBAL field is edited by the J edit code, as indicated by the keyword EDTCDE(J).
- 7 A text description (TEXT keyword) is provided for some fields. The TEXT keyword is used for documentation purposes and appears in various listings.

Valid Keys for a Record or File

For a keyed-sequence access path, you can define one or more fields in the DDS to be used as the key fields for a record format. (These fields must not be floating-point fields.) All record types in a file do not have to have the same key fields. For example, an order header record can have the ORDER field defined as the key field, and the order detail records can have the ORDER and LINE fields defined as the key fields.

The key for a file is determined by the valid keys for the record types in that file. The file's key is determined in the following manner:

- If all record types in a file have the same number of key fields defined in the DDS that are identical in attributes, the *key for the file* consists of all fields in the key for the record types. (The corresponding fields do not have to have the same name.) For example, if the file has three record types and the key for each record type consists of fields A, B, and C, the file's key consists of fields A, B, and C. That is, the file's key is the same as the records' key.
- If all record types in the file do not have the same key fields, the key for the file consists of the key fields *common* to all record types. For example, a file has three record types and the key fields are defined as follows:
 - REC1 contains key field A.
 - REC2 contains key fields A and B.
 - REC3 contains key fields A, B, and C.

The file's key is field A—the key field common to all record types.

- If no key field is common to all record types, there is no key for the file.

In an RPG/400 program, you can specify a search argument on certain file operation codes to identify the record you want to process. The RPG/400 program compares the search argument with the key of the file or record, and processes the specified operation on the record whose key matches the search argument.

Valid Search Arguments

You can specify a search argument in the RPG/400 operations CHAIN, DELET, READE, REDPE, SETGT, and SETLL that specify a file name or a record name.

For an operation to a file name, the maximum number of fields that you can specify in a search argument is equal to the total number of key fields valid for the file's key. For example, if all record types in a file do not contain all of the same key fields, you can use a key list (KLIST) to specify a search argument that is composed only of the number of fields common to all record types in the file. If a file contains three record types, the key fields are defined as follows:

- REC1 contains key field A.
- REC2 contains key fields A and B.
- REC3 contains key fields A, B, and C.

The search argument can only be a single field with attributes identical to field A because field A is the only key field common to all record types.

For an operation to a record name, the maximum number of fields that you can specify in a search argument is equal to the total number of key fields valid for that record type.

Externally Described Disk Files

If the search argument consists of one field, you can specify a literal, a field name, or a KLIST name with one KFLD. If the search argument is composed of more than one field (a composite key), you must specify a KLIST with multiple KFLDs.

The attributes of each field in the search argument must be identical to the attributes of the corresponding field in the file or record key. The attributes include the length, the data type (character or numeric), and the number of decimal positions. The attributes are listed in the key-field-information data table of the compiler listing. See the example in Chapter 2, "Entering RPG/400 Specifications."

In all these file operations (CHAIN, DELET, READE, REDPE, SETGT, and SETLL), you can also specify a search argument that contains fewer than the total number of fields valid for the file or record. Such a search argument refers to a partial key.

Referring to a Partial Key

The rules for the specification of a search argument that refers to a partial key are as follows:

- The search argument is composed of fields that correspond to the leftmost (high-order) fields of the key for the file or record.
- Only the rightmost fields can be omitted from the key list (KLIST) for a search argument that refers to a partial key. For example, if the total key for a file or record is composed of key fields A, B, and C, the valid search arguments that refer to a partial key are field A, and fields A and B.
- Each field in the search argument must be identical in attributes to the corresponding key field in the file or record. The attributes include the length, data type (character or numeric), the number of decimal positions, and format (for example, packed or zoned).
- A search argument cannot refer to a portion of a key field.

If a search argument refers to a partial key, the file is positioned at the first record that satisfies the search argument or the record retrieved is the first record that satisfies the search argument. For example, the SETGT and SETLL operations position the file at the first record on the access path that satisfies the operation and the search argument. The CHAIN operation retrieves the first record on the access path that satisfies the search argument. The DELET operation deletes the first record on the access path that satisfies the search argument. The READE operation retrieves the next record if the portion of the key of that record (or the record of the specified type) on the access path matches the search argument. The REDPE operation retrieves the prior record if the portion of the key of that record (or the record of the specified type) on the access path matches the search argument. For more information on the above operation codes, see the *RPG/400 Reference*.

Processing Methods for Externally Described DISK Files

You can process externally described DISK files sequentially by key, randomly by key, randomly by relative record number, sequentially within limits, or consecutively (without a key or relative record number). A K in position 31 of the file description specifications for an externally described file indicates that the file is to be processed by key. If processing is sequential, records are retrieved in key sequence. If processing is random, key values are used to identify the records. A blank in position 31 indicates that the file is processed by relative record number, sequentially (arrival sequence) or randomly. Random or sequential processing is determined by the entries in positions 16 and 28 of the file description specifications and the oper-

ation code used on the calculation specifications (for example, CHAIN, SETLL, READ).

Program-Described Disk Files

Program-described files, which are identified by an F in position 19 of the file description specifications, can be described as indexed files, as sequential files, or as record-address files.

Indexed File

An indexed file is a program-described DISK file whose access path is built on key values. You must create the access path for an indexed file by using data description specifications.

An indexed file is identified by an I in position 32 of the file description specifications.

The key fields identify the records in an indexed file. You specify the length of the key field in positions 29 and 30, the format of the key field in position 31, and the starting location of the key field in positions 35 through 38 of the file description specifications.

An indexed file can be processed sequentially by key, sequentially within limits, or randomly by key.

Valid Search Arguments

For a program-described file, a search argument must be a single field. For the CHAIN and DELET operations, the search argument must be the same length as the key field that is defined on the file description specifications for the indexed file. For the other file operations, the search argument may be a partial field.

The DDS specifies the fields to be used as a key field. Positions 35 through 38 of the file description specifications specify the starting position of the first key field. The entry in positions 29 and 30 of the file description specifications must specify the total length of the key as defined in the DDS.

Figure 56 and Figure 57 show examples of how to use the DDS to describe the access path for indexed files.

Program-Described Disk Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++.Len++TDpB.....Functions+++++*****
A          R FORMATA                                PFILE(ORDDTLP)
A          TEXT('Access Path for Indexed +
A          File')
A          FLDA          14
A          ORDER        5  0
A          FLDB          101
A          K ORDER
A*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORDDTLL IP F 118 3PI 15 DISK
F*
```

Figure 56. Using Data Description Specifications to Define the Access Path for an Indexed File

You must use data description specifications to create the access path for a program-described indexed file.

In the DDS for the record format FORMATA for the logical file ORDDTLL, the field ORDER, which is five digits long, is defined as the key field, and is in packed format. The definition of ORDER as the key field establishes the keyed access for this file. Two other fields, FLDA and FLDB, describe the remaining positions in this record as character fields.

The program-described input file ORDDTLL is described on the file description specifications as an indexed file. Positions 29 and 30 must specify the number of positions in the record required for the key field as defined in the DDS: three positions. Positions 35 through 38 specify position 15 as the starting position of the key field in the record. Because the file is defined as program-described by the F in position 19, the RPG/400 compiler does not retrieve the external field-level description of the file at compilation time. Therefore, you must describe the fields in the record on the input specifications.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++.Len++TDpB.....Functions+++++*****
A          R FORMAT                                PFILE(ORDDTLP)
A          TEXT('Access Path for Indexed +
A          File')
A          FLDA                                14
A          ORDER                                5
A          ITEM                                 5
A          FLDB                                96
A          K ORDER
A          K ITEM
A*

```

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORDDTLL IP F      120 10AI      15 DISK
F*

```

Figure 57. (Part 1 of 2). Using Data Description Specifications to Define the Access Path (Composite Key) for an Indexed File

In this example, the data description specifications define two key fields for the record format FORMAT in the logical file ORDDTLL. For the two fields to be used as a composite key for a program described indexed file, the key fields must be contiguous in the record.

On the file description specifications, the length of the key field is defined as 10 in positions 29 and 30 (the combined number of positions required for the ORDER and ITEM fields). The starting position of the key field is described as 15 in positions 37 and 38. The starting position must specify the first position of the first key field.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IDSname....NODsExt-file++.....OccrLen+.....*
IKEY          DS
I.....Ext-field+.....PFromTo++DField+.....*
I              1   5 K1
I              6  10 K2
I*

```

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C              MOVE ORDER      K1
C              MOVE ITEM       K2
C          KEY      CHAINORDDTLL          99
C*

```

Figure 58. (Part 2 of 2). Using Data Description Specifications to Define the Access Path (Composite Key) for an Indexed File

Program-Described Disk Files

When the DDS specifies a composite key, you must build a search argument in the program to CHAIN to the file. (A KLIST cannot be used for a program-described file.) One way is to create a data structure with subfields equal to the key fields defined in the DDS. Then, in the calculations, set the subfields equal to the value of the key fields, and use the data-structure name as the search argument in the CHAIN operation.

In this example, the MOVE operations set the subfields K1 and K2 equal to the value of ORDER and ITEM, respectively. The data-structure name (KEY) is then used as the search argument in the CHAIN operation.

Sequential Files

Sequential files are files where the order of the records in the file is based on the order the records are placed in the file (that is, in arrival sequence). For example, the tenth record placed in the file occupies the tenth record position.

Sequential files can be processed randomly by relative record number, consecutively, or by a record-address file. You can use either the SETLL or SETGT operation code to set limits on the file.

Record Address File

You can use a record-address file to process another file. A record-address file can contain (1) limits records that are used to process a file sequentially within limits, or (2) relative record numbers that are used to process a file by relative record numbers. The record-address file itself must be processed sequentially.

A record-address file is identified by an R in position 16 of the file description specifications. If the record-address file contains relative record numbers, position 32 must contain a T. The name of the record-address file must also be specified in positions 11 through 18 of the extension specifications, and the name of the file to be processed by the record-address file must be specified in positions 19 through 26 of the extension specifications.

Limits Records

For sequential-within-limits processing, the record-address file contains limits records. A limits record contains the lowest record key and the highest record key of the records in the file to be read.

The format of the limits records in the record-address file is as follows:

- The low key begins in position 1 of the record; the high key immediately follows the low key. No blanks can appear between the keys.
- Each record in the record-address file can contain only one set of limits. The record length must be greater than or equal to twice the length of the record key.
- The low key and the high key in the limits record must be the same length. The length of the keys must be equal to the length of the key field of the file to be processed.
- A blank entry equal in length to the record key field causes the RPG/400 compiler to read the next record in the record-address file.

Relative Record Numbers

For relative-record-number processing, the record-address file contains relative record numbers. Each record retrieved from the file being processed is based on a relative record number in the record-address file. A record-address file containing relative record numbers cannot be used for limits processing. Each relative record number in the record-address file is a multi-byte binary field where each field contains a relative record number. You can specify the record-address file length as 4, 3, or blank, depending on the source of the file. When using a record-address file from the AS/400 environment, specify the record-address file length as 4, since each field is 4 bytes in length. When using a record-address file from the System/36 environment, specify the record-address file length as 3, since each field is 3 bytes in length. If you specify the record-address file length as blank, the compiler will check the primary record length at run time and determine whether to treat the record-address file as 3 byte or as 4 byte. A minus 1 (-1 or hexadecimal FFFFFFFF) relative-record-number value stops the use of a relative-record-address file record. End of file occurs when all records from the record-address file have been processed.

Externally Described File as Program Described

A file that is externally described can be treated as a program-described file in an RPG/400 program. Specify an F in position 19 of the file description specifications, and describe the fields in the records on input and/or output specifications. When an F is specified in position 19 of the file description specifications for an externally described file, the compiler does not copy in the external description.

Methods for Processing Disk Files

The methods of disk file processing include:

- Relative-record-number processing
- Consecutive processing
- Sequential-by-key processing
- Random-by-key processing
- Sequential-within-limits processing.

Relative-Record-Number Processing

Random input or update processing by relative record number applies to full procedural files only. The desired record is accessed by the CHAIN operation code.

Relative record numbers identify the positions of the records relative to the beginning of the file. For example, the relative record numbers of the first, fifth, and seventh records are 1, 5, and 7, respectively.

For an externally described file, input or update processing by relative record number is determined by a blank in position 31 of the file description specifications and the use of the CHAIN operation code. Output processing by relative record number is determined by a blank in position 31 and the use of the RECNO option on a file description specifications continuation line for the file.

You can use the RECNO option for the file description specifications continuation line to specify a numeric field that contains the relative record number that specifies where a new record is to be added to this file. The RECNO field must be defined as numeric with zero decimal positions. The field length must be large enough to

Methods for Processing Disk Files

contain the largest record number for the file. A RECNO field must be specified if new records are to be placed in the file by using output specifications or a WRITE operation. When you update or add a record to a file by relative record number, the record must already have a place in the member. For an update, that place can be a valid existing record; for a new record, that place can be a deleted record. You can use the CL command INZPFM to initialize records for use by relative record number. The current relative record number is placed in the RECNO field for all retrieval operations or operations that reposition the file (for example, SETLL, CHAIN, READ).

Consecutive Processing

During consecutive processing, records are read in the order they appear in the file.

For output and input files that do not use random functions (such as SETLL, SETGT, CHAIN, or ADD), the RPG/400 compiler defaults to or operates as though SEQONLY(*YES) had been specified on the CL command OVRDBF (Override with Database File). (The RPG/400 compiler does not operate as though SEQONLY(*YES) had been specified for update files.) SEQONLY(*YES) allows multiple records to be placed in internal data management buffers; the records are then passed to the RPG/400 compiler one at a time on input. If, in the same job, two logical files use the same physical file, and one file is processed consecutively and one is processed for random update, a record could be updated that has already been placed in the buffer that is presented to the program. In this case, when the record is processed from the consecutive file, the record does not reflect the updated data. To prevent this problem, use the CL command OVRDBF and specify the option SEQONLY(*NO), which indicates that you do not want multiple records transferred for a consecutively processed file.

For more information on sequential only processing, see the *Database Guide*.

Sequential-by-Key Processing

For the sequential-by-key method of processing, records are read from the file in key sequence.

The sequential-by-key method of processing is valid for keyed files used as primary, secondary, or full procedural files.

For output files and for input files that do not use random functions (such as SETLL, SETGT, CHAIN, or ADD) and that have only one record format, the RPG/400 compiler defaults to or operates as though SEQONLY(*YES) had been specified on the CL command OVRDBF. (The RPG/400 compiler does not operate as though SEQONLY(*YES) had been specified for update files.) SEQONLY(*YES) allows multiple records to be placed in internal data management buffers; the records are then passed to the RPG/400 compiler one at a time on input. If, in the same job, two files use the same physical file, and one file is processed sequentially and one is processed for random update, a record could be updated that has already been placed in the buffer that is presented to the program. In this case, when the record is processed from the sequential file, the record does not reflect the updated data. To prevent this problem, use the CL command OVRDBF and specify the option SEQONLY(*NO), which indicates that you do not want multiple records transferred for a sequentially processed file.

For more information on sequential only processing, see the *Database Guide*.

Figure 59 on page 125 shows different ways a header record and the detail records associated with the header record can be processed. Part 1 shows an example of the file being read sequentially by key; parts 2 through 4 show examples in which the READ operation code is used; part 5 shows the processing of these records by the matching record technique.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* In this example, the order header record (ORDHDR) and the order
F* detail record (ORDDTL) are contained in the same file (ORDFIL).
F* The ORDFIL file is defined as a primary input file and is read
F* sequentially by key. In the data description specifications for
F* the file, the key for the ORDHDR record is defined as the ORDER
F* field, and the key for the ORDDTL record is defined as the ORDER
F* field plus the LINE (line number) field, which is a composite key.
F*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORDFILL IP E K DISK
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* A record-identifying indicator is assigned to each record; these
I* record-identifying indicators are used to control processing for
I* the different record types.
IRcdname+....In.....*
IORDHDR 01
I*
IORDDTL 02
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C *IN01 IFEQ '1'
C "
C " Process header
C "
C END
C*
C *IN02 IFEQ '1'
C "
C " Process detail
C "
C END

```

Figure 59 (Part 1 of 7). Processing Order Header and Order Detail Records

Methods for Processing Disk Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* This example is the same as the previous example except that the
F* ORDFIL file is defined as a full-procedural file, and the reading
F* of the file is done by the READ operation code.
F*
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FORDFILL IF E K DISK
F*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The two records (ORDHDR and ORDDTL) are contained in the same
I* file, and a record-identifying indicator is assigned to each
I* record. The record-identifying indicators are used to control
I* processing for the different record types. No control levels
I* or match fields can be specified for a full-procedural file.
I*
IRcdname+....In.....*
```

IORDHDR	01
IORDDTL	02

```
I*
```

Figure 59 (Part 2 of 7). Processing Order Header and Order Detail Records


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The READ operation code reads a record from the ORDFIL file. An
C* end-of-file indicator is specified in positions 58 and 59. If
C* the end-of-file indicator 99 is set on by the READ operation,
C* the program branches to the EOFEND tag and processes the end-of-
C* file routine. The record-identifying indicators control the
C* processing of the different record types.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          READ ORDFIL          99
C 99          GOTO EOFEND
C*
C          *IN01          IFEQ '1'
C          "
C          "          Process header
C          "
C          END
C*
C          *IN02          IFEQ '1'
C          "
C          "          Process detail
C          "
C          END
C*
C          EOFEND          TAG
C          "
C          "          End-of-file routine
C          "

```

Figure 59 (Part 3 of 7). Processing Order Header and Order Detail Records

Methods for Processing Disk Files

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* This example is similar to the one shown in Part 2 of this figure.
F* However, the READ operation code is used to read each record
F* (ORDHDR and ORDDTL) instead of reading the file. The program
F* logic controls when each READ occurs. No record-identifying
F* indicators are needed because the program logic knows which
F* record it is working with according to the record format name.
F*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORDFILL IF E K DISK
F*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C
C
C
C
C          READ ORDHDR          99
C 99          GOTO END
C
C          "
C          "
C          "
C*
C          READ ORDDTL          99
C 99          GOTO END
C
C          "
C          "
C          "
C*
C          END          TAG
C
C
C
C

```

Process header

Process detail

Figure 59 (Part 4 of 7). Processing Order Header and Order Detail Records

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* In this example, the order header records (ORDHDR) are contained
F* in the ORDHDRL file, and the order detail records (ORDDTL) are
F* contained in the ORDDTLL file. The ORDHDRL is defined as a
F* primary input file, and the reading of records from the file is
F* controlled by the program cycle. The ORDDTLL file is defined as
F* a full-procedural file, and the READE operation is used to read
F* records from the file.
F*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORDDTLL IF E K DISK
FORDHDRL IP E K DISK
F*
```

Figure 59 (Part 5 of 7). Processing Order Header and Order Detail Records

Methods for Processing Disk Files

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The ORDER field in the SETLL operation is used to position the
C* ORDDTLL file at the first ORDDTL record that has a key equal to
C* or greater than the contents of the ORDER field. The ORDER
C* field is used as the search argument for the READE operation.
C* The READE operation retrieves the next ORDDTL record from the
C* file if the key of the record is equal to the search argument
C* specified in factor 1. If the key and the search argument are
C* not equal, the indicator specified in positions 58 and 59 is
C* set on.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C
C          "
C          "      Process header
C          "
C          "
C          ORDER      SETLLORDDTL          20
C N20          GOTO NONE
C          LOOP      TAG
C          ORDER      READEORDDTL          21
C 21          GOTO ENDFIL
C          "
C          "
C          "      Process detail
C          "
C          "
C          GOTO LOOP
C          NONE      TAG
C          "
C          "
C          ENDFIL    TAG

```

Figure 59 (Part 6 of 7). Processing Order Header and Order Detail Records

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* In this example, the order header records (ORDHDR) are contained
F* in the ORDHDRL file, and the order detail records (ORDDTL) are
F* contained in the ORDDTLL file. The ORDHDRL is defined as a
F* primary input file, and the ORDDTLL file is defined as a
F* secondary input file. The order header and order detail records
F* are processed as matching record, with the ORDER field in both
F* records assigned the match level value of M1. Record-identifying
F* indicators 01 and 02 are assigned to the records to control the
F* processing for the different record types.
F*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FORHDRL IP E K DISK
FORDDTLL IS E K DISK
F*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IRcdname+....In.....*
IORHDR 01
I ORDER M1
IORDDL 02
I ORDER M1
I*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpdcFactor2+++ResultLenDHHiLoEqComments+++++*
C 01NMR " [Process header]
C 01 MR " [Process header]
C 01 MR " [Process header]
C*
C 02NMR " [Process detail]
C 02 MR " [Process detail]
C 02 MR " [Process detail]
```

Figure 59 (Part 7 of 7). Processing Order Header and Order Detail Records

Sequential-within-Limits Processing

Sequential-within-limits processing by a record-address file is specified by an L in position 28 of the file description specifications and is valid for a file with a keyed access.

You can specify sequential-within-limits processing for an input or an update file that is designated as a primary, secondary, or full-procedural file. The file can be externally described or program-described (indexed). The file should have keys in ascending sequence.

To process a file sequentially within limits from a record-address file, the program reads:

- A limits record from the record-address file
- Records from the file being processed within limits with keys greater than or equal to the low-record key and less than or equal to the high-record key in the limits record. If the two limits supplied by the record-address file are equal, only the records with the specified key are retrieved.

The program repeats this procedure until the end of the record-address file is reached.

Figure 61 on page 134 shows an example of an indexed file being processed sequentially within limits. Figure 62 on page 136 shows the same example with externally described files instead of program-described files.

Keyed Processing Examples

Figure 63 on page 136 shows an example of processing certain records in a group. Figure 64 on page 139 shows examples of how to process the first record in a file and the last record in a file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FCHANGE IP E K DISK
FMASTER UF E K DISK
F*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IRcdname+....In.....*
IMSTREC 01
ICHGREC 02
I*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C 02 ACCT CHAINMSTREC 03
C 02N03 MOVE NEW NAMADR
C 02N03 UPDATMSTREC
C*
```

Figure 60. Random Processing of an Externally Described DISK File by Key

The update file MASTER is to be processed by keys. The DDS for each of the externally described files (MASTER and CHANGE) identify the ACCT field as the key field. As each record is read from the primary input file, CHANGE, the account number field (ACCT) is used as the search argument to chain to the corresponding record in the MASTER file. Input specifications are used to assign record-identifying indicators to the records in the CHANGE and MASTER files. The MASTER file contains one record format MSTREC that contains two fields, ACCT and NAMADR (name and address). The CHANGE file contains one record format CHGREC that contains two fields, ACCT and NEW. The data in the NEW field must be moved into the NAMADR field before the MSTREC can be updated.

Methods for Processing Disk Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* The input file MASTER, which is a program-described file (F in
F* position 19), is described as an indexed file to be processed
F* by keys. (The access path for an indexed file must be created
F* by data description specifications.)
F*
F* MASTER is processed sequentially within limits (L in position 28)
F* by the record address file LIMITS. Each set of limits from the
F* record-address file consists of the low and high account numbers
F* of the records in the MASTER file to be processed. Because the
F* account number key field (ACCT) is eight positions long, each
F* set of limits consists of two 8-position keys.
F*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FLIMITS IR F 16 8 EDISK
FMASTER IP F 64L 8AI 1 DISK
FPRINT 0 F 96 OF PRINTER
F*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* The record-address file name LIMITS must be specified in positions
E* 11 through 18 of the extension specifications. The name of the
E* file to be processed by the record address file must be specified
E* in positions 19 through 26 of the extension specifications.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E LIMITS MASTER
E*
```

Figure 61 (Part 1 of 2). Processing an Indexed File Sequentially within Limits


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* Input specifications must be used to describe the records in the
I* program-described file MASTER.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IMASTER NS 01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I 1 8 ACCT
I 9 64 NAMADR
I*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
O*
O* As MASTER is processed within each set of limits, the corres-
O* ponding records are printed. Processing of the MASTER file is
O* complete when the record-address file LIMITS reaches end of file.
O*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT D 1 01
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
O ACCT 8
O NAMADR 70
O*

```

Figure 61 (Part 2 of 2). Processing an Indexed File Sequentially within Limits

Methods for Processing Disk Files

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FLIMITS IR F 16 8 EDISK
FMASTER IP E L K DISK
FPRINT 0 F 96 OF PRINTER
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IRcdname+....In.....*
IMSTREC 01
I*
```

Figure 62. Processing an Externally Described File Sequentially within Limits

If the program shown in Figure 61 on page 134 used externally described files, the file description specifications would be coded as shown above. The input specifications are used to assign a record-identifying indicator to the record in the externally described file. The MASTER file contains the record format MSTREC. The external descriptions for the file identify the key fields. These keys should be in ascending sequence.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* This example shows how to retrieve the first record of a group.
C* The SETLL operation is used to position the file at the first
C* ORDDTL record that has a key equal to or greater than the search
C* argument contained in the ORDER field. The READE operation reads
C* the next ORDDTL record from the file if the key of the record is
C* equal to the search argument (the ORDER field) specified in
C* factor 1. If the key is not equal to the search argument, the
C* indicator specified in positions 58 and 59 is set on.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C ORDER SETLLORDDTL
C ORDER READEORDDTL 22
C 22 GOTO ENDFIL
C "
C "
C " Process ORDDTL
C "
```

Figure 63 (Part 1 of 5). Processing Certain Records in a Group

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* This example shows how to retrieve the last record of a group.
C* The SETGT operation is used to position the file at the next
C* ORDDTL record that has a key greater than the search argument
C* contained in the ORDER field. The REDPE operation reads the next
C* prior ORDDTL record from the file if the key of the record is
C* equal to the search argument (the ORDER field) specified in
C* factor 1. If the key is not equal to the search argument, the
C* indicator specified in positions 58 and 59 is set on.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          ORDER      SETGTORDDTL
C          ORDER      REDPEORDDTL          22
C   22          GOTO ENDFIL
C          "
C          "
C          "
C          "
C          "
C          "

```

Process ORDDTL

Figure 63 (Part 2 of 5). Processing Certain Records in a Group

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* This example shows how to retrieve the last record of the previous
C* group. The ORDER field, which contains the key of the current
C* group, is used in the SETLL operation to position the file at the
C* first ORDDTL record that has a key equal to or greater than the
C* search argument contained in the ORDER field. The READP operation
C* then reads the prior record. If there is no prior record in the
C* file, the program branches to the ENDFIL routine.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          ORDER      SETLLORDDTL
C          ORDER      READPORDDTL          22
C   22          GOTO ENDFIL
C          "
C          "
C          "
C          "
C          "
C          "

```

Process ORDDTL

Figure 63 (Part 3 of 5). Processing Certain Records in a Group

Methods for Processing Disk Files

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* This example retrieves the last record of a group. One or more
C* records for the group exist in the file. The SETGT operation
C* positions the file at the next record that contains a key value
C* greater than the search argument contained in the ORDER field.
C* For example, if the ORDER field contains a value of 10, SETGT
C* positions the file at the record that contains a key value
C* greater than 10:
C*
C* Keys
C* 9
C* 9
C* 10
C* 10
C* SETGT →
C* 11
C*
C* The READP operation then reads the prior record of the ORDDTL
C* record format, thus reading the last record of the previous group.
C* READP requires an end-of-file indicator in positions 58 and 59;
C* therefore, if the beginning of the file is encountered, the halt
C* indicator H6 is set on and the program ends abnormally.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C           ORDER      SETGTORDDTL
C           READPORDDTL                                H6
C  H6           RETRN
C
C           "
C           "
C           "
C           "

```

Process ORDDTL

Figure 63 (Part 4 of 5). Processing Certain Records in a Group

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* Reading the first record of the next group requires the SETGT
C* operation to position the file and the READ operation. The ORDER
C* field, which contains the key of the current group, is specified
C* in factor 1 of the SETGT operation. The READ operation is then
C* used to read the first record of the next group. An indicator
C* must be specified in positions 58 and 59 of the READ operation to
C* test for end of file. This technique can be used if the program
C* knows the key value for a group of records or for a specific
C* record and wants the next group. SETGT can be used to eliminate
C* reading unwanted records that would be bypassed.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          ORDER      SETGTORDDTL
C          READ ORDDTL                22
C  22          GOTO ENDFIL
C          "
C          "
C          "      

|                |
|----------------|
| Process ORDDTL |
|----------------|


C          "

```

Figure 63 (Part 5 of 5). Processing Certain Records in a Group

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* After the file is opened, the first record is retrieved by a
C* subsequent READ operation. To access the first record in a file
C* after some processing has been done, use the figurative constant
C* *LOVAL (assuming ascending key sequence). Set the lower limits
C* by using the constant with the SETLL operation.
C* Use the READ operation for the next record of the ORDDTL record
C* format. If no records exist, end of file occurs, and the
C* program branches to the NONE routine.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *LOVAL     SETLLORDDTL
C          READ ORDDTL                22
C  22          GOTO NONE
C          "
C          "
C          "      

|                |
|----------------|
| Process ORDDTL |
|----------------|


C          "

```

Figure 64 (Part 1 of 2). Processing Certain Records in a File

Valid File Operations

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C* Use the figurative constant *HIVAL (assuming ascending key
C* sequence to access the last record in a file. By using *HIVAL
C* with a SETGT operation, the file is positioned at the next
C* record that has a key field value greater than the value
C* specified in factor 1.
C* The READP operation reads the next prior record, which in this
C* example is the last record in the file.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *HIVAL   SETGTORDDTL
C          READPORDDTL                22
C  22          GOTO NONE
C          "
C          "
C          "
C          "
C          " 

|                |
|----------------|
| Process ORDDTL |
|----------------|


```

Figure 64 (Part 2 of 2). Processing Certain Records in a File

Valid File Operations

Figure 65 on page 141 shows the valid file operation codes allowed for DISK files processed by keys and Figure 66 on page 142 for DISK files processed by non-keyed methods. The operations shown in these figures are valid for externally described DISK files and program-described DISK files.

Before running your program, you can override a file to another file. In particular, you can override a sequential file in your program to an externally described, keyed file. (The file is processed as a sequential file.) You can also override a keyed file in your program to another keyed file, providing the key fields are compatible. For example, the overriding file must not have a shorter key field than you specified in your program.

File-Description Specifications Positions					Calculation Specifications Positions
15	16	28 ¹	31 ²	66	28-32
I	P/S		K/A/P		CLOSE, FEOD, FORCE
I	P/S		K/A/P	A	WRITE, CLOSE, FEOD, FORCE
I	P/S	L	K/A/P		CLOSE, FEOD, FORCE
U	P/S		K/A/P		UPDAT, DELET, CLOSE, FEOD, FORCE
U	P/S		K/A/P	A	UPDAT, DELET, WRITE, CLOSE, FEOD, FORCE
U	P/S	L	K/A/P		UPDAT, CLOSE, FEOD, FORCE
I	F		K/A/P		READ, READE, REDPE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F		K/A/P	A	WRITE, READ, REDPE, READE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F	L	K/A/P		READ, OPEN, CLOSE, FEOD
U	F		K/A/P		READ, READE, REDPE, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
U	F		K/A/P	A	WRITE, UPDAT, DELET, READ, READE, REDPE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
U	F	L	K/A/P		READ, UPDAT, OPEN, CLOSE, FEOD
O	Blank		K/A/P	A	WRITE (add new records to a file), OPEN, CLOSE, FEOD
O	Blank		K/A/P		WRITE (initial load of a new file) ³ , OPEN, CLOSE, FEOD
Note: ¹ An L must be specified in position 28 to specify sequential-within-limits processing by a record-address file for an input or an update file.					
Note: ² Externally described files require a K in position 31; program-described files require an A or P in position 31 and an I in position 32.					
Note: ³ An A in position 66 is not required for the initial loading of records into a new file. If A is specified in position 66, ADD must be specified on the output specifications. The file must have been created with the OS/400 CREATE FILE command.					

Figure 65. Valid File Operations for Keyed Processing Methods (Random by Key, Sequential by Key, Sequential within Limits)

Valid File Operations

File-Description Specifications Positions					Calculation Specifications Positions
15	16	31	54-59	66	28-32
I	P/S	Blank			CLOSE, FEOD, FORCE
I	P/S	Blank	RECNO		CLOSE, FEOD, FORCE
U	P/S	Blank			UPDAT, DELET, CLOSE, FEOD, FORCE
U	P/S	Blank	RECNO		UPDAT, DELET, CLOSE, FEOD, FORCE
I	F	Blank			READ, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F	Blank	RECNO		READ, READP, SETLL, SETGT,
U	F	Blank			READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
U	F	Blank	RECNO		READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
I	R	A/P/ Blank ¹			OPEN, CLOSE, FEOD
I	R	Blank ²			OPEN, CLOSE, FEOD
O	Blank	Blank	RECNO	A	WRITE ³ (add records to a file), OPEN, CLOSE, FEOD
O	Blank	Blank	RECNO		WRITE ⁴ (initial load of a new file), OPEN, CLOSE, FEOD
O	Blank	Blank	Blank		WRITE (sequentially load or extend a file), OPEN, CLOSE, FEOD
Note: ¹ If position 31 is blank for a record-address-limits file, the format of the keys in the record-address file is the same as the format of the keys in the file being processed.					
Note: ² A record-address file containing relative record numbers requires a T in position 32.					
Note: ³ The RECNO field that contains the relative record number must be set prior to the WRITE operation or if ADD is specified on the output specifications.					
Note: ⁴ An A in position 66 is not required for the initial loading of the records into a new file; however, if A is specified in position 66, ADD must be specified on output specifications. The file must have been created with the OS/400 CREATE FILE command.					

Figure 66. Valid File Operations for Non-keyed Processing Methods (Sequential, Random by Relative Record Number, and Consecutive)

Chapter 8. Using WORKSTN Files

The WORKSTN file allows an RPG/400 program to communicate interactively with a work-station user or to use the Intersystem Communications Function (ICF) to communicate with other programs. This chapter describes:

- Intersystem Communications Function (ICF)
- Externally described WORKSTN files
- Program-described WORKSTN files
- Multiple-device files.

The chapter also includes a number of examples for using WORKSTN files.

Intersystem Communications Function

You can use the ICF to write programs that communicate with (send data to and receive data from) other application programs on other systems.

To use the ICF, define a WORKSTN file in your program that refers to an ICF device file. Use either the system supplied file QICDMF or a file created using the OS/400 command CRTICFF.

You code for ICF by using the ICF as a file in your program. The ICF is similar to a display file and it contains the communications formats required for the sending and receiving of data between systems.

For further information on the ICF, refer to the *ICF Programmer's Guide*.

Externally Described WORKSTN Files

An RPG/400 WORKSTN file can use an externally described display-device file or ICF device file, which contains file information and a description of the fields in the records to be written.

In addition to the field descriptions (such as field names and attributes), the DDS for a display-device file are used to:

- Format the placement of the record on the screen by specifying the line-number and position-number entries for each field and constant.
- Specify attention functions such as underlining and highlighting fields, reverse image, or a blinking cursor.
- Specify validity checking for data entered at the display work station. Validity-checking functions include detecting fields where data is required, detecting mandatory fill fields, detecting incorrect data types, detecting data for a specific range, checking data for a valid entry, and processing modules 10 or 11 check-digit verification.
- Control screen management functions, such as if fields are to be erased, overlaid, or kept when new data is displayed.
- Associate indicators 01 through 99 with command attention keys or command function keys. If a function key is described as a command function key (CF), both the response indicator and the data record (with any modifications entered

Externally Described WORKSTN Files

on the screen) are returned to the program. If a function key is described as a command attention key (CA), the response indicator is returned to the program but the data record remains unmodified. Therefore, input-only character fields are blank and input-only numeric field are filled with zeros, unless these fields have been initialized otherwise.

- Assign an edit code (EDTCDE) or edit word (EDTWRD) keyword to a field to specify how the field's values are to be displayed.
- Specify subfiles.

A display-device-record format contains three types of fields:

- *Input fields.* Input fields are passed from the device to the program when the program reads a record. Input fields can be initialized with a default value. If the default value is not changed, the default value is passed to the program. Input fields that are not initialized are displayed as blanks into which the workstation user can enter data.
- *Output fields.* Output fields are passed from the program to the device when the program writes a record to a display. Output fields can be provided by the program or by the record format in the device file.
- *Output/input (both) fields.* An output/input field is an output field that can be changed. It becomes an input field if it is changed. Output/input fields are passed from the program when the program writes a record to a display and passed to the program when the program reads a record from the display. Output/input fields are used when the user is to change or update the data that is written to the display from the program.

If you specify the keyword INDARA in the DDS for a WORKSTN file, the RPG/400 program passes indicators to the WORKSTN file in a separate indicator area, and not in the input/output buffer.

For a detailed description of an externally described data-device file and for a list of valid DDS keywords, see the *DDS Reference*.

Figure 67 on page 145 shows an example of the DDS for a display-device file.

Processing an Externally Described WORKSTN File

When an externally described WORKSTN file is processed, the OS/400 system transforms data from the program to the format specified for the file and displays the data. When data is passed to the program, the data is transformed to the format used by the program.

The OS/400 system provides device-control information for processing input/output operations for the device. When an input record is requested from the device, the OS/400 system issues the request, and then removes device-control information from the data before passing the data to the program. In addition, the OS/400 system can pass indicators to the program indicating which fields, or if any fields, in the record have been changed.

When the program requests an output operation, it passes the output record to the OS/400 system. The OS/400 system provides the necessary device-control information to display the record. It also adds any constant information specified for the record format when the record is displayed.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLInPosFunctions+++++*****
A** ITEM MASTER INQUIRY
A          REF(DSTREF) 1
A          R PROMPT      TEXT('Item Prompt Format')
A 73N61      OVERLAY 2
A          CA01(98 'End of Program') 3
A          1 2'Item Inquiry'
A          3 2'Item Number'
A          ITEM        R      I 3 15PUTRETAIN 4
A 61          ERRMSG('Invalid Item Number' 61) 5
A          R RESPONSE  TEXT('Response Format')
A          OVERLAY 2
A          LOCK 6
A          5 2'Description'
A          DESCRP      R      5 15
A          5 37'Price'
A          PRICE        R      5 44
A          7 2'Warehouse Location' 7
A          WHSLOC      R      7 22
A          9 2'On Hand'
A          ONHAND      R      9 10
A          9 19'Allocated' 8
A          ALLOC        R      9 30
A          9 40'Available'
A          AVAIL        R      9 51
A*
```

Figure 67. Example of the Data Description Specifications for a Display Device File

This display device file contains two record formats: PROMPT and RESPONSE.

- 1 The attributes for the fields in this file are defined in the DSTREF field reference file.
- 2 The OVERLAY keyword is used so that both record formats can be used on the same display.
- 3 Function key 1 is associated with indicator 98, which is used by the programmer to end the program.

Externally Described WORKSTN Files

- 4** The PUTRETAIN keyword allows the value that is entered in the ITEM field to be kept in the display. In addition, the ITEM field is defined as an input field by the I in position 38. ITEM is the only input field in these record formats. All of the other fields in the record are output fields since position 38 is blank for each of them.
- 5** The ERRMSG keyword identifies the error message that is displayed if indicator 61 is set on in the program that uses this record format.
- 6** The LOCK keyword prevents the work-station user from using the keyboard when the RESPONSE record format is initially displayed.
- 7** The constants such as 'Description', 'Price', and 'Warehouse Location' describe the fields that are written out by the program.
- 8** The line and position entries identify where the fields or constants are written on the display.

When a record is passed to a program, the fields are arranged in the order in which they are specified in the DDS. The order in which the fields are displayed is based on the display positions (line numbers and position) assigned to the fields in the DDS. The order in which the fields are specified in the DDS and the order in which they appear on the screen need not be the same.

Function Key Indicators on Display Device Files

The function key indicators, KA through KN and KP through KY are valid for a program that contains a display device WORKSTN file if the associated function key is specified in the DDS.

The function key indicators relate to the function keys as follows: function key indicator KA corresponds to function key 1, KB to function key 2 . . . KX to function key 23, and KY to function key 24.

Function keys are specified in the DDS with the CFxx (command function) or CAxx (command attention) keyword. For example, the keyword CF01 allows function key 1 to be used. When you press function key 1, function key indicator KA is set on in the RPG/400 program. If you specify the function key as CF01 (99), both function key indicator KA and indicator 99 are set on in the RPG/400 program. If the work-station user presses a function key that is not specified in the DDS, the OS/400 system informs the user that an incorrect key was pressed.

If the work-station user presses a specified function key, the associated function key indicator in the RPG/400 program is set on when fields are extracted from the record (move fields logic) and all other function key indicators are set off. If a function key is not pressed, all function key indicators are set off at move fields time. The function key indicators are set off if the user presses the Enter key.

Command Keys on Display Device Files

You can specify the command keys Help, Roll Up, Roll Down, Print, Clear, and Home in the DDS for a display device file with the keywords HELP, ROLLUP, ROLLDOWN, PRINT, CLEAR, and HOME.

Command keys can be processed by an RPG/400 program whenever the RPG/400 compiler processes a READ or an EXFMT operation on a record format for which the appropriate keywords are specified in the DDS. When the command keys are in effect and a command key is pressed, the OS/400 system returns control to the RPG/400 program. If a response indicator is specified in the DDS for the command selected, that indicator is set on and all other response indicators that are in effect for the record format and the file are set off.

If a response indicator is not specified in the DDS for a command key, the following happens:

- For the Print key without *PGM specified, the print function is processed.
- For the Roll Up and Roll Down keys used with subfiles, the displayed subfile rolls up or down, within the subfile. If you try to roll beyond the start or end of a subfile, you get a run-time error.
- For the Print Key specified with *PGM, Roll Up and Roll Down keys used without subfiles, and for the Clear, Help, and Home keys, one of the *STATUS values 1121-1126 is set, respectively, and processing continues.

Processing WORKSTN Files

This section explains the valid file operation codes for a WORKSTN file.

EXFMT Operation

The EXFMT operation is a combination of a WRITE followed by a READ to the same record format. If you define a WORKSTN file on the file description specifications as a full-procedural (F in position 16) combined file (C in position 15) that uses externally described data (E in position 19) the EXFMT (execute format) operation code can be used to write and read from the display.

READ Operation

The READ operation is valid for a full-procedural combined file or a full-procedural input file that uses externally described data or program-described data. The READ operation retrieves a record from the display. However, a format must exist at the device before any input operations can occur. This requirement can be satisfied on a display device by conditioning an output record with the 1P indicator, by writing the first format to the device from another program, or, if the read is by record-format name, by using the keyword INZRCD on the record description in the DDS.

WRITE Operation

The WRITE operation writes a new record to a display and is valid for a combined file or an output file. Output specifications and the EXCPT operation can also be used to write to a WORKSTN file. See the *RPG/400 Reference* for a complete description of each of these operation codes.

Figure 68 on page 148 shows the valid file operation codes for a WORKSTN file.

Processing WORKSTN Files

File-Description Specifications Positions		Calculation Specifications Positions
15	16	28-32
I	P/S	CLOSE, ACQ, REL, NEXT, POST, FORCE
I	P/S	WRITE ¹ , CLOSE, ACQ, REL, NEXT, POST, FORCE
I	F	READ, OPEN, CLOSE, ACQ, REL, NEXT, POST
C	F	READ, WRITE ¹ , EXFMT ² , OPEN, CLOSE, ACQ, REL, NEXT, POST, UPDAT ³ , CHAIN ³ , READC ³
O	Blank	WRITE ¹ , OPEN, CLOSE, ACQ, REL, POST
Note: ¹ The WRITE operation is not valid for a program-described file used with a format name.		
Note: ² If the EXFMT operation is used, the file must be externally described (an E in position 19 of the file description specifications).		
Note: ³ For subfile record formats, the UPDAT, CHAIN, and READC operations are also valid.		

Figure 68. Valid File Operation Codes for a WORKSTN File

WORKSTN file

Figure 69 on page 149 is a processing chart for WORKSTN files.

File Description Specifications

Line	Form Type	Filename	File Type		Mode of Processing		Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered	
			File Designation	End of File	Length of Key Field or of Record Address Field	Record Address Type							Number of Tracks for Cylinder Overflow	Number of Extents
		File Format		Type of File Organization or Additional Area		Continuation Lines		Option		Entry		File Condition U1-UB, UC		
		Block Length		Overflow Indicator		K		A/D		R/U/N		Tape Rewind		
		Record Length		Key Field Starting Location		Option		Entry		File Condition U1-UB, UC		Tape Rewind		
		External Record Name		Extension Code E/L		Option		Entry		File Condition U1-UB, UC		Tape Rewind		
						Option		Entry		File Condition U1-UB, UC		Tape Rewind		
0 2	F		IP	E			WORKSTN							
0 3	F		IS	E			WORKSTN							
0 4	F													
0 5	F		CP	F			WORKSTN							
0 6	F		CS	F			WORKSTN							
0 7	F													
0 8	F		IF	E			WORKSTN							
0 9	F													
1 0	F		CF	F			WORKSTN							
1 1	F													
1 2	F		O	E			WORKSTN							
1 3	F													
1 4	F		IP	F			WORKSTN							
1 5	F		IS	F			WORKSTN							
1 6	F													
1 7	F		CP	F			WORKSTN							
1 8	F		CS	F			WORKSTN							
1 9	F													
2 0	F		IF	F			WORKSTN							
2 1	F													
2 2	F		CF	F			WORKSTN							
2 3	F													
2 4	F		O	F			WORKSTN							
2 5	F													
2 6	F		IR	F		A	EWOR KSTN							
2 7	F													
2 8	F	Subfile Processing												
2 9	F													
3 0	F			E			WORKSTN							
3 1	F						RECNO	KSFILE	RECORD					
3 2	F						FIELD		FORMAT					
3 3	F								NAME					
3 4	F													

Figure 69. Processing Chart for WORKSTN Files

Valid File Operations:

1. CLOSE, FORCE
2. WRITE, CLOSE, FORCE
3. READ, OPEN, CLOSE
4. READ, WRITE, EXFMT, OPEN, CLOSE
5. WRITE, OPEN, CLOSE
6. READ, WRITE, OPEN, CLOSE
7. OPEN, CLOSE
8. READC, CHAIN, WRITE, UPDAT, (valid only for record defined as a subfile)

Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. WRITE operations to a program-described file require a data-structure name in the result field; WRITE operations to a program-described file that uses a format name on output specifications are not valid.
3. Subfile processing is valid only for an externally described file.

Subfiles

Subfiles can be specified in the DDS for a display-device file to allow you to handle multiple records of the same type on the display. (See Figure 70 on page 151.) A subfile is a group of records that is read from or written to a display-device file. For example, a program reads records from a database file and creates a subfile of output records. When the entire subfile has been written, the program sends the entire subfile to the display device in one write operation. The work-station user can change data or enter additional data in the subfile. The program then reads the entire subfile from the display device into the program and processes each record in the subfile individually.

Records that you want to be included in a subfile are specified in the DDS for the file. The number of records that can be included in a subfile must also be specified in the DDS. One file can contain more than one subfile, and up to 12 subfiles can be active concurrently. Two subfiles can be displayed at the same time.

The DDS for a subfile consists of two record formats: a subfile-record format and a subfile control-record format. The subfile-record format contains the field information that is transferred to or from the display file under control of the subfile control-record format. The subfile control-record format causes the physical read, write, or control operations of a subfile to take place. Figure 71 on page 152 shows an example of the DDS for a subfile-record format, and Figure 72 on page 153 shows an example of the DDS for a subfile control-record format.

For a description of how to use subfile keywords, see the *DDS Reference*.


```

Customer Name Search
Search Code _____

Number  Name                Address                City                State
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
XXXX   XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XX
    
```

Figure 70. Subfile Display

To use a subfile for a display device file in an RPG/400 program, you must specify the SFILF keyword in positions 54 through 59 on a file description specifications continuation line for the WORKSTN file. The SFILF keyword must be specified on a separate continuation line. The WORKSTN file must be an externally described file (E in position 19).

You use positions 60 through 67 of the continuation line to specify the name of the subfile record format (not the control-record format). Positions 47 through 52 must specify the name of the field that contains the relative record number to be used in processing the subfile.

In an RPG/400 program, relative record number processing is defined as part of the SFILF definition. The SFILF definition implies a full-procedural update file with ADD for the subfile. Therefore, the file operations that are valid for the subfile are not dependent on the definition of the main WORKSTN file. That is, the WORKSTN file can be defined as a primary file or a full-procedural file.

Use the CHAIN, READC, UPDAT, or WRITE operation codes with the subfile record format to transfer data between the program and the subfile. Use the READ, WRITE, or EXFMT operation codes with the subfile control-record format to transfer data between the program and the display device or to process subfile control operations.

Processing WORKSTN Files

Subfile processing follows the rules for relative-record-number processing. The RPG/400 program places the relative-record number of any record retrieved by a READC operation into the field named in positions 47 through 52 of the file description specifications SFILE continuation line. This field is also used to specify the record number that the RPG/400 program uses for WRITE operation to the subfile or for output operations that use ADD. The field name specified in positions 47 through 52 must be defined as numeric with zero decimal positions. The field must have enough positions to contain the largest record number for the file. (See the SFLSIZ keyword in the *DDS Reference*.) The WRITE operation code and the ADD specification on the output specifications require that a relative-record-number field be specified in positions 47 through 52 of the file description specifications SFILE continuation line.

If a WORKSTN file has an associated subfile, all implicit input operations and explicit calculation operations that refer to the file name are processed against the main WORKSTN file. Any operations that specify a record format name that is not designated as a subfile are processed on the main WORKSTN file.

If you press a specified function key during a read of a non-subfile record, subsequent reads of a subfile record will cause the corresponding function key indicator to be set on again, even if the function key indicator has been set off between the reads. This will continue until a non-subfile record is read from the WORKSTN file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A** CUSTOMER NAME SEARCH
A
A          R SUBFIL          REF(DSTREF) 1
A          SFL 2
A          TEXT('Subfile Record')
A          CUST      R          7 3
A          NAME      R          7 10
A          ADDR      R          7 32 3
A          CITY      R          7 54
A          STATE     R          7 77
A*
```

Figure 71. Data Description Specifications for a Subfile Record Format

The data description specifications (DDS) for a subfile record format describe the records in the subfile:

- 1** The attributes for the fields in the record format are contained in the field reference file DSTREF as specified by the REF keyword.
- 2** The SFL keyword identifies the record format as a subfile.
- 3** The line and position entries define the location of the fields on the display.

Use of Subfiles

Some typical ways you can make use of subfiles include:

- Display only. The work-station user reviews the display.
- Display with selection. The user requests more information about one of the items on the display.
- Modification. The user changes one or more of the records.

- Input only, with no validity checking. A subfile is used for a data entry function.
- Input only, with validity checking. A subfile is used for a data entry function, but the records are checked.
- Combination of tasks. A subfile can be used as a display with modification, plus the input of new records.

The following figure shows an example of data description specifications for a subfile control-record format. For an example of using a subfile in an RPG/400 program, see “WORKSTN File Examples” on page 158.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R  FILCTL                               SFLCTL(SUBFIL)
A N70                                             SFLCLR
A 70                                             SFLDSPCTL
A 71                                             SFLDSP
A                                             SFLSIZ(15)
A                                             SFLPAG(15)
A                                             TEXT('Subfile Control Record')
A                                             OVERLAY
A 71                                             ROLLUP(97 'Continue Search')
A                                             CA01(98 'End of Program')
A                                             HELP(99 'Help Key')
A                                             1 2'Customer Name Search'
A                                             3 2'Search Code'
A          SRHCOD  R          I 3 14PUTRETAIN
A                                             5 2'Number'
A                                             5 10'Name'
A                                             5 32'Address'
A                                             5 54'City'
A                                             5 76'State'
A*

```

Figure 72. Data Description Specifications for a Subfile Control-Record Format

The subfile control-record format defines the attributes of the subfile, the search input field, constants, and function keys. The keywords you can use indicate the following:

- SFLCTL names the associated subfile (SUBFIL).
- SFLCLR indicates when the subfile should be cleared (when indicator 70 is off).
- SFLDSPCTL indicates when to display the subfile control record (when indicator 70 is on).
- SFLDSP indicates when to display the subfile (when indicator 71 is on).
- SFLSIZ indicates the total number of records to be included in the subfile (15).
- SFLPAG indicates the total number of records in a page (15).
- ROLLUP indicates that indicator 97 is set on in the program when the user presses the Roll Up key.

Program-Described WORKSTN File

- HELP allows the user to press the Help key for a displayed message that describes the valid function keys.
- PUTRETAIN allows the value that is entered in the SRHCOD field to be kept in the display.

In addition to the control information, the subfile control-record format also defines the constants to be used as column headings for the subfile record format.

Program-Described WORKSTN File

You can use a program-described WORKSTN file with or without a format name specified on the output specifications. The format name, if specified, refers to the name of a data description specifications record format. This record format describes:

- How the data stream sent from an RPG/400 program is formatted on the screen
- What data is sent
- What ICF functions to perform.

If a format name is used, input and output specifications must be used to describe the input and output records.

You can specify the PASS option on the file description specifications continuation line for a program-described WORKSTN file. Positions 60 through 65 must contain *NOIND. The PASS *NOIND indicates that the RPG/400 program will not additionally pass indicators to data management on output or receive them on input. It is your responsibility to pass indicators by describing them as fields (in the form *INxx, *IN, or *IN,x) in the input or output record. They must be specified in the sequence required by the data description specifications (DDS). You can use the DDS listing to determine this sequence.

Program-Described WORKSTN File with a Format Name

The following specifications apply to using a format name for a program-described WORKSTN file.

Output Specifications

On the output specifications, you must specify the WORKSTN file name in positions 7 through 14. The format name, which is the name of the DDS record format, is specified as a literal or named constant in positions 45 through 54 on the succeeding field description line. K1 through K8 must be specified (right-adjusted) in positions 40 through 43 on the line containing the format name. The K identifies the entry as a length rather than an end position, and the number indicates the length of the format name. For example, if the format name is CUSPMT, the entry in positions 40 through 43 is K6. (Leading zeros following the K are allowed.) The format name cannot be conditioned (indicators in positions 23 through 31 are not valid).

Output fields must be located in the output record in the same order as defined in the DDS; however, the field names do not have to be the same. The end position entries for the fields refer to the end position in the output record passed from the RPG/400 program to data management, and not to the location of the fields on the screen.

To pass indicators on output, do one of the following:

- Specify the keyword `INDARA` in the DDS for the `WORKSTN` file. Do not use the `PASS *NOIND` option on the file specifications continuation line and do not specify the indicators on the output specifications. The program and file use a separate indicator area to pass the indicators.
- Specify the `PASS *NOIND` option on the file specifications continuation line. Specify the indicators in the output specifications as fields in the form `*INxx`. The indicator fields must precede other fields in the output record, and they must appear in the order specified by the `WORKSTN` file DDS. You can determine this order from the DDS listing.

Input Specifications

The input specifications describe the record that the `RPG/400` program receives from the display or `ICF` device. The `WORKSTN` file name must be specified in positions 7 through 14. Input fields must be located in the input record in the same sequence as defined in the DDS; however, the field names do not have to be the same. The field location entries refer to the location of the fields in the input record.

To receive indicators on input, do one of the following:

- Specify the keyword `INDARA` in the DDS for the `WORKSTN` file. Do not use the `PASS *NOIND` option on the file specifications continuation line and do not specify the indicators on the input specifications. The program and file use a separate indicator area to pass the indicators.
- Specify the `PASS *NOIND` option on the file specifications continuation line. Specify the indicators in the input specifications as fields in the form `*INxx`. They must appear in the input record in the order specified by the `WORKSTN` file DDS. You can determine this order from the DDS listing.

A record identifying indicator should be assigned to each record in the file to identify the record that has been read from the `WORKSTN` file. A hidden field with a default value can be specified in the DDS for the record identification code.

Calculation Specifications

The operation code `READ` is valid for a program-described `WORKSTN` file that is defined as a combined, full-procedural file. See Figure 68 on page 148. The file name must be specified in factor 2 for this operation. A format must exist at the device before any input operations can take place. This requirement can be satisfied on a display device by conditioning an output record with `1P` or by writing the first format to the device in another program (for example, in the `CL` program). The `EXFMT` operation is not valid for a program-described `WORKSTN` file. You can also use the `EXCPT` operation to write to a `WORKSTN` file.

Additional Considerations

When using a format name with a program-described WORKSTN file, you must also consider the following:

- The name specified in positions 45 through 54 of the output specifications is assumed to be the name of a record format in the DDS that was used to create the file.
- If a Kn specification is present for an output record, it must also be used for any other output records for that file. If a Kn specification is not used for all output records to a file, a run-time error will occur.

For an example of using a format name with a program-described display device WORKSTN file, see “Sample Program 6—Program-Described WORKSTN File with a FORMAT Name on Output Specifications” on page 206.

Program-Described WORKSTN File without a Format Name

When a record-format name is not used, a program-described display-device file describes a file containing one record-format description with one field. The fields in the record must be described within the program that uses the file.

When you create the display file by using the Create Display File command, the file has the following attributes:

- A variable record length can be specified; therefore, the actual record length must be specified in the using program. (The maximum record length allowed is the screen size minus one.)
- No indicators are passed to or from the program.
- No function key indicators are defined.
- The record is written to the display beginning in position 2 of the first available line.

Input File

For an input file, the input record, which is treated by the OS/400 device support as a single input field, is initialized to blanks when the file is opened. The cursor is positioned at the beginning of the field, which is position 2 on the display.

Output File

For an output file, the OS/400 device support treats the output record as a string of characters to be sent to the display. Each output record is written as the next sequential record in the file; that is, each record displayed overlays the previous record displayed.

Combined File

For a combined file, the record, which is treated by the OS/400 device support as a single field, appears on the screen and is both the output record and the input record. Device support initializes the input record to blanks, and the cursor is placed in position 2.

For more information on program-described-display-device files, see the *Data Management Guide*.

Multiple-Device Files

Any RPG/400 WORKSTN file with at least one of the keywords ID, IND, NUM, or SAVDS specified (on the file specifications continuation line) is a multiple-device file. Through a multiple-device file, your program may access more than one device.

The RPG/400 program accesses devices through program devices, which are symbolic mechanisms for directing operations to an actual device. When you create a file (using the DDS and commands such as the create file commands), you consider such things as which device is associated with a program device, whether or not a file has a requesting program device, which record formats will be used to invite devices to respond to a READ-by-file-name operation, and how long this READ operation will wait for a response. For detailed information on the options and requirements for creating a multiple-device file, see the chapter on display files in the *Data Management Guide*, and information on ICF files in the *ICF Programmer's Guide*, and the manuals you are referred to in these two publications.

With multiple-device files, you make particular use of the following operation codes:

- In addition to opening a file, the OPEN operation can acquire (at most) one device for a multiple-device file. You specify which device when you create the file.
- The ACQ (acquire) operation acquires any other devices for your file.
- The REL (release) operation releases a device from the file.
- The WRITE operation, when used with the DDS keyword INVITE, invites a program device to respond to subsequent read-from-invited-program-devices operations. See the section on inviting a program device in the *ICF Programmer's Guide* and the *Data Management Guide*.
- The READ operation either processes a read-from-invited-program-devices operation or a read-from-one-program-device operation. When no NEXT operation is in effect, a program-cycle-read or READ-by-file-name operation waits for input from any of the devices that have been invited to respond (read-from-invited-program-device). Other input and output operations, including a READ-by-file-name after a NEXT operation, and a READ-by-format-name, process a read-from-one-program-device operation using the program device indicated in a special field. (The field is named in the ID entry of the file specifications continuation lines.)

This device may be the device used on the last input operation, a device you specify, or the requesting program device. See the sections on reading from invited program devices and on reading from one program device in the *ICF Programmer's Guide* and the *Data Management Guide*.

- The NEXT operation specifies which device is to be used in the next READ-by-file-name operation or program-cycle-read operation.
- The POST operation puts information in the INFDS information data structure. The information may be about a specific device or about the file. (The POST operation is not restricted to use with multiple-device files.)

See the *RPG/400 Reference* for details of the RPG/400 operation codes.

On the file specifications continuation line, you can specify several options to control the processing of multiple-device files.

WORKSTN File Examples

- The NUM entry indicates the maximum number of devices that can be acquired for a file.

By using a value of 1 for NUM, it is possible to get functions associated with a multiple-device file for a file that has only one device. For example, Figure 107 on page 211 illustrates the use of the time-out feature of the READ operation for a multiple-device file.

- The ID entry specifies the name of a field. The field can contain the name of a program device to which some input and output operations are directed.

When a read-from-one-program-device or WRITE operation is issued, the device used for the operation is the device identified by the field specified in the ID entry. This field is initialized to blanks and is updated with the name of the device from which the last successful input operation occurred. It can also be set explicitly by moving a value to it. The ACQ operation code does not affect the value of this field. If there is no entry, the operation is performed against the device from which the last successful input operation occurred. A blank device name is used if a read operation has not yet been performed successfully from a device.

When a read-from-one-program device or WRITE operation is issued with a blank device name, the RPG/400 compiler implicitly uses the device name of the requestor device for the program. If you call an RPG/400 program interactively and acquire an ICF device against which you want to perform one of these operations, you must explicitly move the device name of the ICF device into the field name specified with the ID entry prior to performing the operation. If this is not done, the device name used will either be blank (in which case the interactive requestor device name is used), or the device name used is the one from the last successful input operation. Once you have performed an I/O operation to the ICF device, you do not need to modify the value again unless an input operation completes successfully with a different device.

- The SAVDS entry indicates a data structure that is saved and restored for each device acquired to a file. The IND entry indicates a set of indicators to be saved and restored for each device acquired to a file. Before an input operation, the current set of indicators and data structure are saved. After the input operation, the RPG/400 compiler restores the indicators and data structure for the device associated with the operation. This may be a different set of indicators or data structure than was available before the input operation.
- The INFDS entry specifies the file information data structure for the WORKSTN file. The RPG/400 *STATUS field and the major/minor return code for the I/O operation can be accessed through this data structure. Particularly when ICF is being used, both fields are useful for detecting errors that occurred during I/O operations to multiple-device files.

Note: When specifying these control options, you must code the NUM option before the ID, IND or SAVDS options.

WORKSTN File Examples

This section illustrates some common work station applications and their RPG/400 coding.

- “Sample Program 1–Inquiry” on page 159 is an example of a basic inquiry program that uses the WORKSTN file in the RPG/400 compiler.

- “Sample Program 2–Data Entry with Master Update” on page 166 is an example of a data entry with master update program.
- “Sample Program 3–Maintenance” on page 174 is an example of a maintenance program.
- “Sample Program 4–WORKSTN Subfile Processing” on page 187 is an example of WORKSTN subfile processing.
- “Sample Program 5–Inquiry by Zip Code and Search on Name” on page 196 is an example of an interactive program in which the search of a name field occurs when the workstation user enters a zip code and a search string in response to the first display written by the program. This sample program illustrates one approach to solving the typical problem of identifying a customer and determining the correct customer number. In this example, the user knows the zip code and something about the customer name, such as some of the characters that constitute the name.
- “Sample Program 6–Program-Described WORKSTN File with a FORMAT Name on Output Specifications” on page 206 is an example of a program-described WORKSTN file with a format name on the output specifications.
- “Sample Program 7–Variable Start Line” on page 208 is an example of using the variable start line to determine where a record format will appear on a display.
- “Sample Program 8–Read Operation with Time-Out” on page 211 shows how to use READ with a time-out.

Sample Program 1–Inquiry

The following figures illustrate a simple inquiry program using the WORKSTN file:

<i>Table 5. List of Figures for WORKSTN Inquiry Program</i>	
Figure	Contents
Figure 73 on page 160 below and Figure 74 on page 161	DDS for database file and display device file
Figure 75 on page 164	File description and calculation specifications
Figure 76 on page 165	Prompt screen
Figure 77 on page 166	Customer information screen

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A* CUSTOMER MASTER FILE -- CUSMSTP
A          R CUSREC
A          CUST          5          TEXT('CUSTOMER NUMBER')
A          NAME          20         TEXT('CUSTOMER NAME')
A          ADDR          20         TEXT('CUSTOMER ADDRESS')
A          CITY          20         TEXT('CUSTOMER CITY')
A          STATE         2          TEXT('CUSTOMER STATE')
A          ZIP           5  0        TEXT('CUSTOMER ZIP CODE')
A          SRHCOD         3          TEXT('CUSTOMER NAME SEARCH CODE')
A          CUSTYP         1          TEXT('CUSTOMER TYPE')
A          ARBAL         10  2        TEXT('ACCOUNTS RECEIVABLE BALANCE')
A*****
A* FILE NAME : CUSMSTL *
A* DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP) *
A*          BY CUSTOMER NUMBER (CUST) *
A*****
A.....T.Name+++++.Len++TDpB.....Functions+++++*****
A          R CUSREC          PFILE(CUSMSTP)
A          K CUST

```

Figure 73. DDS for WORKSTN Inquiry-Program File CUSMSTP

The DDS for the database file used by this program describe one record format: CUSREC. The logical file CUSMSTL keyed by customer number is based on the physical file CUSMSTP, as indicated by the PFILE keyword. Each field in the record format is defined in the physical file CUSMSTP.

Note: Normally, the field attributes, such as the number of decimal positions and the data type, are defined in a field-reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what they are.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : CUSFMT *
A* DESCRIPTION: DISPLAY FILE FOR CUSTOMER MASTER INQUIRY *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          DSPSIZ(24 80 *DS3)
A          CHGINPDFT(CS)
A          CA03(15 'END OF JOB')
A          PRINT
A          INDARA
A          R CUSHDG
A          OVERLAY
A          2 4TIME
A          DSPATR(HI)
A          2 29'Customer Master Inquiry'
A          DSPATR(HI)
A          DSPATR(UL)
A          2 70DATE
A          EDTCDE(Y)
A          DSPATR(HI)
A          R CUSFTG
A          23 20'ENTER - Continue'
A          DSPATR(HI)
A          23 49'F3 - End Job'
A          DSPATR(HI)
A          R CUSPMT
A          OVERLAY
A          CUST          5A I 10 50DSPATR(HI)
A          DSPATR(CS)
A 99          ERRMSG('Customer Number not Found'
A          99)
A          10 26'Enter Customer Number:'
A          DSPATR(HI)

```

Figure 74 (Part 1 of 2). DDS for WORKSTN Inquiry-Program Display Device File CUSFMT

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A      R CUSFLDS
A
A      OVERLAY
A      8 25'Name'
A      NAME      20A 0 8 35DSPATR(HI)
A      10 25'Address'
A      ADDR      20A 0 10 35DSPATR(HI)
A      12 25'City'
A      CITY      20A 0 12 35DSPATR(HI)
A      14 25'State'
A      STATE     2A 0 14 35DSPATR(HI)
A      14 41'Zip Code'
A      ZIP       5S 00 14 50DSPATR(HI)
A      16 25'A/R Balance'
A      ARBAL     10Y 20 16 42DSPATR(HI)
A      EDTCDE(J)
A      6 25'Customer'
A      CUST      5A 0 6 35DSPATR(HI)

```

Figure 74 (Part 2 of 2). DDS for WORKSTN Inquiry-Program Display Device File CUSFMT

The DDS for the display device file CUSFMT to be used by this program specify file level entries and describe four record formats: CUSHDG, CUSFTG, CUSPMT, and CUSFLDS.

The file level entries define the screen size (DSPSIZ), input defaults (CHGINPFT), command attention key used to end the program, print key (PRINT), and a separate indicator area (INDARA).

The CUSHDG record format contains the constant 'Customer Master Inquiry', which identifies the display. It also contains the keywords TIME and DATE, which will display the current date and time on the screen.

The CUSFTG record format contains the constants 'ENTER - Continue' and 'F3 - End Job', which describe the processing options.

The CUSPMT record format contains the prompt "Enter Customer Number:" and the input field CUST into which the workstation user enters the customer number. Column separators define the input field on the screen where the user is to enter the customer number. The error message "Customer Number not Found" is also included in this record format. This message is displayed if indicator 99 is set on by the program.

The CUSFLDS record format contains the constants 'Name', 'Address', 'City', 'State', 'Zip Code', 'A/R Balance', and 'Customer' that identify the fields to be written out from the program. This record format also describes fields that correspond to these constants. All of these fields are described as output fields because they are filled in by the program; the user does not enter any data into these fields. To enter another customer number, the user presses Enter in response to this record.

In addition to describing the constants, fields and attributes for the screen, the record formats also define the display attributes for the constants and fields and the line numbers and horizontal positions where the constants and fields are to be displayed.

Notice the use of the `OVERLAY` keyword; the `CUSHDG`, `CUSPMT` and `CUSFLDS` record formats will overlay the `CUSFTG` record format. The `CUSFTG` format will remain on the screen when any of the other formats are written to the screen.

Note: Normally, the field attributes are defined in a field-reference file instead of the DDS for a file. However, they are shown here so you can see the field attributes.

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*  PROGRAM ID   - CUSTINQ                                     *
F*  PROGRAM NAME - CUSTOMER MASTER INQUIRY                   *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FCUSMSTL IF E           K           DISK
FCUSFMT CF E           WORKSTN

CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C           *IN15       DOWEQ'0'
C*
C* WRITE HEADING AND FOOTING EXCEPT IF ERROR HAS OCCURRED
C* AND PROMPT FOR CUSTOMER NUMBER
C           *IN99       CASEQ'0'       HEADNG
C           END
C           EXFMTCUSPMT
C* IF NOT END OF JOB AND VALID CUSTOMER NUMBER
C* DISPLAY CUSTOMER INFORMATION
C           *IN15       IFEQ '0'
C           CUST       CHAINCUSREC           99
C           *IN99       IFEQ '0'
C           EXFMTCUSFLDS
C           END IF
C           END IF
C           END DO
C           MOVE '1'       *INLR
C*****
C*  SUBROUTINE - HEADNG                                     *
C*  PURPOSE    - DISPLAY HEADING AND FOOTING               *
C*****
C           HEADNG       BEGSR
C           WRITECUSFTG
C           WRITECUSHDG
C           ENDSR

```

Figure 75. File Description and Calculation Specifications for WORKSTN Inquiry Program

For this program, only the RPG/400 file description and calculation specifications are required. Input and output specifications are not required because both files are externally described files (as indicated by the E in position 19). Both files are described as full-procedural files, as indicated by the F in position 16, because the I/O operations are controlled by programmer-specified operation codes. In addition, the K in position 31 of the file description specifications for the CUSMSTL file indicates that the file is processed keys.

The DOWEQ operation performs a loop until the user presses F3 to end the job. F3 sets indicator 15 on, as defined in the DDS. If indicator 15 is on, the loop is ended, the LR indicator is turned on, and the program ends.

The CASEQ operation performs subroutine HEADNG, which writes the heading and footings to the screen. Headings and footings will not be written to the screen when an error has occurred.

The EXFMT operation writes the CUSPMT record to the display. This record prompts the user to enter a customer number. If the user enters a customer number and presses Enter, the same EXFMT operation then reads the record back into the program.

If the user does not end the job, the CHAIN operation retrieves a record from the CUSMSTL file. Note that the record format name CUSREC is specified for the CHAIN operation rather than the file name. If the record is not found, indicator 99 is set on and the program loops back to display the CUSPMT record again. The message Customer Number not Found is displayed, the ERRMSG keyword in the DDS is conditioned by indicator 99, and the keyboard is locked. The user must press the Reset key in response to this message to unlock the keyboard. The user can then enter another customer number.

If the CHAIN operation retrieves a record from the CUSMSTL file, the EXFMT operation writes the record CUSFLDS to the display work station. This record contains the customer's name, address information, and accounts receivable balance.

The user then presses Enter, and the program loops back to the EXFMT operation and writes record CUSPMT to the display work station. The user can enter another customer number or end the program.

Figure 76 is the initial display written to the display WORKSTN by the EXFMT.

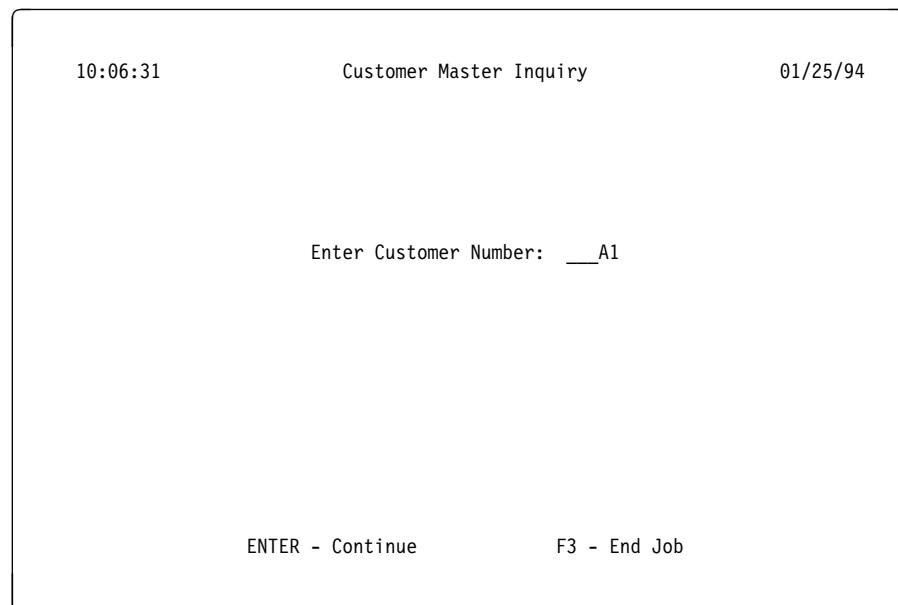


Figure 76. Customer Inquiry Prompt Screen

WORKSTN File Examples

The following display appears if a record is found in the CUSTMSTL file with the same customer number that was entered by the user in response to the first display:

10:06:31	Customer Master Inquiry	01/25/94
Customer	__A1	
Name	COLLINS	
Address	12 MILLDON ROAD	
City	OLYMPIA	
State	WA	Zip Code 50079
A/R Balance	11,111,111.00	
ENTER - Continue		F3 - End Job

Figure 77. Customer Inquiry Information Screen

Sample Program 2–Data Entry with Master Update

The following figures illustrate a data-entry program that prompts the user, updates a master record, and writes a transaction file:

<i>Table 6. List of Figures for WORKSTN Data Entry Program</i>	
Figure	Contents
Figure 78 on page 167 below and Figure 79 on page 168	DDS for master file, transaction file, and display device file
Figure 80 on page 170	File description and calculation specifications
Figure 81 on page 172	Prompt screen
Figure 82 on page 173	Display of current information
Figure 83 on page 173	Updated screen


```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A* PARTS MASTER FILE -- PARTMST
A      R MSTREC
A      PART#          5S 0      TEXT('PART NUMBER')
A      DESCRP        20         TEXT('DESCRIPTION')
A      ISSUE          7S 0      TEXT('QTY ISSUED')
A      RECPT          7S 0      TEXT('QTY RECEIVED')
A      ONHAND         9S 0      TEXT('QTY ON HAND')
A      DTLUPD         6S 0      TEXT('DATE LAST UPDATE')
A      K PART#
A*
A* PARTS TRANSACTION FILE -- TRNFIL
A      R TRNREC
A      PARTNO         5S 0      TEXT('PART NUMBER')
A      QTYISS         7S 0      TEXT('QTY ISSUED')
A      QTYREC         7S 0      TEXT('QTY RECEIVED')
A      DATE           6S 0      TEXT('CURRENT DATE')

```

Figure 78. DDS for Data-Entry/Update Master File and Transaction File

The DDS for the database files used by this program describe two record formats: MSTREC and TRNREC. The master file PARTMST is a keyed physical file; the transaction file TRNFIL is a sequential file.

Note: Normally, the field attributes, such as the number of decimal positions and the data type, are defined in a field-reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : PRTUPD *
A* DESCRIPTION: TRANSACTION AND MASTER FILE UPDATE *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          CHGINPDFT(CS)
A          PRINT(QSYSPRT)
A          INDARA
A          CA03(03 'END OF JOB')
A          R PROMPT
A          2 4TIME DSPATR(HI)
A          2 28'PART TRANSACTION ENTRY'
A          DSPATR(HI UL)
A          2 70DATE EDTCDE(Y) DSPATR(HI)
A          6 4'Enter Part Number'
A          DSPATR(HI)
A          PART# R Y I 6 23REFFLD(PART# PARTMST)
A          DSPATR(CS) CHECK(RB)
A 61 ERRMSG('PART # NOT FOUND' +
A          61)
A          23 6'ENTER - Continue'
A          DSPATR(HI)
A          23 29'F3 - End Job'
A          DSPATR(HI)
A          R TRNFMT
A          CA12(12 'CANCEL TRANS')
A          2 4TIME DSPATR(HI)
A          2 28'PART TRANSACTION ENTRY'
A          DSPATR(HI UL)
A          2 70DATE EDTCDE(Y) DSPATR(HI)
A          6 10'Part Number'
A          DSPATR(HI)

```

Figure 79 (Part 1 of 2). DDS for Data-Entry/Update PRTUPD Display Device File

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A      PART#      R      Y  0  6 23REFFLD(PART# PARTMST)
A      DESCRP     R           0  7 23REFFLD(DESCRP PARTMST)
A                                     9 10'Qty On Hand'
A      ONHAND     R      Y  0  9 23REFFLD(ONHAND PARTMST)
A                                     DSPATR(HI) EDTCDE(Z)
A                                     11 10'Qty Issued '
A      QTYISS     R      Y  B 11 25REFFLD(QTYISS TRNFIL)
A                                     CHECK(RB)
A                                     DSPATR(HI CS)
A                                     13 10'Qty Received'
A      QTYREC     R      Y  B 13 25REFFLD(QTYREC TRNFIL)
A                                     CHECK(RB) DSPATR(HI CS)
A                                     23  6'ENTER - Continue'
A                                     DSPATR(HI)
A                                     23 29'F3 - End Job'
A                                     DSPATR(HI)
A                                     23 46'F12 - Cancel Transaction'
A                                     DSPATR(HI)

```

Figure 79 (Part 2 of 2). DDS for Data-Entry/Update PRTUPD Display Device File

The DDS for the PRTUPD display device file contains two record formats: PROMPT and TRNFMT. The PROMPT record prompts for the part number to be processed. If the part is not found, an error message is displayed. The TRNFMT record is used to enter issue and receipt quantities. The fields are defined as output/input (B in position 38) and output (O in position 38).

F3 has been defined at the file level and is valid for all record formats. F12 is defined at the record level for the TRNFMT record format and is not valid for any other format.

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*  PROGRAM ID   - DTAENT                                     *
F*  PROGRAM NAME - TRANSACTION MAINTENANCE                   *
F*  THIS PROGRAM PERFORMS THE FOLLOWING FUNCTIONS:           *
F*    - ADDS NEW TRANSACTION RECORDS TO THE FILE TRNFIL     *
F*    - UPDATES PART MASTER FILE PARTMST                     *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPARTMST UF  E          K          DISK
FTRNFIL  0  E          K          DISK
FPRTUPD  CF  E          WORKSTN

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*  MAINLINE                                                 *
C*****
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++
C          EXFMTPROMPT
C*
C          *IN03      DOWEQ '0'
C*
C          PART#     CHAINMSTREC          61
C          *IN61     CASEQ '0'          NXTSCN
C          END
C*
C          *IN03     IFEQ '0'
C          EXFMTPROMPT
C          END
C*
C          END
C          MOVE '1'          *INLR

```

Figure 80 (Part 1 of 2). File Description Specification and Calculation Specification for Data Entry/Update Program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - NXTSCN                                     *
C*   PURPOSE    - ADD PART TRANSACTIONS                     *
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           NXTSCN      BEGSR
C           EXFMTTRNFMT
C           *IN03       IFEQ '0'
C           *IN12       ANDEQ'0'
C           ADD QTYISS   ISSUE
C           ADD QTYREC   RECPT
C           ADD QTYREC   ONHAND
C           SUB QTYISS   ONHAND
C           Z-ADDUDATE   DTLUPD
C           Z-ADDUDATE   DATE
C           UPDATMSTREC
C           WRITETRNFMT
C           Z-ADD*ZERO   QTYISS
C           Z-ADD*ZERO   QTYREC
C           ELSE
C           EXCPTRLS
C           END
C           ENDSR
OMSTREC  E           RLS

```

Figure 80 (Part 2 of 2). File Description Specification and Calculation Specification for Data Entry/Update Program

This program (data entry with master update) prompts the user for a transaction, updates a master record, and writes a transaction record.

The program has two disk files (PARTMST and TRNFIL) and one WORKSTN file (PRTUPD). The program begins by prompting the workstation user for a part number. The user can press F3, which is associated with indicator 03 in the DDS, to end the program.

The CHAIN operation retrieves the master record. If the record is not found, an error message is displayed; otherwise, the record format TRNFMT is displayed. The user can press F12 to cancel the transaction; the master record is released, and the PROMPT record format is displayed again. The user can press F3 to end the program, or the user can process the transaction. When the user presses ENTER after entering issue or receipt quantities, the master file PARTMST is updated with the current date, new on hand quantity, issues and receipts, and the transaction is added to the transaction file TRNFIL.

The workstation user responds to the prompts on the first screen by entering a part number as shown in Figure 81 on page 172.

WORKSTN File Examples

10:12:14	PART TRANSACTION ENTRY	01/25/94
Enter Part Number ____1		
ENTER - Continue F3 - End Job		

Figure 81. Prompt Screen for Data Entry/Update Program

Because part number 1 is in the Customer Master File, the program displays the following record for that part.

```

10:12:20          PART TRANSACTION ENTRY          01/25/94

      Part Number 00001
                  ALPHABC

      Qty On Hand   50

      Qty Issued    20

      Qty Received  50

ENTER - Continue      F3 - End Job      F12 - Cancel Transaction
    
```

Figure 82. TRNFMT Screen

The workstation user can press Enter to continue or F12 to cancel the transaction.

```

10:12:38          PART TRANSACTION ENTRY          01/25/94

      Part Number 00001
                  ALPHABC

      Qty On Hand   80

      Qty Issued    0000000

      Qty Received  0000000

ENTER - Continue      F3 - End Job      F12 - Cancel Transaction
    
```

Figure 83. TRNFIL Screen

Sample Program 3–Maintenance

The following figures illustrate a simple inquiry program using the WORKSTN file:

Figure	Contents
Figure 84 below and Figure 85 on page 175	DDS for master file and display device file
Figure 86 on page 179	File description and calculation specifications
Figure 87 on page 184	Display mode prompt screen
Figure 88 on page 185	Add mode prompt screen
Figure 89 on page 186	Update mode prompt screen
Figure 90 on page 186	Delete mode prompt screen

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A* CUSTOMER MASTER FILE -- CUSTMSTR
A      R  CSTMST
A      CUST#          5S 0      TEXT('CUSTOMER NUMBER')
A      CSTNAM         20        TEXT('CUSTOMER NAME')
A      CSTAD1         20        TEXT('CUSTOMER ADDRESS')
A      CSTAD2         20        TEXT('CUSTOMER ADDRESS')
A      CSTCTY         20        TEXT('CUSTOMER CITY')
A      CSTSTE          2        TEXT('CUSTOMER STATE')
A      CSTZIP         5S 0      TEXT('CUSTOMER ZIP CODE')
A      K  CUST#
```

Figure 84. DDS for Maintenance Program Master File

The DDS for the database file used by this program describe one record format: CSTMST. Each field in the record format is described, and the CUST# field is identified as the key field for the record format.

Note: Normally, the field attributes, such as number of decimal positions and data type, are defined in a field-reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : CSTENT *
A* DESCRIPTION: DISPLAY FILE FOR CUSTOMER MASTER INQUIRY *
A* SELECT OPTION SCREEN *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++
A REF(CUSTOMSTR)
A CHGINPDFT(CS)
A PRINT(QSYSVRT)
A INDARA
A R HDRSCN
A TEXT('PROMPT FOR CUST NUMBER')
A CA03(03 'END OF INQUIRY')
A CA05(05 'ADD MODE')
A CA06(06 'UPDATE MODE')
A CA07(07 'DELETE MODE')
A CA08(08 'DISPLAY MODE')
A MODE 8A 0 1 4DSPATR(HI)
A 1 13'MODE'
A DSPATR(HI)
A 2 4TIME
A DSPATR(HI)
A 2 28'CUSTOMER FILE MAINTENANCE'
A DSPATR(HI RI)
A 2 70DATE
A EDTCDE(Y)
A DSPATR(HI)
A CUST# R Y I 10 25DSPATR(CS)
A CHECK(RZ)
A 51 ERRMSG('CUSTOMER ALREADY ON +
A FILE' 51)
A 52 ERRMSG('CUSTOMER NOT ON FILE' +
A 52)

```

Figure 85 (Part 1 of 4). DDS for Display Device File for Customer Master Inquiry

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          10 33'<--Enter Customer Number'
A          DSPATR(HI)
A          23  4'F3 End Job'
A          23 21'F5 Add'
A          23 34'F6 Update'
A          23 50'F7 Delete'
A          23 66'F8 Display'
A          R CSTINQ          TEXT('DISPLAY CUST INFO')
A          CA12(12 'PREVIOUS SCREEN')
A          MODE              8  0  1  4DSPATR(HI)
A          1 13'MODE'
A          DSPATR(HI)
A          2  4TIME
A          DSPATR(HI)
A          2 28'CUSTOMER FILE MAINTENANCE'
A          DSPATR(HI RI)
A          2 70DATE
A          EDTCDE(Y)
A          DSPATR(HI)

```

Figure 85 (Part 2 of 4). DDS for Display Device File for Customer Master Inquiry

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A                               4 14'Customer:'
A                               DSPATR(HI UL)
A          CUST#      R          0 4 25DSPATR(HI)
A          CSTNAM     R          B 6 25DSPATR(CS)
A 04                               DSPATR(PR)
A          CSTAD1     R          B 7 25DSPATR(CS)
A 04                               DSPATR(PR)
A          CSTAD2     R          B 8 25DSPATR(CS)
A 04                               DSPATR(PR)
A          CSTCTY     R          B 9 25DSPATR(CS)
A 04                               DSPATR(PR)
A          CSTSTE     R          B 10 25DSPATR(CS)
A 04                               DSPATR(PR)
A          CSTZIP     R          B 10 40DSPATR(CS)
A                               EDTCDE(Z)
A 04                               DSPATR(PR)
A                               23 2'F12 Cancel'
A          MODE1      8 0 23 20
A          R CSTBLD   TEXT('ADD CUST RECORD')
A                               CA12(12 'PREVIOUS SCREEN')
A          MODE      8 0 1 4DSPATR(HI)
A                               1 13'MODE'      DSPATR(HI)
A                               2 4TIME
A                               DSPATR(HI)
A                               2 28'CUSTOMER FILE MAINTENANCE'
A                               DSPATR(HI RI)
A                               2 70DATE
A                               EDTCDE(Y)
A                               DSPATR(HI)
A          4 14'Customer:' DSPATR(HI UL)

```

Figure 85 (Part 3 of 4). DDS for Display Device File for Customer Master Inquiry

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          CUST#      R          0  4 25DSPATR(HI)
A          6 20'Name'    DSPATR(HI)
A          CSTNAM     R          I  6 25
A          7 17'Address' DSPATR(HI)
A          CSTAD1     R          I  7 25
A          8 17'Address' DSPATR(HI)
A          CSTAD2     R          I  8 25
A          9 20'City'    DSPATR(HI)
A          CSTCTY     R          I  9 25
A          10 19'State'  DSPATR(HI)
A          CSTSTE     R          I 10 25
A          10 36'Zip'    DSPATR(HI)
A          CSTZIP     R          Y  I 10 40
A          23  2'F12 Cancel Addition'

```

Figure 85 (Part 4 of 4). DDS for Display Device File for Customer Master Inquiry

The DDS for the CSTENT display device file contains three record formats: HDRSCN, CSTINQ, and CSTBLD. The HDRSCN record prompts for the customer number and the mode of processing. The CSTINQ record is used for the Update, Delete, and Display modes. The fields are defined as output/input (B in position 38). The fields are protected when Display or Delete mode is selected (DSPATR(PR)). The CSTBLD record provides only input fields (I in position 38) for a new record.

The CUSHDG record format contains the constant 'Customer Master Inquiry'; the ERRMSG keyword defines the messages to be displayed if an error occurs. The CA keywords define the function keys that can be used and associate the function keys with indicators in the RPG program.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*  PROGRAM ID   - CUSTMNT                                     *
F*  PROGRAM NAME - CUSTOMER MASTER MAINTENANCE               *
F*  THIS PROGRAM ADDS, UPDATES, DELETES AND DISPLAYS         *
F*  CUSTOMER RECORDS IN THE CUSTOMER MASTER FILE.           *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FCUSTMSTRUF  E          K          DISK          A
FCSTENT  CF  E          WORKSTN

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          CSTKEY    KLIST
C          KFLD      CUST#
C*****
C*      MAINLINE                                           *
C*****
C          MOVE 'DISPLAY 'MODE
C          EXFMTHDRSCN
C*
C          *IN03    DOWEQ'0'
C          EXSR  SETMOD
C*
C          CUST#    IFNE *ZERO
C          MODE     CASEQ'ADD'    ADDSUB
C          MODE     CASEQ'UPDATE'  UPDSUB
C          MODE     CASEQ'DELETE'  DELSUB
C          MODE     CASEQ'DISPLAY' INQSUB
C          END
C          END
C*
C          EXFMTHDRSCN
C          END
C          MOVE '1'    *INLR

```

Figure 86 (Part 1 of 5). File Description and Calculation Specifications for Maintenance Program

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - ADDSUB                                     *
C*   PURPOSE   - ADD NEW CUSTOMER TO FILE                   *
C*****
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C       ADDSUB      BEGSR
C       CSTKEY      CHAINCSTMST                          50
C       *IN50       IFEQ '0'
C                   MOVE '1'          *IN51
C                   ELSE
C                   MOVE '0'          *IN51
C                   MOVE *BLANK      CSTNAM
C                   MOVE *BLANK      CSTAD1
C                   MOVE *BLANK      CSTAD2
C                   MOVE *BLANK      CSTCTY
C                   MOVE *BLANK      CSTSTE
C                   Z-ADD*ZERO       CSTZIP
C                   EXFMTCSTBLD
C       *IN12       IFEQ '0'
C                   WRITECSTMST
C                   END
C                   END
C                   ENDSR

```

Figure 86 (Part 2 of 5). File Description and Calculation Specifications for Maintenance Program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - UPDSUB                                     *
C*   PURPOSE    - UPDATE CUSTOMER MASTER RECORD           *
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           UPDSUB      BEGSR
C           MOVE '0'          *IN04
C           CSTKEY      CHAINCSTMST                52
C           *IN52       IFEQ '0'
C                       EXFMTCSTINQ
C           *IN12       IFEQ '0'
C                       UPDATCSTMST
C                       ELSE
C                       EXCPTRLS
C                       END
C                       END
C                       ENDSR
C*****
C*   SUBROUTINE - DELSUB                                     *
C*   PURPOSE    - DELETE CUSTOMER MASTER RECORD           *
C*****
C           DELSUB      BEGSR
C           MOVE '1'          *IN04
C           CSTKEY      CHAINCSTMST                52
C           *IN52       IFEQ '0'
C                       EXFMTCSTINQ
C           *IN12       IFEQ '0'
C                       DELETCSTMST
C                       ELSE
C                       EXCPTRLS
C                       END
C                       END
C                       ENDSR

```

Figure 86 (Part 3 of 5). File Description and Calculation Specifications for Maintenance Program

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - INQSUB                                     *
C*   PURPOSE   - DISPLAY CUSTOMER MASTER RECORD           *
C*****
CL0N01N02N03Factor1+++OpCdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           INQSUB     BEGSR
C           MOVE '1'          *IN04
C           CSTKEY     CHAINCSTMST                52
C           *IN52      IFEQ '0'
C                   EXFMTCSTINQ
C                   EXCPTRLS
C                   END
C                   ENDSR
C*****
C*   SUBROUTINE - SETMOD                                     *
C*   PURPOSE   - SET MAINTENANCE MODE                     *
C*****
C           SETMOD     BEGSR
C           *IN05      IFEQ '1'
C                   MOVE 'ADD'      'MODE
C                   MOVE MODE      MODE1
C                   ELSE
C           *IN06      IFEQ '1'
C                   MOVE 'UPDATE'  'MODE
C                   MOVE MODE      MODE1
C                   ELSE
C           *IN07      IFEQ '1'
C                   MOVE 'DELETE'  'MODE
C                   MOVE MODE      MODE1
C                   ELSE

```

Figure 86 (Part 4 of 5). File Description and Calculation Specifications for Maintenance Program


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C*
C          *IN08          IFEQ '1'
C                      MOVE 'DISPLAY 'MODE
C                      MOVE MODE          MODE1
C                      ELSE
C                      END
C                      END
C                      END
C                      END
C                      ENDSR
OCSTMST  E                      RLS

```

Figure 86 (Part 5 of 5). File Description and Calculation Specifications for Maintenance Program

This program maintains a customer master file for additions, changes, and deletions. The program can also be used for inquiry.

The program first sets the default (display) mode of processing and displays the customer maintenance prompt screen. The workstation user can press F3, which turns on indicator 03, to request end of job. Otherwise, to work with customer information, the user enters a customer number and presses Enter. The user can change the mode of processing by pressing F5 (ADD), F6 (UPDATE), F7 (DELETE), or F8 (DISPLAY).

To add a new record to the file, the program uses the customer number as the search argument to chain to the master file. If the record does not exist in the file, the program displays the CSTBLD screen to allow the user to enter a new customer record. If the record is already in the file, an error message is displayed. The user can press F12, which sets on indicator 12, to cancel the add operation and release the record. Otherwise, to proceed with the add operation, the user enters information for the new customer record in the input fields and writes the new record to the master file.

To update, delete, or display an existing record, the program uses the customer number as the search argument to chain to the master file. If a record for that customer exists in the file, the program displays the customer file inquiry screen CSTINQ. If the record is not in the file, an error message is displayed. If the mode of processing is display or delete, the input fields are protected from modification. Otherwise, to proceed with the customer record, the user can enter new information in the customer record input fields. The user can press F12, which sets on indicator 12, to cancel the update or delete operation, and release the record. Display mode automatically releases the record when Enter is pressed.

WORKSTN File Examples

In the following screen, the workstation user responds to the prompt by entering customer number 00001 to display the customer record.

```
DISPLAY  MODE
10:09:01                CUSTOMER FILE MAINTENANCE                01/25/94

                                00001  <--Enter Customer Number

F3 End Job      F5 Add      F6 Update      F7 Delete      F8 Display
```

Figure 87 (Part 1 of 2). Display Mode Screens for Maintenance Program

Because the customer record for customer number 00001 exists in the Master File, the data is displayed as follows:

```
DISPLAY  MODE
10:09:11                CUSTOMER FILE MAINTENANCE                01/25/94

Customer: 00001

SMITH, JOE
SUITE 20000
QUEEN STREET
PORTLAND
OR                99999

F12 Cancel DISPLAY
```

Figure 87 (Part 2 of 2). Display Mode Screens for Maintenance Program

The workstation user responds to the add prompt by entering a new customer number as shown in the following screen.

```

ADD      MODE
10:09:20      CUSTOMER FILE MAINTENANCE      01/25/94

          00009  <--Enter Customer Number

F3 End Job   F5 Add   F6 Update   F7 Delete   F8 Display
    
```

Figure 88 (Part 1 of 2). Add Mode Screens for Maintenance Program

In the screen below, a new customer is added to the Customer Master File.

```

ADD      MODE
10:09:36      CUSTOMER FILE MAINTENANCE      01/25/94

Customer: 00009

      Name LANE, ROBERT
      Address Bellavista
      Address 17 Donleavy
      City Ontario
      State CA      Zip 15679

F12 Cancel Addition
    
```

Figure 88 (Part 2 of 2). Add Mode Screens for Maintenance Program

WORKSTN File Examples

The workstation user responds to the update prompt by entering a customer number as shown in the following screen.

```
UPDATE  MODE                                01/25/94
10:10:43                                CUSTOMER FILE MAINTENANCE

                                00006  <--Enter Customer Number

F3 End Job      F5 Add      F6 Update      F7 Delete      F8 Display
```

Figure 89. Update Mode Screen for Maintenance Program

The workstation user responds to the delete prompt by entering a new customer number in the following screen.

```
DELETE  MODE                                01/25/94
10:10:52                                CUSTOMER FILE MAINTENANCE

                                00009  <--Enter Customer Number

F3 End Job      F5 Add      F6 Update      F7 Delete      F8 Display
```

Figure 90. Delete Mode Screen for Maintenance Program

Sample Program 4–WORKSTN Subfile Processing

The following figures illustrate a WORKSTN file:

Figure	Contents
Figure 91 below and Figure 92 on page 188	DDS for master file and display device file
Figure 93 on page 191	File description and calculation specifications
Figure 94 on page 194	Prompt screen
Figure 95 on page 195	Display screen

```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
A.....T.Name+++++.Len++TDpB.....Functions+++++
A* CUSTOMER MASTER FILE -- CUSMSTP
A          R CUSREC
A          CUST          5          TEXT('CUSTOMER NUMBER')
A          NAME          20         TEXT('CUSTOMER NAME')
A          ADDR          20         TEXT('CUSTOMER ADDRESS')
A          CITY          20         TEXT('CUSTOMER CITY')
A          STATE         2          TEXT('CUSTOMER STATE')
A          ZIP           5 0        TEXT('CUSTOMER ZIP CODE')
A          SRHCOD        3          TEXT('CUSTOMER NAME SEARCH CODE')
A          CUSTYP        1          TEXT('CUSTOMER TYPE')
A          ARBAL         10 2       TEXT('ACCOUNTS RECEIVABLE BALANCE')
A*****
A* FILE NAME : CUSZIPL *
A* DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP) *
A*          BY CUSTOMER ZIP CODE (ZIP) *
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A          R CUSREC          PFILE(CUSMSTP)
A          ZIP          R
A          NAME          R
A          ARBAL          R
A          K ZIP

```

Figure 91. DDS for WORKSTN Subfile-Processing Program Master File

The DDS for the database file used by this program describe one record format: CUSREC. The logical file CUSZIPL keyed by zip code is based on the physical file CUSMSTP, as indicated by the PFILE keyword. The record format created by the logical file will include only those fields specified in the logical file DDS. All other fields will be excluded.

Note: Normally, the field attributes, such as number of decimal positions and data type, are defined in a field-reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : CUSSRC *
A* DESCRIPTION: DISPLAY CUSTOMER MASTER BY ZIP CODE *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++
A REF(CUSMSTP)
A CHGINPDFT(CS)
A PRINT(QSYSPRT)
A INDARA
A CA03(03 'END OF JOB')
A R HEAD
A OVERLAY
A 2 4TIME
A DSPATR(HI)
A 2 28'CUSTOMER FILE SEARCH'
A DSPATR(HI RI)
A 2 70DATE
A EDTCDE(Y)
A DSPATR(HI)
A R FOOT1
A 23 6'ENTER - Continue'
A DSPATR(HI)
A 23 29'F3 - End Job'
A DSPATR(HI)
A R FOOT2
A 23 6'ENTER - Continue'
A DSPATR(HI)
A 23 29'F3 - End Job'
A DSPATR(HI)
A 23 47'F4 - RESTART ZIP CODE'
A DSPATR(HI)

```

Figure 92 (Part 1 of 2). DDS for WORKSTN Subfile-Processing Program Display Device File

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R PROMPT
A
A          OVERLAY
A          4 4'Enter Zip Code'
A          DSPATR(HI)
A          ZIP          R          Y I 4 19DSPATR(CS)
A          CHECK(RZ)
A 61          ERRMSG('ZIP CODE NOT FOUND' +
A          61)
A          R SUBFILE          SFL
A          NAME          R          9 4
A          ARBAL          R          9 27EDTCDE(J)
A          R SUBCTL          SFLCTL(SUBFILE)
A 55          SFLCLR
A N55          SFLDSPCTL
A N55          SFLDSP
A          SFLSIZ(13)
A          SFLPAG(13)
A          ROLLUP(95 'ROLL UP')
A          OVERLAY
A          CA04(04 'RESTART ZIP CDE')
A          4 4'Zip Code'
A          ZIP          R          0 4 14DSPATR(HI)
A          7 4'Customer Name'
A          DSPATR(HI UL)
A          7 27'A/R Balance'
A          DSPATR(HI UL)

```

Figure 92 (Part 2 of 2). DDS for WORKSTN Subfile-Processing Program Display Device File

The DDS for the CUSSRC display device file contains six record formats: HEAD, FOOT1, FOOT2, PROMPT, SUBFILE, and SUBCTL.

The PROMPT record format requests the user to enter a zip code. If the zip code is not found in the file, an error message is displayed. The user can press F3, which sets on indicator 03, to end the program.

The SUBFILE record format must be defined immediately preceding the subfile-control record format SUBCTL. The subfile record format, which is defined with the keyword SFL, describes each field in the record, and specifies the location where the first record is to appear on the display (here, on line 9).

The subfile-control record format contains the following unique keywords:

- SFLCTL identifies this format as the control record format and names the associated subfile record format.
- SFLCLR describes when the subfile is to be cleared of existing records (when indicator 55 is on). This keyword is needed for additional displays.
- SFLDSPCTL indicates when to display the subfile-control record format (when indicator 55 is off).

WORKSTN File Examples

- SFLDSP indicates when to display the subfile (when indicator 55 is off).
- SFLSIZ specifies the total size of the subfile. In this example, the subfile size is 13 records that are displayed on lines 9 through 21.
- SFLPAG defines the number of records on a page. In this example, the page size is the same as the subfile size.
- ROLLUP indicates that indicator 95 is set on in the program when the roll up function is used.

The OVERLAY keyword defines this subfile-control record format as an overlay format. This record format can be written without the OS/400 system erasing the screen first. F4 is valid for repeating the search with the same zip code. (This use of F4 allows a form of roll down.)


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*  PROGRAM ID   - CUSTSFL                                     *
F*  PROGRAM NAME - CUSTOMER MASTER SEARCH                   *
F*  THIS PROGRAM DISPLAYS THE CUSTOMER MASTER FILE BY ZIP CODE *
F*****
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FCUSZIPL IF E           K           DISK
FCUSSRC  CF E           WORKSTN
F                               RECNUMKSFIL  SUBFILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           CSTKEY      KLIST
C           KFLD        ZIP
C*****
C*  MAINLINE                                               *
C*****
C           WRITEFOOT1
C           WRITEHEAD
C           EXFMTPROMPT
C*
C           *IN03      DOWEQ '0'
C           CSTKEY    SETLLCUSREC           20
C           *IN20     IFEQ *ZERO
C           MOVE '1'   *IN61
C           ELSE
C           EXSR SFLPRC
C           END
C           *IN03     IFEQ '0'
C           *IN04     IFEQ '0'
C           *IN61     IFEQ '0'
C           WRITEFOOT1
C           WRITEHEAD
C           END

```

Figure 93 (Part 1 of 3). File Description Specification and Calculation Specification for WORKSTN Subfile Processing Program

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C                EXFMTPROMPT
C                END
C                END
C                END
C*
C                SETON                                LR
C*****
C*  SUBROUTINE - SFLPRC                                *
C*  PURPOSE   - PROCESS SUBFILE AND DISPLAY           *
C*****
C      SFLPRC    BEGSR
C      NXPAG     TAG
C              EXSR SFLCLR
C              EXSR SFLFIL
C      SAMPAG    TAG
C              WRITEFOOT2
C              WRITEHEAD
C              EXFMTSUBCTL
C      *IN95     IFEQ '1'
C      *IN71     IFEQ '0'
C              GOTO NXPAG
C              ELSE
C              GOTO SAMPAG
C              END
C              END
C              ENDSR

```

Figure 93 (Part 2 of 3). File Description Specification and Calculation Specification for WORKSTN Subfile Processing Program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - SFLFIL                                     *
C*   PURPOSE   - FILL SUBFILE                               *
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           SFLFIL    BEGSR
C           *IN21     DOWEQ'0'
C           ZIP       READECUSREC                          71
C           *IN71     IFEQ '1'
C                   MOVE '1'          *IN21
C                   ELSE
C                   ADD 1             RECNUM
C                   WRITESUBFILE     21
C                   END
C                   END
C                   ENDSR
C*****
C*   SUBROUTINE - SFLCLR                                     *
C*   PURPOSE   - CLEAR SUBFILE RECORDS                     *
C*****
C           SFLCLR    BEGSR
C                   MOVE '1'          *IN55
C                   WRITESUBCTL
C                   MOVE '0'          *IN55
C                   MOVE '0'          *IN21
C                   Z-ADD*ZERO        RECNUM 50
C                   ENDSR

```

Figure 93 (Part 3 of 3). File Description Specification and Calculation Specification for WORKSTN Subfile Processing Program

The file description specifications identify the disk file to be searched and the display device file to be used (CUSSRC). The continuation line for the WORKSTN file identifies the record format (SUBFILE) that is to be used as a subfile. The relative-record-number field (RECNUM) specified in positions 47 through 52 of the continuation line controls which record within the subfile is being accessed.

The program displays the PROMPT record format and waits for the workstation user's response. F3 sets on indicator 03, which controls the end of the program. The zip code (ZIP) is used to position the CUSZIPL file by the SETLL operation. Notice that the record format name CUSREC is used in the SETLL operation instead of the file name CUSZIPL. If no record is found, an error message is displayed.

The SFLPRC subroutine handles the processing for the subfile: clearing, filling, and displaying. The subfile is prepared for additional requests in subroutine SFLCLR. If indicator 55 is on, no action occurs on the display, but the main storage area for the subfile records is cleared. The SFLFIL routine fills the subfile with records. A record is read from the CUSZIPL file. If the zip code is the same, the record count

WORKSTN File Examples

(RECNUM) is incremented and the record is written to the subfile. This subroutine is repeated until either the subfile is full (indicator 21 on the WRITE operation) or end of file occurs on the CUSZIPL file (indicator 71 on the READE operation). When the subfile is full or end of file occurs, the subfile is written to the display by the EXFMT operation by the subfile-control record control format. The user reviews the display and decides whether:

- To end the program by pressing F3.
- To restart the zip code by pressing F4. The PROMPT record format is not displayed, and the subfile is displayed starting over with the same zip code.
- To fill another page by pressing ROLL UP. If end of file has occurred on the CUSZIPL file, the current page is redisplayed; otherwise, the subfile is cleared and the next page is displayed.
- To continue with another zip code by pressing ENTER. The PROMPT record format is displayed. The user can enter a zip code or end the program.

In the screen below, the user enters a zip code in response to the prompt.

```
10:11:08          CUSTOMER FILE SEARCH          01/25/94
Enter Zip Code 72901

ENTER - Continue      F3 - End Job
```

Figure 94. Prompt Screen for WORKSTN Subfile-Processing Program

The subfile is written to the screen as shown:

10:11:23	CUSTOMER FILE SEARCH	01/25/94
Zip Code 72901		
Customer Name	A/R Balance	
BRADFIELD	11,111,111.00	
LEUNG	22,222,222.00	
ALLEN	33,333,333.00	
BELL	44,444,444.00	
KETCHUM	55,555,555.00	
FRASER	66,666,666.00	
GOODING	77,777,777.00	
LANE	88,888,888.00	
MARSHALL	11,111,111.00	
ROBERTS	11,111,222.00	
EWING	33,333,333.00	
LOGAN	44,444,444.00	
KENT	55,555,555.00	
ENTER - Continue	F3 - End Job	F4 - RESTART ZIP CODE

Figure 95. Display Screen for WORKSTN Subfile-Processing Program

Sample Program 5–Inquiry by Zip Code and Search on Name

The following figures illustrate a simple inquiry program using the WORKSTN file:

Figure	Contents
Figure 96 below and Figure 97 on page 197	DDS for master file and display device file
Figure 98 on page 200	File description and calculation specifications
Figure 99 on page 204	Prompt screen
Figure 100 on page 205	Information screen
Figure 101 on page 205	Detailed information screen

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++
A* CUSTOMER MASTER FILE -- CUSMSTP
A      R CUSREC
A      CUST          5      TEXT('CUSTOMER NUMBER')
A      NAME         20      TEXT('CUSTOMER NAME')
A      ADDR          20      TEXT('CUSTOMER ADDRESS')
A      CITY          20      TEXT('CUSTOMER CITY')
A      STATE         2       TEXT('CUSTOMER STATE')
A      ZIP           5 0     TEXT('CUSTOMER ZIP CODE')
A      SRHCOD        3       TEXT('CUSTOMER NAME SEARCH CODE')
A      CUSTYP        1       TEXT('CUSTOMER TYPE')
A      ARBAL        10 2     TEXT('ACCOUNTS RECEIVABLE BALANCE')
A*****
A* FILE NAME : MLGMSTL1 *
A* DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP) *
A*           BY ZIP CODE (ZIP) AND NAME(NAME) *
A*****
A.....T.Name+++++.Len++TDpB.....Functions+++++
A      R CUSREC          PFILE(CUSMSTP)
A      K ZIP
A      K NAME

```

Figure 96. DDS for Inquiry by Zip Code Master File

The DDS for the database file used in this program defines one record format named CUSREC, and identifies the ZIP and NAME fields as the key fields.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : MLG265D *
A* DESCRIPTION: DISPLAY CUSTOMER MASTER BY ZIP CODE & NAME *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          DSPSIZ(24 80 *DS3)
A          REF(CUSMSTP)
A          CHGINPDFT(CS)
A          PRINT(QSYSPRT)
A          INDARA
A          CA03(03 'END OF JOB')
A          R HEAD
A          OVERLAY
A          2 4TIME
A          DSPATR(HI)
A          2 29'Customer Master Inquiry'
A          DSPATR(HI UL)
A          2 70DATE
A          EDTCDE(Y)
A          DSPATR(HI)
A          R FOOT1
A          23 6'ENTER - Continue'
A          DSPATR(HI)
A          23 29'F3 - End Job'
A          DSPATR(HI)
A          R FOOT2
A          23 6'ENTER - Continue'
A          DSPATR(HI)
A          23 29'F3 - End Job'
A          DSPATR(HI)
A          23 47'F4 - Restart Zip Code'
A          DSPATR(HI)
A          R PROMPT
A          OVERLAY
A          4 4'Enter Zip Code'
A          DSPATR(HI)
A          ZIPCD      R      Y  I 4 19REFFLD(ZIP CUSMSTP)
A          CHECK(RZ) DSPATR(CS)
A          5 7'Search Name'
A          DSPATR(HI)
A          SRCNAM    R          I 5 19REFFLD(NAME CUSMSTP)
A          DSPATR(CS)

```

Figure 97 (Part 1 of 3). DDS for Inquiry by Zip Code Display Device File

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R SUBFILE
A          CHANGE(99 'FIELD CHANGED')
A          SFL
A          SEL          1  B  9  8DSPATR(CS)
A          VALUES(' ' 'X')
A          ZIP          R          0  9 17
A          CUST          R          0  9 30
A          NAME          R          0  9 43
A          R SUBCTL          SFLCTL(SUBFILE)
A          SFLSIZ(0013)
A          SFLPAG(0013)
A 55          SFLCLR
A N55          SFLDSPCTL
A N55          SFLDSP
A          ROLLUP(95 'ROLL UP')
A          OVERLAY
A          CA04(04 'RESTART ZIP CDE')
A          4  4'Zip Code'
A          ZIPCD          R          0  4 17REFFLD(ZIP CUSMSTP)
A          DSPATR(HI)
A          5  4'Search Name'
A          SRCNAM          R          0  5 17REFFLD(NAME CUSMSTP)
A          DSPATR(HI)
A          7  6'Select'
A          DSPATR(HI)
A          8  6' "X"          Zip Code          Number -
A          Customer Name          '
A          DSPATR(HI)
A          DSPATR(UL)

```

Figure 97 (Part 2 of 3). DDS for Inquiry by Zip Code Display Device File


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A      R CUSDSP
A
A      OVERLAY
A      CA04(04 'RESTART ZIP CDE')
A      6 25'Customer'
A      CUST      5A 0 6 35DSPATR(HI)
A      8 25'Name'
A      NAME      20A 0 8 35DSPATR(HI)
A      10 25'Address'
A      ADDR      20A 0 10 35DSPATR(HI)
A      12 25'City'
A      CITY      20A 0 12 35DSPATR(HI)
A      14 25'State'
A      STATE     2A 0 14 35DSPATR(HI)
A      14 41'Zip Code'
A      ZIP       5S 00 14 50DSPATR(HI)
A      16 25'A/R Balance'
A      ARBAL     10Y 20 16 42DSPATR(HI)
A      EDTCDE(J)

```

Figure 97 (Part 3 of 3). DDS for Inquiry by Zip Code Display Device File

The DDS for the CUSSRC display device file contains seven record formats: HEAD, FOOT1, FOOT2, PROMPT, SUBFILE, SUBCTL, and CUSDSP.

The PROMPT record format requests the user to enter a zip code and search name. If no entry is made, the display starts at the beginning of the file. The user can press F3, which sets on indicator 03, to end the program.

The SUBFILE record format must be defined immediately preceding the subfile-control record format SUBCTL. The subfile-record format defined with the keyword SFL, describes each field in the record, and specifies the location where the first record is to appear on the display (here, on line 9).

The subfile-control record format SUBCTL contains the following unique keywords:

- SFLCTL identifies this format as the control record format and names the associated subfile record format.
- SFLCLR describes when the subfile is to be cleared of existing records (when indicator 55 is on). This keyword is needed for additional displays.
- SFLDSPCTL indicates when to display the subfile-control record format (when indicator 55 is off).
- SFLDSP indicates when to display the subfile (when indicator 55 is off).
- SFLSIZ specifies the total size of the subfile. In this example, the subfile size is 15 records that are displayed on lines 9 through 23.
- SFLPAG defines the number of records on a page. In this example, the page size is the same as the subfile size.
- ROLLUP indicates that indicator 95 is set on in the program when the roll up function is used.

WORKSTN File Examples

The OVERLAY keyword defines this subfile-control record format as an overlay format. This record format can be written without the OS/400 system erasing the screen first. F3 is valid for repeating the search with the same zip code. (This use of F3 allows a form of roll down.)

The CUSDSP record format displays information for the selected customers.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*   PROGRAM ID   - MLG265                                     *
F*   PROGRAM NAME - MAILING LIST SEARC BY ZIP CODE/NAME      *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FMLGMSTL1IF  E           K           DISK
FMLG265D CF  E           WORKSTN
F
RECNUMKSFILE SUBFILE
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C           CSTKEY      KLIST
C           KFLD           ZIPCD
C           KFLD           SRCNAM
C           ZIPKEY      KLIST
C           KFLD           ZIP
C           KFLD           NAME
C*****
C*   MAINLINE                                               *
C*****
C           WRITEFOOT1
C           WRITEHEAD
C           EXFMTPROMPT
C           *IN03      DOWEQ'0'
C           CSTKEY      SETLLCUSREC
C           EXSR SFLPRC
C           EXSR SFLCHG
C           *IN03      IFEQ '0'
C           *IN04      ANDEQ'0'
C           WRITEFOOT1
C           WRITEHEAD
C           EXFMTPROMPT
C           END
C           END
C*
C           SETON           LR
```

Figure 98 (Part 1 of 4). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - SFLPRC                                     *
C*   PURPOSE   - PROCESS SUBFILE AND DISPLAY               *
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           SFLPRC    BEGSR
C           NXTPAG    TAG
C                   EXSR SFLCLR
C                   EXSR SFLFIL
C           SAMPAG    TAG
C                   WRITEFOOT2
C                   WRITEHEAD
C                   EXFMTSUBCTL
C           *IN95     IFEQ '1'
C           *IN71     IFEQ '0'
C                   GOTO NXTPAG
C                   ELSE
C                   GOTO SAMPAG
C                   END
C                   END
C                   ENDSR

```

Figure 98 (Part 2 of 4). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

WORKSTN File Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - SFLFIL                                     *
C*   PURPOSE   - FILL SUBFILE                             *
C*****
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C           SFLFIL    BEGSR
C           *IN21     DOWEQ'0'
C                   READ CUSREC                          71
C           *IN71     IFEQ '1'
C                   MOVE '1'          *IN21
C                   ELSE
C                   ADD 1             RECNUM
C                   MOVE *BLANK      SEL
C                   WRITESUBFILE    21
C                   END
C                   END
C                   ENDSR
C*****
C*   SUBROUTINE - SFLCLR                                     *
C*   PURPOSE   - CLEAR SUBFILE RECORDS                     *
C*****
C           SFLCLR    BEGSR
C                   MOVE '1'          *IN55
C                   WRITESUBCTL
C                   MOVE '0'          *IN55
C                   MOVE '0'          *IN21
C                   Z-ADD*ZERO        RECNUM 50
C                   ENDSR

```

Figure 98 (Part 3 of 4). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*   SUBROUTINE - SFLCHG                                     *
C*   PURPOSE   - CUSTOMER RECORD SELECTED                   *
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           SFLCHG     BEGSR
C           READCSUBFILE                               98
C           *IN98      IFEQ '0'
C           ZIPKEY     CHAINCUSREC                      71
C           EXFMTCUSDSP
C           END
C           ENDSR

```

Figure 98 (Part 4 of 4). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

The file description specifications identify the disk file to be searched and the display device file to be used (MLG265D). The continuation line for the WORKSTN file identifies the record format (SUBFILE) to be used as a subfile. The relative-record-number field (RECNUM) specified in positions 47 through 52 of the continuation line controls, which record within the subfile is being accessed.

The program displays the PROMPT record format and waits for the workstation user's response. F3 sets on indicator 03, which controls the end of the program. The zip code (ZIP) and name (NAME) are used as the key to position the MLGMSTL1 file by the SETLL operation. Notice that the record format name CUSREC is used in the SETLL operation instead of the file name MLGMSTL1.

The SFLPRC subroutine handles the processing for the subfile: clearing, filling, and displaying. The subfile is prepared for additional requests in subroutine SFLCLR. If indicator 55 is on, no action occurs on the display, but the main storage area for the subfile records is cleared. The SFLFIL routine fills the subfile with records. A record is read from the MLGMSTL1 file, the record count (RECNUM) is incremented, and the record is written to the subfile. This subroutine is repeated until either the subfile is full (indicator 21 on the WRITE operation) or end of file occurs on the MLGMSTL1 file (indicator 71 on the READ operation). When the subfile is full or end of file occurs, the subfile is written to the display by the EXFMT operation by the subfile-control record control format. The user reviews the display and decides:

- To end the program by pressing F3.
- To restart the subfile by pressing F4. The PROMPT record format is not displayed, and the subfile is displayed starting over with the same zip code.
- To fill another page by pressing the ROLL UP keys. If end of file has occurred on the MLGMST1 file, the current page is displayed again; otherwise, the subfile is cleared, and the next page is displayed.
- To display customer detail by entering X, and pressing ENTER. The user can then return to the PROMPT screen by pressing ENTER, display the subfile again by pressing F4, or end the program by pressing F3.

WORKSTN File Examples

In the following screen, the user responds to the initial prompt by entering a zip code and name.

```
11:07:56           Customer Master Inquiry           01/25/94
Enter Zip Code 26903
  Search Name CUMMINGS

ENTER - Continue      F3 - End Job
```

Figure 99. Prompt Screen for Zip Code Search

The user requests more information by entering X in the following screen.

```

11:09:20                Customer Master Inquiry                01/25/94

Zip Code  26903
Search Name CUMMINGS

Select
"X"      Zip Code  Number  Customer Name
        26903    00011  CUMMINGS
        26903    00012  DONLEAVY
        26903    00013  DREYFUS
        26903    00014  FREDERICKS
        26903    00015  RYERSON
X       26903    00016  SANDFORD
        26903    00017  STEVENS
        26903    00018  TALLBOY
        26903    00019  TORRENCE
        26903    00020  WALTERS
        27810    00021  GRAY
        27810    00022  GRAYSON
        27810    00023  HALIBURTON

ENTER - Continue      F3 - End Job      F4 - Restart Zip Code
    
```

Figure 100. Information Display for Zip Code Search

In the following screen, the user selects the appropriate function key to continue or end the inquiry.

```

11:09:20                Customer Master Inquiry                01/25/94

Customer 00016
Name     SANDFORD
Address  40 YONGE EAST
City     HAMILTON
State    WA   Zip Code 26903
A/R Balance      100.00

ENTER - Continue      F3 - End Job      F4 - Restart Zip Code
    
```

Figure 101. Detailed Information Display for Zip Code Search

Sample Program 6—Program-Described WORKSTN File with a FORMAT Name on Output Specifications

The following figures illustrate the use of a WORKSTN within FORMAT name on output specifications.

Table 10. List of Figures for FORMAT Name on Output Specifications	
Figure	Contents
Figure 102 below	DDS for display device file
Figure 103 on page 207	File description, input, calculation and output specifications

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : CUSINQ *
A* DESCRIPTION: DISPLAY FILE FOR FORMAT NAME ON OUTPUT *
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A
A      R ITMPMT
A
A          TEXT('INVENTORY INQUIRY PROMPT')
A          CF01(15 'END OF PROGRAM')
A          1 2'Inventory Inquiry Prompt'
A          2 2'Enter Item Number'
A          RECID          1  I 2 23DFT('A') DSPATR(ND PR)
A          ITEM          5  I 2 25
A 99
A          ERRMSG('Item Not Found' 99)
A      R ITMDTL
A          TEXT('INVENTORY DETAIL')
A          OVERLAY
A          5 2'Item No.'
A          5 14'Description'
A          5 41'Price'
A          5 53'Sold'
A          5 62'On hand'
A          ITEM          5      7 2
A          DESCRP       20     7 14
A          PRICE        8      7 41
A          PENDNG       5      7 53
A          ONHAND       5      7 62
    
```

Figure 102. DDS for Program-Described WORKSTN File within FORMAT Name on Output Specifications

The data description specifications for the display device file CUSINQ describe how the data sent from the RPG/400 program is displayed on the screen.


```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FCUSINQ CP F 50 WORKSTN KPASS *NOIND
FINVMSTL IF E K DISK
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IRcdname+....In.....*
ICUSINQ NS 03 2 CA
I.....Ext-field+.....Field+L1M1..PlMnZr...*
I 1 1 *IN15
I 2 2 RECID
I 3 7 ITEM
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C *IN03 IFEQ '1'
C ITEM CHAININVDTL 99
C END
C *IN15 IFEQ '1'
C MOVE '1' *INLR
C RETRN
C END
```

Figure 103 (Part 1 of 2). File Description, Input, Calculation, and Output Specifications for Program-Described WORKSTN File within FORMAT Name on Output Specifications

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OCUSINQ D 1P
0 OR 03
0 OR 99
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0 K6 'ITMPMT'
0 *IN99 1
0 D 03N99
0 K6 'ITMDTL'
0 ITEM 5
0 DESCRP 25
0 PRICE 33
0 PENDNG 38
0 ONHAND 43
```

Figure 103 (Part 2 of 2). File Description, Input, Calculation, and Output Specifications for Program-Described WORKSTN File within FORMAT Name on Output Specifications

On the output specifications, because the format name ITMPMT is conditioned by 1P, it is written to the file before any input operations take place. This format is also written to the file when indicator 03 or indicator 99 is on. If indicator 99 is on, the

error message that is defined in DDS is displayed. To pass indicator 99 on output, define the field *IN99 in the output record. The format ITMDTL is written to the file when indicator 03 is on and indicator 99 is not on. The end positions for the fields must be the same as the end positions defined on the DDS listing.

Sample Program 7–Variable Start Line

The following figures shows the program examples for a variable start line

<i>Table 11. List of Figures for Variable Start Line</i>	
Figure	Contents
Figure 104 below	DDS for display device file
Figure 105 on page 209	File description, extension, and calculation specifications

```
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
A*****
A* FILE NAME : INQUIRY *
A* DESCRIPTION: DISPLAY FILE FOR VARIABLE START LINE *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A
A PRINT
A R PROMPT SLNO(*VAR)
A MONTH 9A 0 6 15DSPATR(HI)
A DAY 2 0 6 26DSPATR(HI)
A YR 2 0 6 30DSPATR(HI)
A 6 45TIME DSPATR(HI)
```

Figure 104. DDS for Variable Start Line

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*   PROGRAM ID   - VARLINE                                     *
F*   PROGRAM NAME - VARIABLE START LINE DISPLAY              *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FINQUIRY CF E                                WORKSTN
F                                                KSLN  SLNFLD
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E                TABM  1 12 2 0 TABD  9  TABLE OF MONTHS
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C                MOVE UDAY      DAY
C                MOVE UYEAR     YR
C                UMONTH      LOKUPTABM      TABD          66
C                *IN66       IFEQ '1'
C                MOVE TABD      MONTH
C                END
C                Z-ADD6          SLNFLD  20
C*
C                EXFMTPROMPT
C*
C                MOVE '1'          *INLR
**
01JANUARY
02FEBRUARY
03MARCH
04APRIL
05MAY
06JUNE
07JULY
08AUGUST
09SEPTEMBER
10OCTOBER
11NOVEMBER
12DECEMBER

```

Figure 105. File Description, Extension, and Calculation Specifications for Variable Start Line

A start-line number (SLN) field determines the line number where a record format is written to a display file. SLN can be specified for both program-described and externally described files. To use a variable start line for a display file record format, specify the SLN option on the file continuation specifications. The DDS for the file must specify SLN0(*VAR) for one or more record formats. Only these record formats are affected by the value of the SLN field.

On output operations to the file, the value of the SLN field determines the line number where record formats are actually written. If the SLN field has a value of 1

WORKSTN File Examples

through 24, 1 is subtracted from the value, and the result is added to the line numbers specified in the DDS. The resulting values are used as the actual line numbers for writing the fields and constants specified in the DDS. However, the start line for the record format is the value of the SLN field. This means that the record format written occupies all the lines between the start of the format and the highest actual line number written to the display. If the SLN field has a value of 0, a format appears on the display as if an SLN field value of 1 were specified. If the value of the SLN field is negative or greater than 24, an RPG/400 1299 error message is issued. For more information, see the *Data Management Guide* and the *DDS Reference*.

In this example, the EXFMT operation uses a start-line number field (SLNFLD) with a value of 6. This causes the record format to be displayed starting at line 06, the output fields are written to line 11:

$(6(\text{SLNFLD}) - 1 + 6(\text{DDS start-line number}))$.

Figure 106 shows a display format specified with a variable start line.

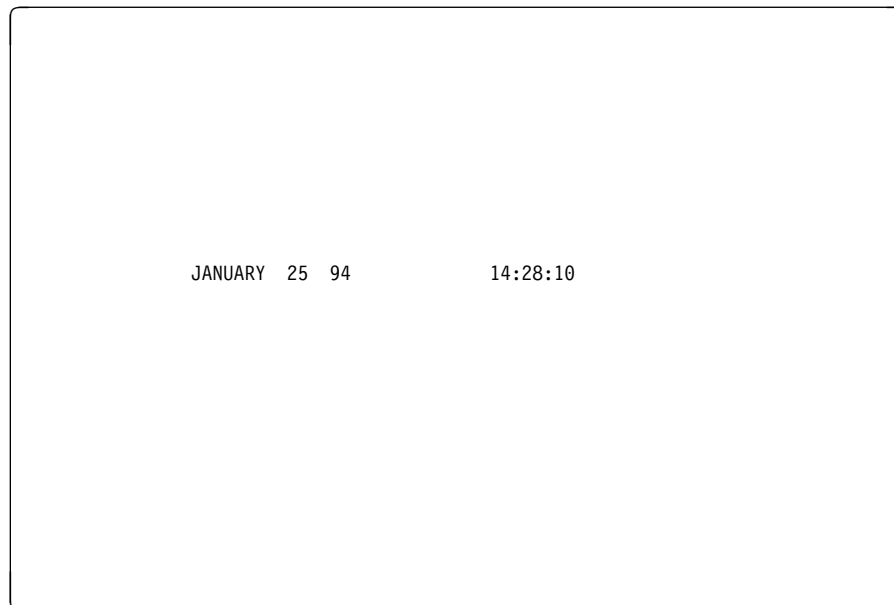


Figure 106. Prompt Screen for Variable Start Line

Sample Program 8–Read Operation with Time-Out

The following figures illustrate the program examples for READ operation with time-out.

<i>Table 12. List of Figures for READ Operation with Time-Out</i>	
Figure	Contents
Figure 107 below	DDS for display device file
Figure 109 on page 212	File description, input, and calculation specifications

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* FILE NAME : HOTELDSP *
A* DESCRIPTION: DISPLAY FILE FOR TIME OUT EXAMPLE *
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++
A
A          R REQUEST          INVITE
A
A          OVERLAY
A          ROOM          5A I 10 46DSPATR(HI)
A          10 26'Enter Room Number:'
A          DSPATR(HI)
    
```

Figure 107. DDS Read Operation with Time-Out

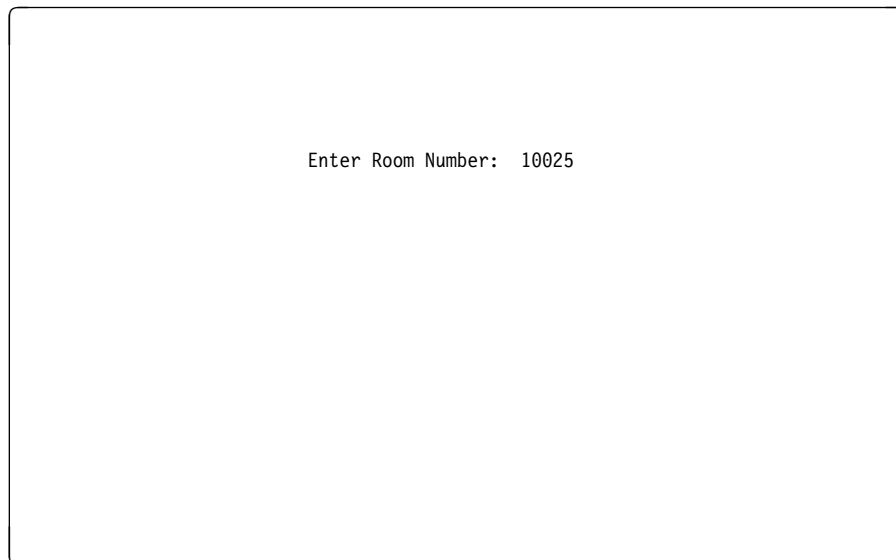


Figure 108. Sample Screen for Time-out

WORKSTN File Examples

Figure 109 shows an example of file description, input, and calculation specifications for READ operation with time-out.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F*   PROGRAM ID   - TIMEOUT                                     *
F*   PROGRAM NAME - TIME OUT ON READ                         *
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FHOTELDSPCF  E                                WORKSTN
F                                     KNUM          1
F                                     KINFDS FEEDBK

IDpname....NODsExt-file++.....OccrLen+.....*
IFEEDBK      DS
I.....Ext-field+.....PFromTo++DField+.....*
I                                     *STATUS  STATUS

CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C                                     WRITEREQUEST
C                                     READ HOTELDSP          9950
C                                     EXSR ERRCHK
C                                     MOVE '1'          *INLR
C*****
C*   SUBROUTINE - ERRCHK                                     *
C*   PURPOSE    - CHECK STATUS FOR MAX WAIT                 *
C*****
C           ERRCHK  BEGSR
C           STATUS  IFEQ 1331
C           MOVE 'SIGNOFF'  CMD    7
C           Z-ADD7          LEN    155
C           CALL 'QCMDEXC'
C           PARM          CMD
C           PARM          LEN
C           END
C           ENDSR

```

Figure 109. File Description, Input, and Calculation Specifications for Read Operation with Time-Out

This program causes the work station to be signed off, when no workstation activity has occurred during a specified length of time.

- In the DDS for the display file H0TELDSP, the keyword INVITE is specified for all formats. You specify a length of time to wait with the WAITRCD parameter on the CRTDSPF (or CHGDSPF) command to create (or change) this file.
- In the file specifications, the file H0TELDSP is specified as a WORKSTN file with the option NUM. RPG treats the file as a multiple-device file.
- In the input specifications, the *STATUS subfield of the file information data structure is named STATUS.
- The WRITE operation puts format REQUEST on the work station and, because of the keyword INVITE, makes the work station an invited device.
- The READ-by-file-name operation to the file H0TELDSP waits for the length of time specified on the WAITRCD parameter for a response from the invited device.
- If no response comes in time, error indicator 99 is set on and the program continues with the next operation.
- The next operation performs the ERRCHK subroutine. This subroutine checks the STATUS subfield of the file information data structure. Status code 1331 indicates the READ operation timed out, and the ERRCHK subroutine signs the work station off. Other status codes produce other results.

Note: This example is not a complete program.

Chapter 9. Data Field Formats and Data Structures

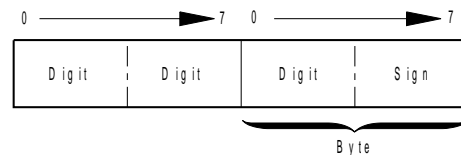
This chapter describes how the RPG/400 program works with data that is stored in fields in data files. Within these files, the fields can be grouped together into data structures.

Format of Fields in Files

The input and output fields of an RPG/400 program can be in character, zoned-decimal, packed-decimal, or binary format. A leading or trailing sign can be specified with zoned-decimal format only. All numeric input fields (unless they are in a data structure) are converted by the compiler to packed-decimal format for internal processing. The program runs in the same way whether numeric data is in packed-decimal format, zoned-decimal format, or binary format. However, the system processes arithmetic calculations more efficiently if the data is in packed-decimal format. Subfields within a data structure are always carried in the format specified by the subfield specification.

Packed-Decimal Format

Packed-decimal format means that each byte of storage (except for the low-order byte) can contain two decimal numbers. Each byte (except the low-order byte) is divided into two 4-bit digit portions. The low-order byte contains one digit in the leftmost portion and the sign (+ or -) in the rightmost portion. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. The packed-decimal format looks like this:



The sign portion of the low-order byte indicates whether the numeric value represented in the digit portions is positive or negative. Figure 110 on page 219 shows what the decimal number 8191 looks like in packed-decimal format.

For a program-described file, you specify packed-decimal input, output, and array or table fields with the following entries:

Packed-decimal input field: Specify P in position 43 of the input specifications.

Packed-decimal output field: Specify P in position 44 of the output specifications. This position must be blank if editing is specified.

Packed-decimal array or table field: Specify P in position 43 or position 55 of the extension specifications. Arrays and tables loaded at compile time cannot be in packed-decimal format.

For an externally described file, the data format is specified in position 35 of the data description specifications.

Format of Fields in Files

Use the following formula to find the length in digits of a packed-decimal field:

$$\text{Number of digits} = 2n - 1,$$

...where n = number of packed input record positions used.

This formula gives you the maximum number of bytes you can represent in packed-decimal format; the upper limit is 30.

Packed fields can be up to 16 bytes long. The chart in Table 13 shows the packed equivalents for zoned-decimal fields up to 16 digits long:

<i>Table 13. Packed Equivalents for Zoned-Decimal Fields up to 16 Digits Long</i>	
Zoned-Decimal Length in Digits	Number of Bytes Used in Packed-Decimal Field
1	1
3	2
5	3
.	.
.	.
.	.
29	15
30	16

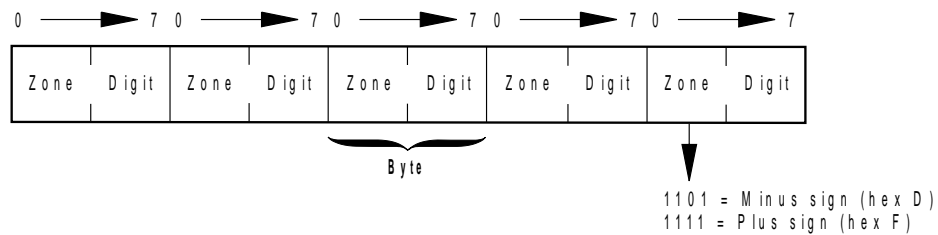
For example, an input field read in packed-decimal format has a length of five positions (as specified on the input or data description specifications). The number of digits in this field equals $2(5) - 1$ or 9. Therefore, when the field is used in the calculation specifications, the result field must be nine positions long.

When a packed-decimal field in one program is converted to a zoned-decimal field in another program, the zoned-decimal field always contains an odd number of bytes. If a field is in packed-decimal format in one program and then is unpacked in another program, the field length can increase by 1. If a field is packed and then unpacked in the same program, the field length does not change. This must be considered when fields are packed for storage on an intermediate device and then used by another program.

Zoned-Decimal Format

Zoned-decimal format means that each byte of storage can contain one digit or one character. Any character or numeric field can be read in zoned-decimal format. In the zoned-decimal format, each byte of storage is divided into two portions: a 4-bit zone portion and a 4-bit digit portion.

The zoned-decimal format looks like this:



The zone portion of the low-order byte indicates the sign (+ or -) of the decimal number. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. In zoned-decimal format, each digit in a decimal number includes a zone portion; however, only the low-order zone portion serves as the sign. Figure 110 on page 219 shows what the number 8191 looks like in zoned-decimal format.

You must also consider the change in field length when coding the end position in positions 40 through 43 of the output specifications. To find the length of the field after it has been packed, use the following formula:

$$\text{Field length} = \frac{n}{2} + 1$$

...where n = number of digits in the zoned decimal field.

(Any remainder from the division is ignored.)

For a program-described file, zoned-decimal format is specified by a blank in position 43 of the input specifications, in position 44 of the output specifications, or in position 43 or 55 of the extension specifications. For an externally described file, the data format is specified in position 35 of the data description specifications.

RPG/400 internally converts zoned decimal data into character data. During this conversion, errors from decimal data are automatically corrected. Decimal data errors can only be detected for fields defined in packed decimal format.

Binary Format

Binary format means that the sign (+ or -) is in the leftmost bit of the field and the integer value is in the remaining bits of the field. Positive numbers have a zero in the sign bit; negative numbers have a one in the sign bit and are in twos complement form. In binary format, each field must be either 2 or 4 bytes long.

Program-Described File

Every input field read in binary format is assigned a field length (number of digits) by the compiler. A length of 4 is assigned to a 2-byte binary field; a length of 9 is assigned to a 4-byte binary field. Because of these length restrictions, the highest decimal value that can be assigned to a 2-byte binary field is 9999 and the highest decimal value that can be assigned to a 4-byte binary field is 999 999 999.

Format of Fields in Files

For program-described files, specify binary input, binary output, and binary array or table fields with the following entries:

- *Binary input field:* Specify B in position 43 of the input specifications.
- *Binary output field:* Specify B in position 44 of the output specifications. This position must be blank if editing is specified.

The length of a field to be written in binary format cannot exceed nine digits. If the length of the field is from one to four digits, the compiler assumes a binary field length of 2 bytes. If the length of the field is from five to nine digits, the compiler assumes a binary field length of 4 bytes.

Because 2-byte input field in binary format is converted by the compiler to a four-digit decimal field, the input value may be too large. If it is, the leftmost digit of the number is dropped. For example, an input field has a binary value of hex 7000. The compiler converts this to 28 672 in decimal. The 2 is dropped and the result is 8672.

- *Binary array or table field:* Specify B in position 43 and/or position 55 of the extension specifications. Arrays and tables loaded at compile time cannot be in binary format.

Externally Described File

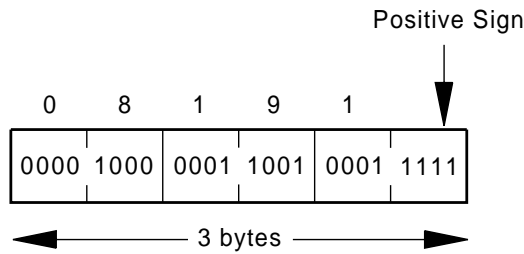
For an externally described file, the data format is specified in position 35 of the data description specifications. The number of digits in the field is exactly the same as the length in the DDS description. For example, if you define a binary field in your DDS specification as having 7 digits and 0 decimal positions, the RPG/400 compiler handles the data like this:

1. The field is defined as a 4-byte binary field in the input specification
2. A Packed(7,0) field is generated for the field in the RPG/400 program.

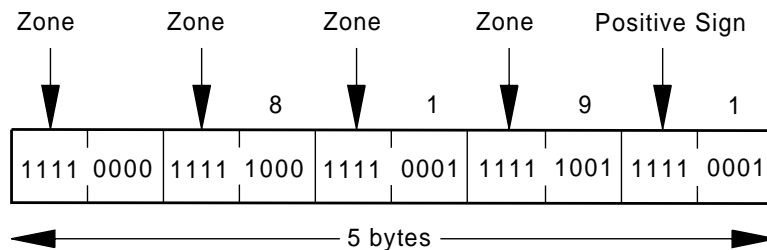
If you want to retain the complete binary field information, redefine the field as a binary subfield in a data structure.

Figure 110 on page 219 shows what the decimal number 8191 looks like in various formats.

Packed Decimal Format:



Zoned Decimal Format:¹



Binary Format:²

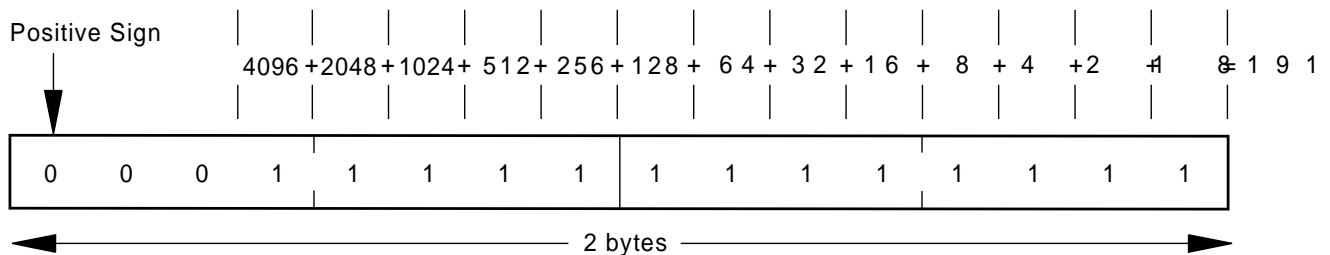


Figure 110. Binary, Packed, and Zoned-Decimal Representation of the Number 8191

¹ If 8191 is read into storage as a zoned-decimal field, it occupies 4 bytes. If it is converted to packed-decimal format, it occupies 3 bytes. When it is converted back to zoned-decimal format, it occupies 5 bytes.

² To obtain the numeric value of a positive binary number add the values of the bits that are on (1), do not include the sign bit. To obtain the numeric value of a negative binary number, add the values of the bits that are off (0) plus one (the sign bit is not included).

Signs

The RPG/400 program ensures that a consistent plus or minus sign is present for all numeric fields. The standard signs for all packed and zoned numeric fields are hexadecimal F for plus and hexadecimal D for minus.

External Formats

When a sign is written out for numeric fields, the sign (+ or –) is included in the units position of the data field unless editing has been done. See the *RPG/400 Reference*.

You can specify an alternative sign format for zoned-decimal format. In the alternative sign format, the numeric field is immediately preceded or followed by a + or – sign. A plus sign is a hexadecimal 4E, and a minus sign is a hexadecimal 60.

For program-described files, specify preceding (L entry) or following (R entry) plus or minus signs in the following positions:

<i>Input field:</i>	Position 43 of the input specifications
<i>Output field:</i>	Position 44 of the output specifications
<i>Array or table field:</i>	Position 43 and/or position 55 of the extension specifications.

When an alternative sign format is specified, the field length must include an additional position for the sign. For example, if a field is 5 digits long and the alternative sign format is specified, a field length of 6 positions must be specified.

Internal Format

All numeric fields, except subfields of a data structure, are stored in packed-decimal format for internal processing. In packed-decimal format, the sign is stored in the last 4 bits of the rightmost byte of the field. See Figure 110 on page 219.

Data Structures

The RPG/400 program allows you to define an area in storage and the layout of the fields, called subfields, within the area. This area in storage is called a data structure. You can use a data structure to:

- Define the same internal area multiple times using different data formats
- Operate on a field and change its contents
- Divide a field into subfields without using the MOVE or MOVEL operation codes
- Define a data structure and its subfields in the same way a record is defined
- Define multiple occurrences of a set of data
- Group non-contiguous data into contiguous internal storage locations.

In addition, there are three special data structures, each with a specific purpose:

- A data area data structure (identified by a U in position 18 of the data structure statement)
- A file information data structure (referred to by the keyword INFDS on a file description specifications continuation line)
- A program-status data structure (identified by an S in position 18 of the data structure statement).

Data structures can be program-described or externally described.

A program-described data structure is identified by a blank in position 17 of the data structure statement. The subfield specifications for a program-described data structure must immediately follow the data structure statement.

An externally described data structure, identified by an E in position 17 of the data structure statement, has subfield descriptions contained in an externally described file with one record format. At compile time, the RPG/400 program uses the external name to locate and extract the external description of the data structure subfields. An external subfield name can be renamed in the program, and additional subfields can be added to an externally described data structure in the program.

For examples of data structures, see “Data Structure Examples” on page 226.

Format of Data Structure Subfields in Storage

Subfields in a data structure are stored in the format specified in position 43 of the data structure subfield specifications. The possible entries for a program-described data structure are:

Entry	Explanation
Blank	Subfield is in zoned-decimal format or is character data, depending on the entry in position 52 of the subfield specifications.
P	Subfield is in packed-decimal format.
B	Subfield is in binary format.

Because the subfields of a data structure are maintained in the format specified, the compiler generates the necessary conversions to process the required function. These conversions can occur at the following times:

- When a record is being read
- At detail or total calculation time
- At detail or total output time.

The rules for determining the length of a subfield in packed-decimal format, zoned-decimal format, and binary format are the same as those for determining the length of a field in packed-decimal format, zoned-decimal format, and binary format. (See “Packed-Decimal Format” on page 215, “Zoned-Decimal Format” on page 216, and “Binary Format” on page 217.)

Data Structure Statement Specifications

Data structure statements are defined on the input specifications and must follow all input specifications for records. The specifications for data structure statements are:

Table 14 (Page 1 of 2). Specifications For Data Structure Statements

Position	Entry
6	I
7-12	Name of the data structure being defined. This entry is optional for a program-described data structure, and is required for an externally described data structure, a file information data structure (INFDS), and a data area data structure.

Table 14 (Page 2 of 2). Specifications For Data Structure Statements

Position	Entry
13-16	Blank
17	<i>Blank</i> : Program-described data structure. <i>E</i> : Externally described data structure. The data structure subfield definitions are retrieved from an externally described record format.
18	<i>Blank</i> : Other than a program status, data area or initialized data structure. <i>I</i> : Globally initialized data structure. <i>S</i> : Program-status data structure. <i>U</i> : Data area data structure.
19-20	DS
21-30	<i>Blank</i> : The data structure is program described. <i>Entry</i> : This is the name of the file whose first record format contains the field descriptions used as the subfield descriptions for this data structure.
31-43	Blank
44-47	<i>Blank</i> : A single occurrence data structure. <i>nnnn</i> : A number (right-adjusted) indicating the number of occurrences of the data structure. Note : This entry must be blank for a data area data structure, a file information data structure, and a program-status data structure.
48-51	Length of data structure (optional). This entry must be right-adjusted.
52-74	Blank

Rules for Specifying Data Structure Statements

Remember the following when you specify data structure statements:

- The data structure name must be a symbolic name with a maximum of six characters. The name can appear on only one data structure specification, cannot be a lookahead field, and can be specified anywhere a character field is allowed.
- All entries for one data structure and its subfields must appear together; they cannot be mixed with entries for other data structures.
- The data structure length is determined by the first specification in the program that defines a length in one of the preceding ways. Subsequent conflicting lengths are incorrect. The length of a data structure is one of the following:
 - The length specified in the input-field specifications if the data structure name is an input field
 - The length specified in positions 48 through 51 of the data structure statement
 - The highest To position of a subfield within a data structure if the data structure name is not an input field.
- A compile-time or prerun-time array cannot be used in a data area data structure or in a multiple-occurrence data structure.

- Data structures are character data and can be from 1 to 9999 characters in length.
- A data structure and a subfield of a data structure cannot have the same name.

Multiple Occurrence Data Structure

A multiple-occurrence data structure is a data structure whose definition is repeated in a program to form a series of data structures with identical formats. You specify the number of occurrences of a data structure in positions 44 through 47 of the data structure statement. When positions 44 through 47 do not contain an entry, the data structure is not a multiple-occurrence data structure. All occurrences of a data structure have the same attributes and can be referred to individually. The OCUR operation code, which can only be used with a multiple-occurrence data structure, allows you to specify which occurrence of a data structure is used for subsequent operations within the program.

Note: Multiple occurrences are not allowed for a data area, file information, or program-status data structure.

For examples on multiple-occurrence data structures, see “Data Structure Examples” on page 226.

Special Data Structures

Special data structures include:

- Data area data structures
- File information data structures (INFDS)
- Program-status data structures.

Data Area Data Structure

A data area data structure, identified by a U in position 18 of the data structure statement, indicates to the RPG/400 program that it should read in and lock the data area of the same name at program initialization and should write out and unlock the same data area at the end of the program. Data area data structures, as in all other data structures, have the type character. A data area read into a data area data structure must also be character. The data area and data area data structure must have the same name unless you rename the data area within the RPG/400 program by using the *NAMVAR DEFN statement.

You can specify the data area operations (IN, OUT, and UNLCK) and have the type for a data area that is implicitly read in and written out. Before you use a data area data structure with these operations, you must specify that data area in the result field of the *NAMVAR DEFN statement.

A data area data structure cannot be specified in the result field of a PARM operation.

If you specify blanks for the data area data structure (positions 7 through 12 of the input specifications line that contains a U in position 18), the RPG/400 program uses a local data area. To provide a name for a local data area, use the *NAMVAR DEFN operation, with *LDA in factor 2 and the name in the result field.

Data Structure Subfield Specifications

For general information on data areas, see Chapter 11, “Communicating with Objects in the System.”

File Information Data Structure

You can specify a file information data structure (defined by the keyword INFDS on a file description specifications continuation line) for each file in the program. This provides you with status information on the file exception/error that occurred. The file information data structure name must be unique for each file. A file information data structure contains predefined subfields that provide information on the file exception/error that occurred. For a discussion of file information data structures and their subfields, see “Exception/Error Handling” on page 70.

Define and name a file information data structure on a file description specifications continuation line with the following entries:

Table 15. Entries to Define and Name a File Information Data Structure

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation line)
54-59	INFDS (identifies this data structure as the file information data structure)
60-65	Name of the file information data structure.

Program-Status Data Structure

A program-status data structure, identified by an S in position 18 of the data structure statement, provides program exception/error information to the program. For a discussion of program-status data structures and their predefined subfields, see “Exception/Error Handling” on page 70.

Data Structure-Subfield Specifications

The subfields of a program-described data structure must immediately follow the data structure specification statement to which they apply. The subfields of an externally described data structure are described externally to the RPG/400 program. The subfield specifications are brought into the RPG/400 program at compilation. The subfields of an externally described data structure can be renamed or additional subfield specifications can appear following the data structure statement. All renamed and initialized external subfields must precede any additional subfield specifications. To add subfields to an externally described data structure, follow the same rules as for subfields for a program-described data structure. The internally described subfields are added to the retrieved descriptions.

The specifications for subfields are as follows:

Table 16 (Page 1 of 2). Specifications for Subfields

Position	Entry
6	I
7	Blank

Table 16 (Page 2 of 2). Specifications for Subfields

Position	Entry
8	I: Indicates an initialized subfield. (Specify the initialization value in positions 21-42 or leave blank for default initialization value.)
9-20	Blank
21-42	positions 21-26: Named constant initialization value if position 8 contains an I. Leave any remaining positions blank. or positions 21-42: Literal initialization value if position 8 contains an I. or positions 21-42: Blank for default initialization value if position 8 contains an I. or positions 21-30: External name to rename a subfield in an externally described data structure. (Specify the name to be used in the program in positions 53 through 58.) Leave any remaining positions blank.
43	<i>P</i> : Indicates that the subfield is in packed-decimal format. <i>B</i> : Indicates that the subfield is in binary format. <i>Blank</i> : Indicates that the subfield is in zoned-decimal format, or is character data.
44-47 48-51	<i>1- to 4-digit numbers</i> : Positions 44 through 47 contain the beginning position, and positions 48 through 51 contain the end position of the subfield. These entries must be right-adjusted; leading zeros can be omitted. or <i>Keywords</i> : For a program-status data structure or a file information data structure (INFDS), place a special keyword (left-adjusted) in this position. A keyword can start at position 44 and extend through to position 51. See "Exception/Error Handling" on page 70 for the keywords and their descriptions.
52	<i>0-9</i> : Indicates the number of decimal positions in a numeric field or an array. <i>Blank</i> : Indicates a character field. Note : This position must contain an entry for a numeric subfield. However, an entry is not required for an array. If an entry is made for an array, the entry must be the same as that specified in the extension specifications.
53-58	The subfield name. Note : If an array is specified as a subfield name, the length indicated in positions 44 through 51 must equal the entire amount of storage required to store the array (for example, 10 binary half-word elements require 20 bytes of storage).
59-74	Blank

Rules for Subfield Specifications

Remember the following when you specify subfield specifications:

- If the length (positions 44 through 51) or decimal positions (position 52) for the subfield differ from prior definitions in the program, the first definition is used and subsequent conflicting definitions are incorrect.
- If the To position (48 through 51) specified for a subfield is larger than the defined length of an input field of the same name or the defined length of the data structure, the subfield specification is incorrect.
- To redefine subfields, specify the same or part of the same From and To positions (44 through 51) for another subfield in the same data structure.
- To define a single position subfield, enter the same number in both positions 44 through 47 and positions 48 through 51.
- Overlapping subfields cannot be used in the same calculation specification.
- If an array or array element with a variable index is specified in the calculation specifications in factor 1, factor 2, or the result field, the entire array is used to determine whether overlap exists.
- Before packed, zoned, or binary numeric subfields are used in arithmetic or editing operations, you must ensure that they are initialized with numeric data.
- An input field name cannot:
 - Appear as both a subfield name and a data structure name
 - Appear more than once as a subfield name.
- The following calculation operations are checked for overlapping subfields:
 - Factor 1 and the result field, and factor 2 and the result field of the ADD, SUB, MULT, DIV, Z-ADD, and Z-SUB operations. Factor 1 and factor 2 of the preceding operations may overlap.
 - Factor 2 and the result field of a MOVE, MOVEL, or MOVEA operation are checked for overlap.
 - Factor 2 and the result field and factor 1 and the result field of a PARM operation are checked for overlap.

Data Structure Examples

Figure 111 on page 227 through Figure 116 on page 236 show some typical uses for data structures.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IFILEIN NS 01 1 CA 2 CB
I.....PFromTo++DField+L1M1FrP1MnZr...*
I 3 18 PARTNO
I 19 29 NAME
I 30 40 PATNO
I 41 61 DR
IDcname....NODsExt-file++.....OccrLen+.....*
IPARTNO DS
I.....Ext-field+.....PFromTo++DField+.....*
I 1 4 MFG
I 5 10 DRUG
I 11 13 STRNTH
I 14 160COUNT
I*

```

Figure 111. Using a Data Structure to Define Subfields within a Field

The data structure subfields can be referred to by the PARTNO name or by the subfields MFG, DRUG, STRNTH, or COUNT.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ITRANSACTNS 01 1 C1 2 C2
I.....PFromTo++DField+L1M1FrP1MnZr...*
I 3 10 PARTNO
I 11 160QTY
I 17 20 TYPE
I 21 21 CODE
I 22 25 LOCATN
IDcname....NODsExt-file++.....OccrLen+.....*
IPRTKEY DS
I.....Ext-field+.....PFromTo++DField+.....*
I 1 4 LOCATN
I 5 12 PARTNO
I 13 16 TYPE
I*

```

Figure 112. Using a Data Structure to Group Fields

When you use a data structure to group fields, fields from non-adjacent locations on the input record can be made to occupy adjacent internal locations. The area can then be referred to by the data structure name or individual subfield name.

Data Structure Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* A multiple-occurrence data structure is used to accumulate a
I* series of totals for specific codes, and the totals of each of
I* the occurrences of the data structure are written.
I* The program-described data structure, TOTDS, has 99 occurrences
I* (positions 46 and 47). The length of the data structure can be
I* specified in positions 48 through 51.
I*
ID$NAME....NOD$EXT-FILE+.....OCCURLEN+.....*
ITOTDS      DS          99
I.....EXT-FIELD+.....PFRMTO+DFIELD+.....*
I          1  50TOTCNT
I          6  12TOT1
I          13 20TOT2
I*

```

Figure 113 (Part 1 of 4). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 1

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* A numeric code field, CODE, contains a value of 01 through 99.
C* This value is different each time the OCUR operation is processed.
C* When the OCUR operation is processed, the CODE field is used to
C* set the current occurrence of TOTDS. If the OCUR operation is
C* successful, the program branches to the ADDR TN subroutine where
C* a record count is made and input values are added to the data
C* structure subfields. If the CODE field contains a value other
C* than 01 through 99, indicator 25 is set on and the program
C* branches to BADCOD.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          CODE      OCUR TOTDS          25
C  25          GOTO BADCOD
C          EXSR ADDR TN
C          "
C          "
C          BADCOD  TAG
C          "
C          "
C

```

Calculations

Calculations

Figure 113 (Part 2 of 4). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 1

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* When the totals for the specific codes in the multiple-occurrence
C* data structure are to be written out, exception output is used.
C* The EXCPT PRTHDG operation causes all exception lines in the
C* output specifications with the name PRTHDG to be written. The
C* do group initially sets field X to 1. The value in X sets the
C* current occurrence of TOTDS. The Z-ADD operation adds TOTCNT to
C* a field of zeros and places the sum in the result field TOTCNT.
C* If TOTCNT contains a plus value, indicator 27 is set on.
C* The EXCPT PRTDS operation causes the current occurrence of the
C* data structure to be written. If overflow occurs while the
C* current occurrence of the data structure is being written, the
C* OF indicator is set on, a page skip occurs, and all exception
C* lines in the output specifications with the name PRTHDG are
C* written. The SETOF operation sets off the OF indicator.
C*
C* The Do group continues processing until field X is greater than
C* 99, the maximum number of occurrences for the multiple-occurrence
C* data structure. When X is greater than 99, control passes to the
C* next statement following the END statement.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C          EXCPTPRTHDG
C          DO 99          X          30
C          X          OCUR TOTDS
C          Z-ADDTOTCNT    TOTCNT    27
C 27          EXCPTPRTDS
C OF          EXCPTPRTHDG
C OF          SETOF          OF
C          END
C          "
C          "
C          Calculations
C

```

Figure 113 (Part 3 of 4). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 1

Data Structure Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The ADDR TN subroutine updates the current occurrence of the
C* multiple-occurrence data structure subfields.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          ADDR TN      BEGSR
C          ADD  1          TOTCNT
C          ADD  FLD1       TOT1
C          ADD  FLD2       TOT2
C*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT  E  206          PRTHDG
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
O          "
O          "  Entries for Report Title
O          "
O          "
O          E  2          PRTHDG
O          "
O          "  Entries for Report Column Headings
O          "
O          "
O          E          PRTDS
O          X          10
O          TOTCNTZ    20
O          TOT1 J     35
O          TOT2 J     50
O*

```

Figure 113 (Part 4 of 4). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 1

In the following example, a multiple-occurrence data structure, TOTDS, is again used to accumulate a series of totals for specific codes and the totals of each of the occurrences of the data structure are written. There are 70 codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* A compile-time array, ARC, is specified in the extension
E* specifications. It has 70 entries. There are 10 entries in
E* each record, and each array element is 6 positions long. The
E* relative location of the alphanumeric code in the array (for
E* example the 37th entry) sets the current occurrence of the data
E* structure.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E                ARC    10 70 6                ARRAY OF CODES
```

Figure 114 (Part 1 of 3). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 2

Data Structure Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The Z-ADD operation sets field X to one. The LOKUP operation
C* starts at the first element of ARC and searches until it finds
C* the first element equal to the code in ACODE. The ACODE field
C* is a character field of 6 characters. The index value, X, is
C* set to the position number of the element located. If the LOKUP
C* does not find an element equal to ACODE, indicator 20 is not set
C* on and the GOTO operation conditioned by N20 branches to the
C* BADCOD TAG. If LOKUP does find an element equal to ACODE, the
C* OCUR operation uses the value in X to set the current occurrence
C* of TOTDS and the program branches to the ADDR TN subroutine, where
C* a record count is made and input values are added to the data
C* structure subfields. The ADDR TN subroutine is not shown. If the
C* occurrence is outside the valid range for the data structure,
C* indicator 26 is set on, and the program branches to the ENDPRT TAG.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C          Z-ADD1          X          30
C          ACODE          LOKUPARC,X          20
C N20          GOTO BADCOD
C          X          OCUR TOTDS          26
C 26          GOTO ENDPRT
C          EXSR ADDR TN
C          "
C          "
C          BADCOD          TAG
C          "
C          "
C          ENDPRT          TAG
C          "
C          "
C          "

```

Figure 114 (Part 2 of 3). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 2

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0*
0* The calculations to print the data structure are not shown.
0* Only part of the output specifications is shown. The PRTDS
0* statement uses the value of field X, which contains the current
0* occurrence of the data structure, as an index to print the
0* corresponding alphanumeric code.
0*
0Name++++DFBASbSaN01N02N03Excnam.....*
0      E      PRTDS
0.....N01N02N03Field+YBEnd+PCConstant/editword+++++...*
0      ARC,X      10
0      TOTCNTZ    20
0      TOT1 J     35
0      TOT2 J     50

```

Figure 114 (Part 3 of 3). Using a Multiple Occurrence Data Structure to Accumulate Totals—Example 2

Data Structure Examples

Both programs (1 and 2) shown in Figure 115 below use data area data structures (defined by the U in position 18 of the input specifications). Program 1 uses the subfields of the data structure to accumulate a series of totals. Program 2 then uses the totals in the subfields to do calculations.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I* PROGRAM 1
I*
IDname....NODsExt-file++.....OccrLen+.....*
ITOTALS      UDS
I.....Ext-field+.....PFromTo++DField+.....*
I              1      82TOTAMT
I              9     182TOTGRS
I             19     282TOTNET
I*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C
C          "
C          "
C          "
C          ADD  AMOUNT  TOTAMT
C          ADD  GROSS   TOTGRS
C          ADD  NET     TOTNET
C*

```

Calculations

Figure 115 (Part 1 of 2). Data Area Data Structures

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I* PROGRAM 2
I*
ID$NAME...NOD$EXT-FILE++.....OCCRLEN+.....*
ITOTALS      UDS
I.....Ext-field+.....PFromTo++DField+.....*
I              1      82TOTAMT
I              9     182TOTGRS
I             19     282TOTNET
I*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C
C              "
C              "
C              "
C      AMOUNT2  COMP TOTAMT              9191
C      GROSS2   COMP TOTGRS              9292
C      NET2     COMP TOTNET              9393
C              "
C              "
C*

```

Calculations

Figure 115 (Part 2 of 2). Data Area Data Structures

Data Structure Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
ID$NAME...NOD$EXT-FILE++.....OCCRLen+.....*
ID$ONE      E D$EXTREC
I.....Ext-field+.....PFromTo++DField+.....*
I          CHARACTER          CHAR
I          1 16 CHZON

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.NAME+++++RLEN++TDpB.....FUNCTIONS+++++*
A          R RECORD          TEXT('EXTERNALLY DESCRIBED RECORD')
A          CHARACTER          10
A          ZONED              6S 2
A          PACKED             4P 0
A          BINARY             4B 0
A*

```

Figure 116. Renaming Subfields in an Externally Described Data Structure

On the data structure statement shown in Figure 116, positions 7 through 12 contain the name of the data structure being defined (DSONE), position 17 contains an E to denote externally described, and positions 19 and 20 contain DS to denote data structure. Positions 21 through 30 contain the name of the file (EXTREC) whose first record format contains the field descriptions used as the subfield descriptions for this data structure (RECORD).

On the first data description specification, position 17 contains an R to denote record format and positions 19 through 28 contain the name of the record format (RECORD). On subsequent data description specifications, positions 19 through 28 contain the names of the fields (CHARACTER, ZONED, PACKED, and BINARY).

Fields in a data structure can also be redefined for program use. Fields CHARACTER and ZONED are also described as one field (CHZON) in the input specifications.

In the RPG/400 program, a field name can contain no more than 6 characters. Therefore, the field name CHARACTER is renamed CHAR in the input specifications. The data structure then uses CHAR as the subfield name.

Chapter 10. Named Constants, Initialization, and SAA Data Types

This chapter describes how you can use named constants and SAA data types in your RPG/400 program. The chapter also addresses initialization of an RPG/400 program.

Named Constants

You can give a name to a constant. This name represents a specific value which cannot be changed when the program is running.

Rules for Named Constants

- Named constants can be specified in Factor 1 and Factor 2 in the calculation specifications and in the Field Name, Constant, or Edit Word fields in the output specifications. They can also be used as array indexes and as the format name in a WORKSTN output specification or as initialization values in an input specification.
- The named constant has no inherent type. That is, no precision is implied by the definition. Actual precision is defined by the context that is specified.
- The named constant can be defined anywhere in the input specifications.
- Character named constants must begin and end with a single quotation mark (').
- If an alphanumeric constant, transparent or hexadecimal literal is specified, then it can be continued to the constant field of the next line by coding a hyphen (-) at the end of the constant instead of an apostrophe. If a numeric constant is specified, then it can be continued to the constant field of the next line by coding a hyphen (-) at the end of the constant immediately following the last digit.
 - The hyphen can be specified in any position on the field.
 - The hyphen works the same way as the minus sign when continuing commands in CL programs. Any blanks in the next input record that follow the leading apostrophe, and precede the first non-blank character, are included in the named constant.
 - Hyphens are permitted in the first position of a named constant literal to allow double-byte data to be moved. See “Moving Bracketed Double-byte Data and Deleting Control Characters (SUBR40R3)” on page 263 for more information on moving double-byte data.
 - For hexadecimal constants, the number of hexadecimal digits in a continuation line does not have to be even. However, the total number of hexadecimal digits in the entire constant must be even. (Hexadecimal literals must begin with an uppercase X, followed by a single quotation mark (X'). Hexadecimal literals, like character constants, end with a single quotation mark ('.))
 - The next input specification must contain an entry in the constant entry alone (apart from an I in position 6). If an alphanumeric, transparent literal

Named Constants

constant, or hexadecimal literal is continued, the first character of each continuation (position 21) must contain an apostrophe.

- The constant can be continued as many times as desired so long as the total length of the constant does not exceed 256 single-byte characters. A numeric constant cannot be longer than 30 digits, with a maximum of nine positions to the right of the decimal point. A hexadecimal literal cannot be longer than 512 hexadecimal digits representing 256 bytes and must contain an even number of digits.
- The named constant represents the constant that is the concatenation of all constants on the main named constant specification and continuation lines.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
```

```
I*
```

```
I* The following is an example of a character named constant:
```

```
I*
```

```
I.....Namedconstant+++++++C.....Fldnme.....
```

```
I*
```

```
I          'ABCDEFG'          C          CHAR
```

```
I*
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
```

```
I*
```

```
I* The following is an example of a continued transparent  
I* constant. The Shift Out (SO) and Shift In (SI) characters  
I* are represented by o and i.
```

```
I*
```

```
I.....Namedconstant+++++++C.....Fldnme.....
```

```
I*
```

```
I          'oK1K2K3i-'          C          TRANS
```

```
I          'oK4K5i'
```

```
I*
```

Figure 117 (Part 1 of 3). The Use of Named Constants


```

*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I*
I* The following is an example of a continued character named
I* constant. The blank immediately preceding the hyphen in each
I* line, and the 3 blanks on the last line of the constant
I* will be included in the constant. The value of the constant
I* LONGNC will be the string:
I* THIS IS A LONG CONSTANT THAT HAS THREE BLANKS  HERE
I*
I.....Namedconstant+++++++C.....Fldnme.....
I*
I          'THIS IS A LONG -      C          LONGNC
I          'CONSTANT THAT -
I          'HAS THREE BLANKS-
I          '  HERE'
I*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I*
I* The following is an example of a continued numeric
I* constant.
I*
I.....Namedconstant+++++++C.....Fldnme.....
I*
I          123456-          C          CHAR
I          789
I*

```

Figure 117 (Part 2 of 3). The Use of Named Constants

Initialization

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I*
I* The following is an example of a hexadecimal literal.
I*
I.....Namedconstant+++++++C.....Fldnme.....
I*
I          X'C1F2C3'          C          HEX1
I*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I*
I* The following is an example of a continued hexadecimal
I* literal.
I*
I.....Namedconstant+++++++C.....Fldnme.....
I*
I          X'c1-          C          HEX2
I          'c2c3'
I*
```

Figure 117 (Part 3 of 3). The Use of Named Constants

Initialization

The initialization support provided by the RPG/400 compiler consists of three parts: the initialization subroutine, the CLEAR and RESET operation codes, and data structure initialization.

Initialization Subroutine (*INZSR)

The initialization subroutine allows you to process calculation specifications before 1P output. It is declared like any other subroutine, but with the special name *INZSR in factor 1. This subroutine will be automatically invoked at the end of the initialization step in the RPG/400 program before 1P output. You can enter any calculations that you want in this subroutine, and it can also be called explicitly by using an EXSR or CASxx operation code.

CLEAR and RESET Operation Codes

The CLEAR operation code sets a variable or all variables in a structure to blank, zero or '0' depending on the type (character, numeric or indicator). If you specify a structure (record format, data structure or array) all fields in that structure are cleared in the order which they are declared.

The RESET operation code sets a variable or all variables in a structure to their initial value. The initial value for a variable is the value it had at the end of the initialization step in the RPG/400 cycle, after the initialization subroutine has been invoked. You can use data structure initialization to assign initial values to subfields, and then change the values during the running of the program, and use the RESET operation code to set the field values back to their initial values. Because the initial value is the value the variable had after the initialization subroutine is executed, you

can use the initialization subroutine to assign initial values to a variable and then later use RESET to set the variable back to this initial value. This applies only to the initialization subroutine when it is run automatically as a part of the initialization step.

For more information on the initialization subroutine and the CLEAR and RESET operation codes see the *RPG/400 Reference*.

Data Structure Initialization

Data structure initialization allows you to initialize data structures and subfields either to blank, zero or a specific value.

By default, a data structure is considered to be a character field, and unless specified, it is initialized to blanks. However, if numeric subfields are not initialized with numeric data before they are used in arithmetic or editing operations, decimal data errors result. Data structure initialization provides a means by which data structure subfields can be initialized at compile-time, at the beginning of the *INIT step, before any other program initialization is performed.

Data structures can be initialized both globally and on a subfield basis.

A globally initialized data structure, identified by an I in column 18 of the data structure specification, is initialized with all characters set to blanks and all numerics set to zeros. Because each subfield is initialized in the order that it appears, you must ensure that overlapping fields are declared in such an order that they are initialized correctly.

A data structure initialized on a subfield basis is identified by an I in column 8 and an initialization value for the subfield in columns 21-42 of the data structure subfield specification. If columns 21-42 contain blanks, the subfield will be initialized to blanks or zeros, depending on whether the subfield is character or numeric. You can specify either a literal value or a named constant name as the initialization value in a format similar to named constants. If columns 21-42 contain a named constant or a literal, the subfield will be initialized to the initialization value specified.

A data structure can be globally initialized, and subfields individually initialized within the structure, by specifying an I in column 18 of the data structure specification and an I in column 8 of each data structure subfield specification. The subfields are initialized in the same order as they are declared in the data structure.

Special Considerations for Initializing Data Structures

You initialize a multiple-occurrence data structure by subfield value, or if you globally initialize the structure, occurrences of the structure are initialized to the same value.

The following rules apply to initializing arrays:

- If an initialization value for a run-time array is specified, each array element is initialized with the same value. To specify different values for each array element, you must use a compile-time or prerun-time array.
- Since compile and prerun-time arrays are initialized by definition, they cannot be initialized using subfield initialization support. When a compile-time or prerun-time array appears as part of a globally initialized data structure, it is not

Initialization

included as part of the global initialization. Compile-time arrays are initialized in the same order that their data is declared after the program and prerun-time arrays in the order which the array input data files are declared.

- If a subfield initialization overlaps a compile-time or prerun-time array, initialization of the array is done last, regardless of the order of the definitions.
- If a subfield and a run-time array definition overlap in a data structure, they will be initialized in the order which they are defined.

The following rules apply to initializing special data structures:

- Data area data structures, by definition, are initialized by being read in at program initialization time, so initialization support is not required for these data structures.
- Other data structures, such as the local data area and the PIP data area, can be initialized.
- Because most of the fields in file information data structures and program-status data structures are initialized by the compiler at initialization time, initialization is not supported for these structures.

Rules for Initializing Subfields

The following rules apply to initializing subfields:

- An initialization value must match the subfield's type, and may not exceed the length or number of decimal positions.
- To continue a literal over more than one line, the initialization value indicator (I in column 8) is specified only on the first line of the literal. All other rules for line continuation follow the conventions used for continuing named constants. See "Named Constants" on page 237.
- A named constant used as an initialization value can be declared either before or after the subfield where it is used. The named constant must be left-justified in columns 21-26 of the subfield specification.
- For externally described data structures:
 - An initialization value for a subfield may only be specified once. If more than one initialization value is found, the first value specified is used. All other specifications are ignored and error messages issued.
 - If the initialization specification for a renamed subfield directly follows the rename specification, the subfield name does not need to be specified on the initialization specification.
 - If a subfield is to be both renamed and initialized, you must rename the subfield before initializing it. If the initialization specification precedes the rename specification, the compiler considers the field as undefined and an error results.
- For program described subfields, if more than one initialization specification appears for a subfield, the specifications are treated as duplicate definitions of the field.

Note: Since compile-time initialization is part of the initialization step of the program, if the program ends with LR off, the subfields will not be automatically initialized during the next call to the program. The program must first be deactivated using the FREE operation.

Initialization and the Program Cycle

Figure 118 shows the order of initialization in an RPG/400 program. The initial value for a field is whatever value the field has at the point after the *INZSR is run.

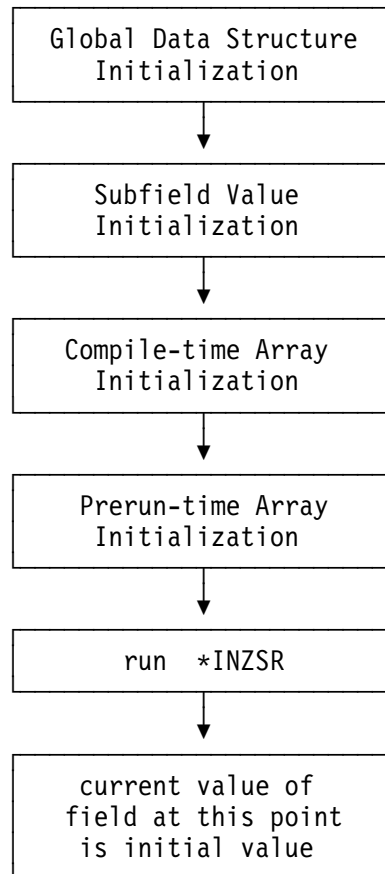


Figure 118. Order of Initialization in an RPG/400 Program.

Initialization Examples

Figure 119 on page 244 through Figure 123 on page 247 show some typical initializations of data structures.

Initialization Examples

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The I in column 18 globally initializes the data structure.
I* Numeric subfields are initialized to 0. Character subfields
I* are initialized to blanks.
I*
ID$name....NOD$Ext-file++.....OccrLen+.....*
IDS1          IDS
I.I.....Init-value+++++++PFromTo++DField+.....*
I                1   52DS1S1
I                6   10 DS1S2
I                11  15 DS1S3
I                12  162DS1S4
I*

```

Figure 119. Globally Initialized Data Structure

```

*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I*
I* In the following example, global data structure initialization
I* is specified for DS1, so the field AMOUNT will be initialized
I* to zero. AMNTCH has been initialized to '1' using subfield
I* value initialization, but because AMOUNT is declared later
I* in the data structure and overlays AMNTCH, both fields will
I* contain zero. If you wanted AMNTCH to be initialized to '1',
I* place it after AMOUNT in the data structure.
I*
I.....Namedconstant+++++++C.....Fldnme.....
I*
IDS1          IDS
I I          '1'                1   6 AMNTCH
I                1   60AMOUNT
I*

```

Figure 120. Initializing Data Structures to 0 or 1

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The data structure below is initialized by subfield. Each
I* subfield is initialized only if an I is specified in column 8 of
I* the subfield specification. Notice that the subfield DS2S2 will
I* not be explicitly initialized to a value. The subfield DS2S4 is
I* initialized to a long literal value continued over several
I* lines. Subfields DS2S5 and DS2S6 are initialized to named constant
I* character and numeric fields respectively. Subfield DS2S7 is
I* initialized to a transparent literal value.
I*
I.....Ext-field+.....PFromTo++DField+.....*
I          -1234567890.234-          C          NUM2
I          56
I          'CHAR-CONST'          C          ALPH1
I*
IDname....NODSExt-file++.....OccrLen+.....*
IDS2          DS
I.I.....Init-Value+++++++PFromTo++DField+.....*
I I          123          1  30DS2S1
I          4  5 DS2S2
I I          '5CHAR'          6  10 DS2S3
I I          'THIS IS A LONG INIT-  11  70 DS2S4
I          'VALUE CONTINUED-
I          'OVER 3 LINES'
I I          ALPH1          71  80 DS2S5
I I          NUM2          81  915DS2S6
I I          'oAABCCDDEEi-          92  118 DS2S7
I          'oFFGGHHi'
I*

```

Figure 121. Data Structure Initialized by Subfield

Initialization Examples

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The data structure DS3 is a globally initialized externally
I* described data structure. Notice that subfield initialization
I* values have been specified for the subfields shown. The subfields
I* not shown, DS3S2 and DS3S5, are not initialized to specific values
I* but will be initialized to blanks or 0. LONGEXTNM is renamed
I* to DS3S6 using a rename specification and then initialized to the
I* named constant value NUM1.
I*
I.....Ext-field+.....PFromTo++DField+.....*
I          123          C          NUM1
I          'CHAR-CONST'      C          ALPH1
I*
IDpname....NODSExt-file++.....OccrLen+.....*
IDS3      EIDS
I.I.....Init-value+++++++PFromTo++DField+.....*
I I          123          DS3S1
I I          '5CHAR'      DS3S3
I I          ALPH1      DS3S4
I          LONGEXTNM      DS3S6
I I          NUM1
```

Figure 122. Initializing an Externally Described Data Structure


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* DS4 is a globally initialized multiple occurrence data structure
E* containing an array. Here, every element of array ARR1
E* will be initialized to the string '10CHAR STR' and all
E* occurrences of the subfield DS4S2 will be initialized to 0.
E*
E...FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E*
E                ARR1          15 10

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IDname....NODSExt-file++.....OccrLen+.....*
IDS4           IDS                   5
I.I.....Init-value+++++++PFromTo++DField+.....*
I I           '10CHAR STR'         1 150 ARR1
I                151 1610DS4S2

```

Figure 123. A Globally Initialized Multiple Occurrence Data Structure

SAA Data Types

The RPG/400 compiler allows you to use SAA database data types date, time, and timestamp, and variable-length fields and null-capable fields.

Variable-Length Fields

By specifying the *VARCHAR value on the CVTOPT keyword of the CRTRPGPGM or CRTRPTPGM commands, the RPG/400 compiler will internally define variable-length fields from an externally described file or data structure as an RPG/400 fixed-length character field. When *VARCHAR is not specified, variable-length fields are ignored and inaccessible in RPG/400 programs. For more information, see the CVTOPT keyword on page 38.

The following conditions apply when *VARCHAR is specified on the CRTRPGPGM or CRTRPTPGM command:

- If a variable-length field is extracted from an externally described file or an externally described data structure, it is declared in an RPG/400 program as a fixed-length character field.
- For single-byte character fields, the length of the declared RPG/400 field is the length of the DDS field plus 2 bytes.
- For DBCS-graphic data fields, the length of the declared RPG/400 field is two times the length of the DDS field plus 2 bytes. For more information on DBCS-graphic data types, see “DBCS-Graphic Data Type Support” on page 251. The two extra bytes in the RPG/400 field contain a binary number which represents the current length (measured in double bytes) of the variable-length field. Figure 124 on page 248 shows the RPG/400 field length of variable-length fields.

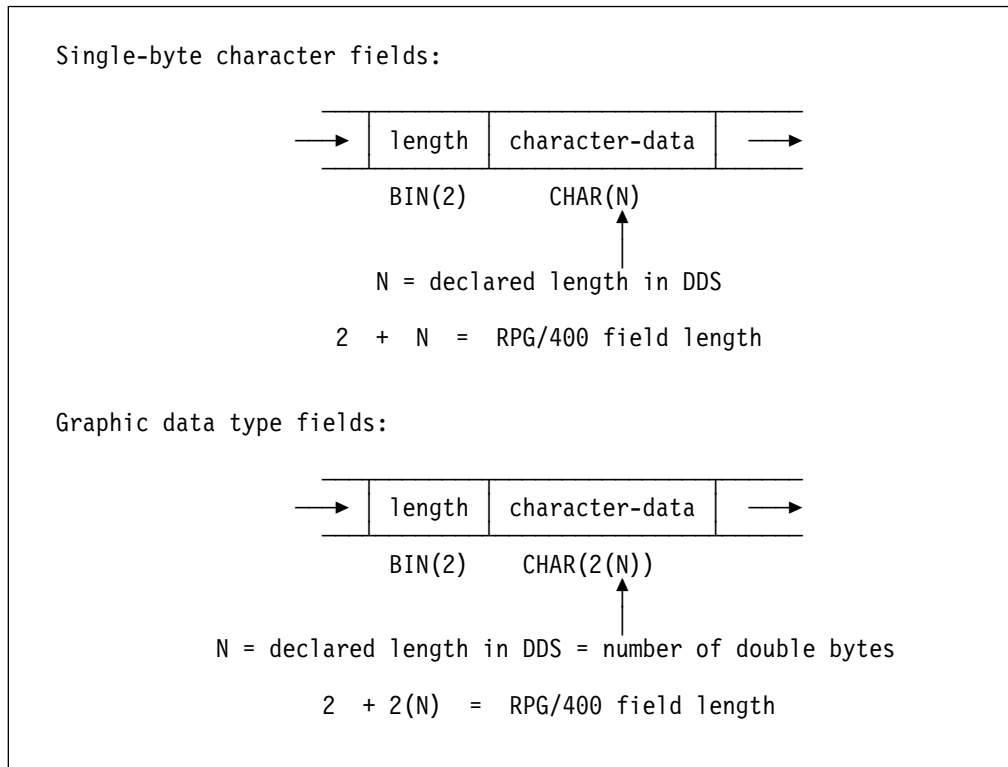


Figure 124. RPG/400 Field Length of Variable-Length Fields

- Your RPG/400 program can perform any valid character calculation operations on the declared fixed-length field. However, because of the structure of the field, the first two bytes of the field must contain valid binary data. An I/O exception error will occur for an output operation if the first two bytes of the field contain invalid field length data.
- Control-level indicators, match field entries, and field indicators are not allowed on an input specification if the input field is a variable-length field from an externally described input file.
- Sequential-within-limits processing is not allowed when a file contains variable-length key fields.
- Keyed operations are not allowed when factor 1 of a keyed operation corresponds to a variable-length key field in an externally described file.
- If you choose to selectively output certain fields in a record and the variable-length field is not specified on the output specification, or if the variable-length field is ignored in the RPG/400 program, the RPG/400 compiler will place a default value in the output buffer of the newly-added record. The default is 0 in the first two bytes and blanks in all of the remaining bytes.
- If you want to change variable-length fields, ensure that the current field length is correct. One way to do this is:
 1. Define a data structure with the variable-length field name as a subfield name.
 2. Define a 2-byte binary subfield at the beginning of the field to retrieve the current field length.
 3. Update the field.

Alternatively, you can move another variable-length field left-aligned into the field. An example of how to change a variable-length field in an RPG/400 program follows.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*
A*   File MASTER contains a variable length field
A*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++
A*
A           R REC
A           FLDVAR           100           VARLEN
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F*   Externally described file name is MASTER.
F*   Compile the RPG/400 program with CVTOPT(*VARCHAR).
F*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
F*
FMASTER UF E                               DISK
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I*   FLDVAR is a variable-length field defined in DDS with
I*   a DDS length of 100. Notice that the RPG field length
I*   is 102.
I*
I.....Ext-field+.....PFromTo++DField+L1M1FrP1MnZr...*
I*
I           DS
I                               1 102 FLDVAR
I                               B 1 20FLDLEN
I                               3 102 FLDCHR
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++
C*
C           READ MASTER           LR
C           MOVEL 'SALES'       FLDCHR
C           Z-ADD5           FLDLEN
C NLR       UPDATREC
```

Figure 125. Changing a Variable-Length Field in an RPG/400 Program

If variable-length graphic fields are required, you can code a 2-byte binary field to hold the length, and a 2(N) length subfield to hold the data portion of the field.

SAA Data Types

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I*   The variable-length graphic field VGRAPH is declared in the
I*   DDS as length 3. This means the maximum length of the field
I*   is 3 double bytes, or 6 bytes. The total length of the field,
I*   counting the length portion, is 8 bytes.
I*
I*   Compile the RPG/400 program with CVTOPT(*VARCHAR *GRAPHIC).
I*
I.....Ext-field+.....PFromTo++DField+L1M1FrP1MnZr...*
I*
IVGRAPH      DS              8
I              B    1    20VLEN
I              3    8  VDATA

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C*   Assume GRPH is a fixed length graphic field of length 2
C*   double bytes. Copy GRPH into VGRAPH and set the length of
C*   VGRAPH to 2.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C              MOVELGRPH      VDATA
C              Z-ADD2        VLEN

```

Figure 126. Using a Variable-Length Graphic Field in an RPG/400 Program

Date, Time, and Timestamp Fields

Date, time and timestamp fields are brought into your RPG/400 program only if you specify the *DATETIME value on the CVTOPT keyword of the CRTRPGPGM or CRTRPTPGM commands. If *DATETIME is not specified, date, time, and timestamp fields are ignored and inaccessible in your RPG/400 program. For a description and the syntax of CVTOPT, see the CVTOPT parameter on page 38.

Date, time or timestamp fields are brought into an RPG/400 program as fixed-length character fields. Your RPG/400 program can perform any valid character operations on the fixed-length fields.

Since date, time, and timestamp data types each have their own format, if a field containing date, time or timestamp information is output to your database file, the format and separators in the field must be exactly as required by the declared format. If the same format or separators are not used, an exception/error will occur. For more information on the formats of these database data types, see the *DDS Reference*.

If you add a record to the database file and date, time, and timestamp fields are not specified for output, the RPG/400 compiler will place default values in the fields for

output. The default value of date for any format is year 1, month 1, and day 1. The default value of time for any format is hour 0, minute 0, and second 0. The proper separators will be placed into the field according to the declared format.

Keyed operations are not allowed when factor 1 of a keyed operation corresponds to an ignored date, time, or timestamp key field.

Sequential-within-limits processing is not allowed if there is an ignored date, time or timestamp key field in the file.

DBCS-Graphic Data Type Support

The DBCS-graphic data type is a character string where each character is represented by 2 bytes. The graphic data type does not contain shift-out (SO) or shift-in (SI) characters. The difference between single byte and DBCS-graphic data is shown in the following figure:

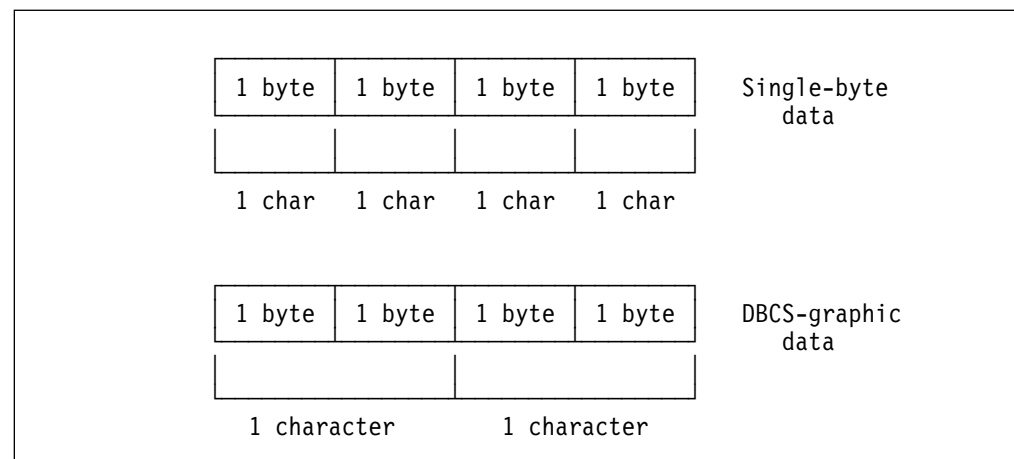


Figure 127. Comparing Single-byte and DBCS-graphic Data

DBCS-graphic data is brought into your RPG/400 program only if you specify the *GRAPHIC value on the CVT0PT keyword of the CRTRPGPGM or CRTRPTPGM commands. If *GRAPHIC is not specified, graphic data is ignored and inaccessible in your RPG/400 program. For a description and the syntax of the CVT0PT, see the CVT0PT parameter on page 38.

The following conditions apply when *GRAPHIC is specified on the CRTRPGPGM or CRTRPTPGM command:

- Graphic data is brought into an RPG/400 program as fixed-length character fields.
- Every DBCS-graphic data **character** has a length of 2 bytes.
- Every fixed-length graphic data **field** has a length of 2 bytes times the number of characters in the field. For a description of the field length of variable-length graphic data fields, see “Variable-Length Fields” on page 247.
- Your RPG/400 program can perform any valid character operations on the fixed-length fields.
- If you add a record to the database file and graphic fields are not specified for output, the RPG/400 compiler will place double-byte blanks in the fields for output. If variable-graphic fields are not specified for output, the RPG/400 com-

piler will place binary zero followed by double-byte blanks in the fields for output. The following conditions will result in blanks being placed in your output fields:

- The fields are not specified for output on the output specification.
- Conditioning indicators are not satisfied for the field.
- The required values are not specified on the CVT0PT keyword.
- Keyed operations are not allowed when factor 1 of a keyed operation corresponds to an ignored graphic field.
- Sequential-within-limits processing is not allowed if there is an ignored graphic key field in the file.

Null Value Support

Null-capable fields containing null values in a database file can be read into your RPG/400 program if you specify the *YES value on the ALWNULL keyword of the CRTRPGPGM or CRTRPTPGM commands. Currently, null value support only applies to externally described input-only files (files with no addition specified on the file specification). For more information, see the ALWNULL parameter on page 44.

When an externally described file contains null-capable fields and *NO is specified on the ALWNULL keyword, the following conditions apply:

- A record containing null values retrieved from an input or update file will cause a data mapping error and an error message will be issued.
- Data in the record is not accessible and none of the RPG/400 fields in the record can be updated with the values from the input record containing null values.
- The RPG/400 compiler is not able to place null values in null-capable fields for updating or adding a record. If you want to place null values in null-capable fields, you can use SQL/400 or other products which have full support of null values.

When an externally described input-only file contains null-capable fields and *YES is specified on the ALWNULL keyword, the following conditions apply:

- When a record is retrieved from a database file and there are some fields containing null values in the record, database default values for the null-capable fields will be placed into those fields containing null values. The default value will be the user defined DDS defaults or system defaults.
- Control-level indicators, match-field entries and field indicators are not allowed on an input specification if the input field is a null-capable field from an externally described input-only file.
- Keyed operations are not allowed when factor 1 on a keyed input calculation operation corresponds to a null-capable key field in an externally described input-only file.
- Sequential-within-limits processing is not allowed when a file contains null-capable key fields.
- Programs created with null-processing files use a run-time routine that can only be used with V2R1M1 or a later release.

Note: For a program-described file, a null value in the record always causes a data mapping error, regardless of the value specified on the `ALWNULL` keyword.

Error Handling for SAA Data Types

For any input or output operation, a data mapping error will cause a severe error message to be issued. For blocked output, if one or more of the records in the block contains data mapping errors and the file is closed before reaching the end of the block, a severe error message is issued and a system dump is created.

Chapter 11. Communicating with Objects in the System

This chapter describes how an RPG/400 program communicates with other programs in the system. The call function available in an RPG/400 program allows it to call other programs or special subroutines. The RPG/400 program also provides the return function to allow control to return from a called program.

Calling Other Programs

The RPG/400 program provides for communication with other programs.

The CALL (call a program) operation code and the RETRN (return to calling program) operation code allow an RPG/400 program to call other programs (for example, another RPG/400 program or a CL program) and to return to the calling program. The PLIST (identify a parameter list) and PARM (identify parameters) operations allow the same data to be accessed by a calling and a called program.

Figure 128 shows a conceptual view of RPG/400 programs calling other programs (RPG/400 and CL) and CL programs calling other programs (RPG/400 and CL).

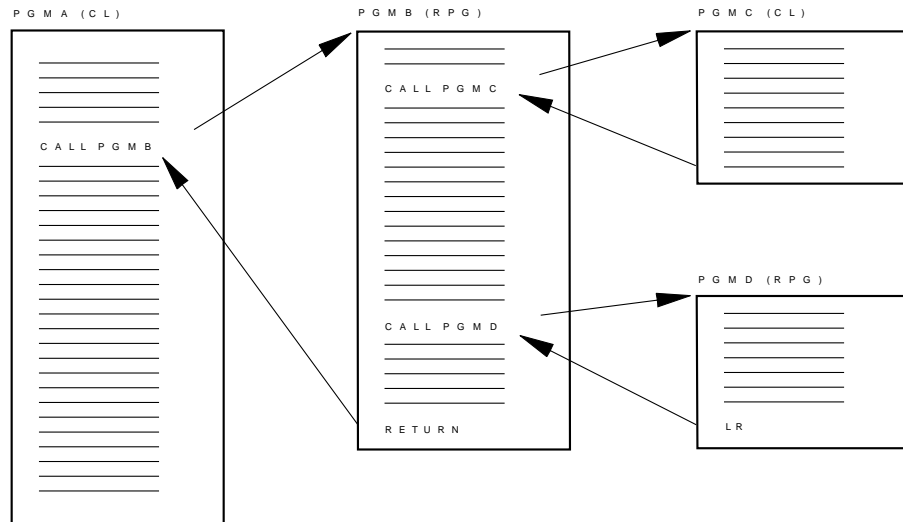


Figure 128. Calling RPG/400 programs and CL programs

See Figure 129 on page 256 for a coding example of an RPG/400 program calling another RPG/400 program using the CALL/RETRN function. See the *CL Programmer's Guide* for information about passing parameters between an RPG/400 program and a CL program.

Calling Other Programs

The CALL/RETRN function provides the following capabilities:

- PLIST and PARM(s) operation codes can be specified with the CALL operation to allow the same data to be accessed by a calling and a called program.

When an RPG/400 program is called for the first time, the program is located, the fields are set up, and the program is given control. On each succeeding call, if the called program has not ended, all fields, indicators, and files in the called program are the same as they were when the program returned on the preceding call. On each succeeding call, if the called program has ended or if FREE was specified, a fresh copy of the program is made available.

- The FREE operation code can be specified to remove a called program from the list of activated programs. If the program is called again, it functions as though it were being called for the first time. However, any files that are opened or any data areas that are locked by the called program are not affected by the FREE operation; the files or data areas are still allocated to the called program.
- The CALL operation can be dynamic; that is, the name of the program to be called can be supplied at run time.
- An explicit return is provided through the RETRN operation code.
- An implicit return is provided if the LR, RT, or H1 through H9 indicators are set on, or if the RPG/400 exception/error handling routine receives control when exception/errors occur.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*  
C*  
C* This example shows an RPG program (MAIN) using the CALL/RETRN  
C* function to call another RPG program (TRANS). The EXFMT operation  
C* in the MAIN program writes the DSPLAY record to the display screen.  
C* TOTRNS and FRTRNS are fields in the record. The work station user  
C* can key data into the TOTRNS field. The information in the TOTRNS  
C* field is to be translated by the TRANS program.  
C*  
C* Return from the TRANS program is to the statement immediately  
C* following the last PARM statement in the MAIN program. The MAIN  
C* program completes the transaction. When the GOTO operation is  
C* processed, the program branches back to the beginning of  
C* calculations. This loop continues until the work station user  
C* presses a command attention key that sets on indicator 98 to end  
C* the program. (On the DDS for the record format DSPLAY, a command  
C* attention key is associated with indicator 98.)  
C*  
C* When indicator 98 is on, the program branches to the ENDPGM TAG  
C* statement and the FREE operation frees the TRANS program. The  
C* MAIN program ends when LR is set on.  
C*
```

Figure 129 (Part 1 of 3). CALL/RETRN Function

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* MAIN LINE PROGRAM
C*
CL0N01N02N03Factor1+++OpdcFactor2+++ResultLenDHHiLoEqComments+++++*
C          BEGIN          TAG
C          EXFMTDSPLAY
C 98          GOTO ENDPGM
C          "
C          "
C          CALL 'TRANS'
C          PARM          TOTRNS
C          PARM          FRTRNS
C          "
C          "
C          "
C          GOTO BEGIN
C          ENDPGM        TAG
C          FREE 'TRANS'
C          SETON          LR

```

Calculations

Figure 129 (Part 2 of 3). CALL/RETRN Function

Calling Other Programs

```

*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C*
C* When the CALL 'TRANS' operation in the MAIN program is processed,
C* the FLDY and FLDX names in the TRANS program are used to access
C* the data in the TOTRNS and FRTRNS fields in the parameter list
C* specified in the MAIN program. Using this data, the TRANS
C* program translates the TOTRNS field, which is called FLDY in the
C* TRANS program, and places the result of the operation in the FLDX
C* field. The RETRN operation in the TRANS program is then processed.
C* (The translated field is called FRTRNS in the MAIN program.)
C* A RETRN operation without the LR indicator on is specified to
C* keep the program and all its work areas intact.
C*
C* TRANSLATE PROGRAM
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *ENTRY    PLIST
C          PARM      FLDY
C          PARM      FLDX
C          START    TAG
C          "
C          "
C          "
C          "
C          RETRN

```

Calculations

Figure 129 (Part 3 of 3). CALL/RETRN Function

CALL (Call a Program)

The CALL operation transfers control from the calling to the called program. A PLIST name is optional in the result field. If specified, it names a list of data that can be communicated between the calling program and the called program. If the called program accesses data in the calling program, and if the result field is blank, the CALL operation must be immediately followed by PARM operations.

When the CALL operation is processed, the calling program passes control to the called program. After the called program is run, control returns to the first statement that can be processed after the CALL operation in the calling program. If an error occurs during processing of the CALL operation (for example, the called program is not found), the RPG/400 exception/error handling routine receives control. See “Exception/Error Handling” on page 70 for detailed information on the RPG/400 exception/error handling routine.

You can query the names of programs called by way of a named constant or literal in an RPG/400 program using the CL command DSPPGMREF. If you call a program via a variable using the CALL operation code, you will see a program entry with the program name *VARIABLE (and no library name) to indicate that a call by variable name is in the program.

Remember the following when specifying CALL:

- A program can contain multiple CALLs to the same program with the same or different PLISTs specified.
- The first CALL to a program causes program initialization. On subsequent CALLs to the same program, program initialization is bypassed unless the FREE operation was specified or the program was ended on a previous CALL.
- The addressing of parameters is limited to data formats common to the calling and called programs.
- When a calling program ends in error or issues a return code greater than 1, the indicators in positions 56 and 57 are set on.
- An RPG/400 program cannot call itself or a program higher in the program stack. For example, if program A calls program B, program B can call neither program A nor B. If program B returns, with or without LR set on, and if program A then calls program C, program C can call program B but not program A or C.
- There are restrictions that apply when using the CALL operation code. For a detailed description of these restrictions, see the *RPG/400 Reference*.

PLIST (Identify a Parameter List) and PARM (Identify Parameters)

The PLIST and PARM operations are calculation operations that can be used with CALL. The PLIST operation:

- Defines a name by which the following list of parameters (PARMs) can be specified in a CALL operation
- Defines the entry parameter list (*ENTRY PLIST) in a called program.

Factor 1 of the PLIST statement must contain the PLIST name. This name can be specified in the result field of one or more CALL operations. If the parameter list is the entry parameter list of a called program, factor 1 must contain *ENTRY. Only one *ENTRY PLIST can be specified in a program.

The *PARMS field in the program status data structure (PSDS) can be used to determine the number of parameters passed to a program from a calling program. By using this field, references to the parameters that are not passed from the calling program can be avoided and the called program can support additional parameters without forcing recompilation or changes to the calling program.

The parameters comprising the PLIST are defined by the immediately following PARM operations. The result field of a PARM statement identifies the data that the called program can address. Connection between the calling and called program is by address; therefore, the parameters are name independent.

Rules for Specifying PLIST

Remember the following when specifying a PLIST statement:

- If PLIST is specified, it must immediately be followed by the PARMs that apply to it. If no PARM statements follow a PLIST statement, the PLIST statement is not permitted.
- Multiple PLIST statements can appear in a program.

Calling Other Programs

- Only one *ENTRY PLIST can occur in a program.
- A PLIST and its associated PARMs can appear anywhere in calculations.

Rules for Specifying PARM

Remember the following when specifying a PARM statement:

- One or more PARM statements must immediately follow a PLIST statement.
- One or more PARM statements can immediately follow a CALL operation.
- If there are more parameters in the calling program than in the called program, the called program does not run.
- If there are more parameters in the called program than in the calling program, an error occurs when an unresolved parameter is used.
- Fields specified as parameters in an *ENTRY PLIST can be used at first-page (1P) time.
- When a multiple occurrence data structure is specified in the result field of a PARM statement, all occurrences of the data structure are passed as a single field.
- The result field of a PARM statement cannot contain:
 - *IN
 - *INxx
 - *IN,xx
 - A data-area name
 - A data area data structure name
 - A label
 - A literal
 - A look-ahead field
 - A named constant
 - A table name
 - A user-date reserved word.

In addition, an array element, a data structure subfield name, the name of a compile-time array, and the name of a program status or file-information data structure (INFDS) or a data structure specified in a *NAMVAR DEFN are not allowed in the result field of PARMs specified for an *ENTRY PLIST. A field name can be specified only once in an *ENTRY PLIST.

- When parameters are passed to an RPG/400 program that is called through CL, the parameters can be specified on the command that calls the program.
- Factor 1 of a PARM statement cannot contain a literal, a look-ahead field, a named constant, or a user-date reserved word.
- Factor 1 and factor 2 must be blank if the result field contains the name of a multiple occurrence data structure.

OS/400 Graphics Support

The RPG/400 program allows you to use the CALL operation to call OS/400 Graphics, which includes the Graphical Data Display Manager (GDDM, a set of graphics primitives for drawing pictures), and Presentation Graphics Routines (a set of business charting routines). Factor 2 must contain the literal or named constant 'GDDM' (not a field name or array element).

Use the PLIST and PARM operations to pass the following parameters:

- The name of the graphics routine you want to run.
- The appropriate parameters for the specified graphics routine. These parameters must be of the data type required by the graphics routine.

The RPG/400 program does not implicitly start or end OS/400 graphics routines.

For more information on OS/400 Graphics, graphics routines and parameters, see the *GDDM Programming Guide* and the *Graphical Data Display Manager Programming*.

FREE (Deactivate a Program)

The FREE operation code:

- Removes a program from the list of activated programs
- Frees static storage if you no longer require the program
- Ensures program initialization (first cycle processing) when a program is called.

FREE neither closes files nor unlocks data areas. You are responsible for closing files and unlocking data areas in your own program. In an interactive environment, you can close files and unlock data areas by using the CL command RETURN (from level 1 of the command entry display) or SIGNOFF. (See the *CL Reference* for the use of RETURN and SIGNOFF commands.)

When the FREE operation is specified, the program named in factor 2 is released from the list of activated programs. If the program is called by the CALL operation again, it functions as though it were being called for the first time (first-cycle processing). If the FREE operation is not successful, the RPG/400 exception/error handling routine receives control. See “Exception/Error Handling” on page 70 for detailed information on the RPG/400 exception/error handling routine.

General-Use Programming Interface

Calling Special Subroutines

The three special subroutines that are available in an RPG/400 program are:

- Message-Retrieving Subroutine (SUBR23R3)
- Moving Bracketed Double-byte Data and Deleting Control Characters (SUBR40R3)
- Moving Bracketed Double-byte Data and Adding Control Characters (SUBR41R3).

Note: For detailed information on the use of CALL and PARM operation codes, see the *RPG/400 Reference*.

Message-Retrieving Subroutine (SUBR23R3)

The message-retrieving subroutine (SUBR23R3) allows you to retrieve messages from a user message member QUSERMSG. If you want to use other message files, you can use the CL command OVRMSGF to override the message file. After the message has been retrieved, it can be changed and written to an output file.

Connection to SUBR23R3 is by the CALL operation code, and input parameters are passed to SUBR23R3 by PARM operation codes. To use SUBR23R3, specify CALL in

Calling Special Subroutines

columns 28 to 32 and 'SUBR23R3' in columns 33 to 42. Five PARM operation codes must be specified after the CALL operation with the following result-field entries:

Result Field	Description																
Message Identity (MSGID)	If LEVEL = 1 or 2, name of a 4-digit numeric field that will be prefixed with 'USR' to form the message identity of the message to be retrieved. or If LEVEL = 3 or 4, name of a 7-position character field that contains the message identifier to be retrieved. The format of this field is <i>aaannnn</i> where <i>a</i> is any value from A to Z or characters #, @, or \$ and <i>n</i> is any value from 0 to 9 or A to F.																
Text area	Name of the alphanumeric field or data structure into which the message text is read. The maximum length of a level-1 message is 132 characters and of a level-2 message is 3000 characters. (Data structures must be used when the message is more than 256 characters.)																
Level	Name of a 1-digit numeric field that designates the user message member level. A value of 1 or 3 in this field indicates a message level of 1; a value of 2 or 4 indicates a message level of 2. (Data structures must be used when the message is more than 256 characters.) The value of 1 or 2 indicates the MSGID field is a 4-digit numeric field, and the value of 3 or 4 indicates the MSGID is a 7-digit alphanumeric field.																
Return Code	Name of a 1-digit numeric field that contains the return codes. The return codes and their meanings are as follows: <table border="0"> <thead> <tr> <th><i>Return Code</i></th> <th><i>Meaning</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Message was successfully retrieved with no truncation. The message may or may not contain text.</td> </tr> <tr> <td>1</td> <td>Message was successfully retrieved; but it was truncated because the length of the text area was less than the message length.</td> </tr> <tr> <td>2</td> <td>Message was not found.</td> </tr> <tr> <td>3</td> <td>Message level was incorrect. (Not 1,2,3, or 4)</td> </tr> <tr> <td>4</td> <td>An incorrect MSGID value was diagnosed. (The value was not 0000 to FFFF.)</td> </tr> <tr> <td>5</td> <td>Message file was not found, or you didn't have the right authority, or message text length exceeds the level-1 maximum length.</td> </tr> <tr> <td>6</td> <td>A not valid TXTL value was diagnosed.</td> </tr> </tbody> </table>	<i>Return Code</i>	<i>Meaning</i>	0	Message was successfully retrieved with no truncation. The message may or may not contain text.	1	Message was successfully retrieved; but it was truncated because the length of the text area was less than the message length.	2	Message was not found.	3	Message level was incorrect. (Not 1,2,3, or 4)	4	An incorrect MSGID value was diagnosed. (The value was not 0000 to FFFF.)	5	Message file was not found, or you didn't have the right authority, or message text length exceeds the level-1 maximum length.	6	A not valid TXTL value was diagnosed.
<i>Return Code</i>	<i>Meaning</i>																
0	Message was successfully retrieved with no truncation. The message may or may not contain text.																
1	Message was successfully retrieved; but it was truncated because the length of the text area was less than the message length.																
2	Message was not found.																
3	Message level was incorrect. (Not 1,2,3, or 4)																
4	An incorrect MSGID value was diagnosed. (The value was not 0000 to FFFF.)																
5	Message file was not found, or you didn't have the right authority, or message text length exceeds the level-1 maximum length.																
6	A not valid TXTL value was diagnosed.																
Text Length (TXTL)	Name of a 4-digit numeric field that contains the length of the text area defined in the calling program.																

The text area, which is specified by the second PARM operation, is blanked before each attempt to retrieve a message; therefore, for some conditions, a blank text area is returned to the user program when the return code value is 2 or greater. A total of 132 (for level-2 messages) positions in the text area are blanked unless the text area is less than 132 (3000 for level-2 messages) characters in length.

Note: You should make sure that the text area you specify does not exceed the text area provided. If the text area does exceed the area provided, unexpected results could occur as the data in your program may be overwritten when the message is retrieved.

SAA Common Programming Interface Support

Source file QIRGINC in the QRPG and QRPGP libraries contains members which hold the includes for multiple SAA Common Programming Interfaces. These includes describe the argument or parameter interfaces. The files are IBM-owned and should not be changed. If you want to tailor one or more of the includes, copy the the member or members you want to change to a source file in one of your libraries.

Note: Because the product libraries QRPG and QRPGP are added to your product library list when you compile, the library that holds your tailored includes must be explicitly defined. Otherwise, the IBM-supplied includes will be used.

If you copy includes to your library, you must refresh these copies when a new release is installed or when changes are made via a PTF. IBM will only provide maintenance to the includes which reside in the QRPG and QRPGP libraries.

Moving Bracketed Double-byte Data and Deleting Control Characters (SUBR40R3)

The SUBR40R3 move and edit routine moves the contents of one field to another field. If the S/O and S/I control characters are found as the first and last characters in the field, SUBR40R3 deletes them. SUBR40R3 is called as shown in Figure 130.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          CALL 'SUBR40R3'
C          PARM          EMPNO  10          SENDING FIELD
C          PARM          SOCSEC  8          RECEIVING FIELD
C          PARM          RETCDE  1          RETURN CODES
C          PARM          RECLEN  30         RECEIVING LEN
C*
```

Figure 130. Calling SUBR40R3

If you want the receiving field to contain all the data that is present in the sending field, you must specify a length for the receiving field that is two positions less than the length of the sending field. This allows two positions for each double-byte character (or one for each EBCDIC character) while the S/O and S/I control characters (and the two positions they occupied) are deleted. If you specify a receiving field longer than the sending field minus two positions, all the data from the sending field is moved and the receiving field is padded on the left with blanks (1-byte EBCDIC blanks). If the receiving field is shorter than the sending field minus two positions, the data being moved is truncated on the left.

Five PARM fields must be specified when SUBR40R3 is called. The first two specify the sending and receiving fields for the move. The third field is where the return codes are written to indicate the status of the move operation. The fourth and fifth fields must be loaded with the lengths of the sending and receiving fields. These

Calling Special Subroutines

are the lengths of the fields specified on the first two PARMs for the call to SUBR40R3 (in Figure 130, you would need to load the lengths of EMPNO and SOCSEC). The return code field must be defined as a 1-position alphanumeric field; the length fields must be defined as 3-position numeric fields with zero decimal positions.

For information on DBCS-graphic data (DBCS data that doesn't use the S/O (shift-out) and S/I (shift-in) control characters), see "DBCS-Graphic Data Type Support" on page 251.

SUBR40R3 produces return codes to indicate the status of the move operation. The following list contains these return codes and their meanings:

Return Code	Explanation
0	Data moved; no errors.
1	Data moved; padding occurred.
2	Data moved; truncation occurred.
3	Data moved; S/O and S/I control characters were not found.
4	Data not moved. Either an odd field length was found, a length of zero was found, the length was greater than 256, or a not valid character was found in the field length. Length specified in fourth and fifth parameters is greater than the field length of the first and second parameters respectively.

If more than one return code can be issued, only the highest return code is returned.

Moving Bracketed Double-byte Data and Adding Control Characters (SUBR41R3)

The SUBR41R3 move and edit routine moves the contents of one field into another field. If the S/O and S/I control characters are not found in the first and last positions of the field, SUBR41R3 adds them to the field when it is moved.

SUBR41R3 is called as shown in Figure 131.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*  
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*  
C          CALL 'SUBR41R3'  
C          PARM          SOCSEC  8  
C          PARM          EMPNO  10  
C          PARM          RETCDE  1  
C          PARM          SNDLEN  30  
C          PARM          RECLEN  30  
C*
```

Figure 131. Calling SUBR41R3

If you want the receiving field to contain all the data that is in the sending field, you must specify the length of the receiving field to be two positions longer than the length of the sending field (to hold the S/O and S/I control characters). If you specify a receiving field that is longer than the sending field plus two, the data is

padding on the left when it is moved into the receiving field. If the receiving field is shorter than the sending field plus two, the data is truncated on the left when it is moved. If the receiving field is specified either longer or shorter than the sending field plus two positions, the S/I control character is still placed in the correct position (the rightmost position).

Five PARM fields must be specified when SUBR41R3 is called. The first two specify the sending and receiving fields for the move. The third field is where the return codes are written to indicate the status of the move operation. The fourth and fifth fields must be loaded with the lengths of the sending and receiving fields. These are the lengths of the fields specified on the first two PARMs for the call to SUBR41R3 (in Figure 131 on page 264, you would need to load the lengths of SOCSEC and EMPNO). The return code field must be defined as a 1-position alphanumeric field; the length fields must be defined as 3-position numeric fields with zero decimal positions.

For information on DBCS-graphic data (DBCS data that doesn't use the S/O (shift-out) and S/I (shift-in) control characters), see “DBCS-Graphic Data Type Support” on page 251.

SUBR41R3 produces return codes to indicate the status of the move. The following list contains these return codes and their meanings:

Return Code	Explanation
0	Data moved; no errors.
1	Data moved; padding occurred to left of S/I control character.
2	data moved; data truncated to left of S/I control character.
3	Data moved; S/O and S/I already present.
4	Data not moved. Either odd field length found, length of zero found, length greater than 256, or not valid character found in field length. Length is specified in fourth and fifth parameters is greater than the field length of the first and second parameters respectively.

If more than one return code can be issued, only the highest return code is issued.

End of General-Use Programming Interface

Returning from a Called Program

An RPG/400 called program returns control to the calling program in one of the following ways:

- With a normal end
- With an abnormal end
- Without an end.

A description of the ways to return from a called program follows.

For a detailed description of where the LR, H1 through H9, and RT indicators, and the RETRN operation are tested in the RPG/400 program cycle, see the section on the RPG/400 program cycle in the *RPG/400 Reference*.

A Normal End

A program ends normally and control returns to the calling program when the LR indicator is on and the H1 through H9 indicators are not on. (For further information on the LR indicator, see the *RPG/400 Reference*.) The LR indicator can be set on by:

- The last record processed from a primary or secondary file during the RPG/400 program cycle
- The programmer.

A program also ends normally if:

- The RETRN operation is processed, the H1 through H9 indicators are not on, and the LR indicator is on
- The RT indicator is on, the H1 through H9 indicators are not on, and the LR indicator is on.

When a program ends normally, the following occurs:

- Parameters are moved from factor 2 to the result field.
- All arrays and tables with a 'To file name' specified on the extension specifications, and all locked data area data structures are written out.
- Any data areas locked by the program are unlocked.
- All files that are open are closed.
- A return code is set to indicate to the calling program that the program has ended normally, and control then returns to the calling program.

On the next call to the program, a fresh copy is available for processing.

An Abnormal End

A program ends abnormally and control returns to the calling program when one of the following occurs:

- An H1 through H9 indicator is on, and the cancel option is taken when a message is issued.
- The cancel option is taken when an RPG/400 error message is issued.
- An *CANCL ENDSR statement in an *PSSR or INFSR subroutine is processed (for further information on the *CANCL return point for the *PSSR and INFSR subroutines, see "Exception/Error Handling" on page 70).
- An H1 through H9 indicator is on when a RETRN operation is processed.
- An H1 through H9 indicator is on when last record (LR) processing occurs in the RPG/400 cycle.

When a program ends abnormally, the following occurs:

- All files that are open are closed.
- Any data areas locked by the program are unlocked.
- An error return code in the program status data structure is set to indicate to the calling program that the called program has ended abnormally.
- Escape message RPG9001 is issued, and control returns to the calling program.

On the next call to the program, a fresh copy is available for processing. (For more information on the program status data structure, see “Exception/Error Handling” on page 70.)

Return without an End

A program can return control to the calling program without ending when either the RETRN operation is processed or the RT indicator is set on, and the LR or H1 through H9 indicators are not on. The RETRN operation causes control to return immediately to the calling program. The RT indicator causes control to return to the calling program after the H1 through H9 indicators and the LR indicator are tested. (For further information on the RT indicator, see the *RPG/400 Reference*.)

A program also returns without ending when something outside the program ends its activation. For example:

- RPG/400 program A calls another program (such as a CL program) that issues an escape message directly to the program calling A.
- A COBOL program calls an RPG/400 program that calls another COBOL program that ends using a STOP RUN. STOP RUN ends the COBOL run unit, which includes the RPG/400 program.

If you call a program and it is returned without an end, when you call the program again, all fields, indicators, and files in the program will hold the same values they did when you left the program, unless another program is called first.

You can use either the RETRN operation code or the RT indicator in conjunction with the LR indicator and the H1 through H9 indicators. Be aware of the testing sequence in the RPG/400 program cycle for the RETRN operation, the RT indicator, the LR indicator, and the H1 through L9 indicators.

Data Areas

A data area is an object used to communicate data such as variable values between programs within a job and between jobs. A data area can be created and declared to a program before it is used in that program or job. For information on how to create and declare a data area, see the *CL Programmer's Guide*. An RPG/400 program does not support data areas defined by the CL command CRTDTAARA in which *LGL is specified as the TYPE parameter. In addition, data areas with type *DEC are not supported for data area data structures. The *NAMVAR DEFN statement can be used to access a data area with type *DEC. The library that contains the data area must be specified in the library list.

The RPG/400 program provides access to a data area through a data area data structure, the data-area operations IN and OUT, or a combination of the two. For information on how to specify a data area data structure, see “Data Structures” on page 220.

For a data area data structure, if the data area exists in a library that is specified in the library list, the data area is copied into the program. If the data area does not exist in a library that is specified in the library list, the name and length of the data structure are used to generate a data area in the job's temporary library (QTEMP).

The RPG/400 program retrieves and locks the contents of a data area at program initialization when a data area data structure is defined in the program. At the end of program, the RPG/400 program writes the data area data structure to the data area from which it came (temporary or permanent library) and unlocks the data area data structure. If a data area data structure is unlocked at the time the RPG/400 program does the update, the RPG/400 program does not write it at the end of program. At the end of job, the job's temporary library, QTEMP, is deleted.

The IN and OUT operations retrieve and write a data area. The lock capability is optional with these operations.

If the program calls another program that uses the same data area that the calling program uses, you must unlock the data area (with the UNLCK operation) before the other program is called. Two programs cannot simultaneously use the same data area for output.

A data area can be locked only once. An RPG/400 program cannot retrieve and lock a data area that has already been locked. Programs that attempt to retrieve and lock data areas include:

- Programs that use a data area as a data area data structure
- Programs that use an IN operation with *LOCK specified in factor 1.

The currently running program cannot lock the data area if:

- The data area was used in a CL command ALCOBJ in the same or another routing step
- The data area was locked by a program that calls the current program.

To access a data area that has been locked with read allowed, retrieve it by using an IN operation with blanks in factor 1. In this case, the program can retrieve the data area but cannot change the data area by using the OUT operation.

The RPG/400 program uses the following lock states:

- An IN operation with *LOCK specified has an exclusive-allow-read (*EXCLRD) lock state.
- An IN operation without *LOCK specified has a shared-for-read (*SHRRD) lock state while transferring data. When the transfer is complete, the RPG/400 program releases the lock state.
- An OUT operation has an exclusive (*EXCL) lock state during the transfer of data and then that lock state is released. The RPG/400 program then releases the exclusive-allow-read (*EXCLRD) lock state established by the IN operation.

Another program's lock state on a data area may interfere with the operation of some or all of the RPG/400 program's lock states. See the discussion of allocating resources in the *CL Programmer's Guide* for further information on the compatibility among locks.

Figure 132 on page 269 shows a data area data structure and the IN and OUT operations (within the same program) accessing the same data area.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* In this example, the data area data structure, TOTALS, is
I* implicitly read and written by RPG and explicitly updated. At
I* program initialization, RPG implicitly retrieves and locks TOTALS
I* After calculations are performed on TOTALS, the *LOCK OUT state-
I* ment updates and maintains the lock status of TOTALS. At end of
I* program, RPG implicitly writes and unlocks TOTALS.
I* TOTAL PROGRAM
I*
IDpname....NODsExt-file++.....OccrLen+.....*
ITOTALS      UDS
I.....Ext-field+.....PFromTo++DField+.....*
I              1      82TOTAMT
I              9     182TOTGRS
I             19     282TOTNET
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C
C
C
C
C      ADD  AMOUNT  TOTAMT
C      ADD  GROSS   TOTGRS
C      ADD  NET     TOTNET
C      *LOCK OUT  TOTALS
C
C
C
C      ADD  AMOUNT  TOTAMT
C      ADD  GROSS   TOTGRS
C      ADD  NET     TOTNET
C
C
C
C      *NAMVAR  DEFN      TOTALS

```

Calculations

Calculations

Calculations

Figure 132. Data Area Data Structure and Data-Area Operations

Program Initialization Parameters (PIP) Data Area

If the RPG/400 program is a pre-started job that is to receive program initialization parameter (PIP) data, the PIP Data Area (PDA) can be used to retrieve the data. To define the PDA, use the *NAMVAR DEFN operation code, and after acquiring the requesting program device, issue an IN operation code with factor 2 specifying the name of the PDA you defined in the DEFN operation code.

Unlike other data areas, you cannot LOCK, UNLOCK, or write data to a PDA using the OUT operation code. For more information on how to define PDAs see the *RPG/400 Reference*.

For more information on pre-started jobs, see the *ICF Programmer's Guide*.

Program Initialization Parameters Data Area

Chapter 12. Auto Report Feature

This chapter contains information on the RPG/400 automatic report function. It is a program that operates before the RPG/400 compiler. Automatic report on the AS/400 system is for conversion of existing automatic report programs. The use of automatic report with AS/400 RPG/400 enhancements such as externally described files or the DEFN operation code may supply undesirable results.

Group Printing

In group printing, data is summarized for a group of input records and only totals are printed on the report. Totals can have subtotals with a final total or only a final total.

Specifications

To specify group printing using automatic report, enter a T in position 15 and *AUTO in positions 32 through 36. A control-level indicator can be specified in positions 23 through 31. When a T-*AUTO specification is used, a line is not printed for each individual record that is read, but only after a complete control group is read.

Fields and constants defined by field description specifications that have a blank or B in position 39 and follow a T-*AUTO record description are printed on the lowest level total line. Fields defined with an A in position 39 are not printed on the total lines, but the total fields created by automatic report are. Generated calculations are printed on their associated total lines. Continued column headings (C in position 39) and total-indicated fields (1 through 9 or R in position 39) can also be specified by field descriptions following a T-*AUTO record description.

Output indicators can be entered in positions 23 through 31 of a field description specification following a T-*AUTO record description if position 39 of the field-description specifications contains a blank or a B. If output indicators are used in a field description that has an A in position 39 following a T-*AUTO specification, those indicators are ignored by automatic report. Output indicators cannot be used in a field description that contains C, 1 through 9, or R in position 39.

Examples

Figure 133 on page 272 shows the file description and input specifications for the group printed reports shown in Figure 135 on page 274 and Figure 137 on page 275. BRANCH and REGION are defined as control fields.

Group Printing

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF.....RlenLK1AIOvKlocEDevice+.....KExit++Entry+A....U1.*
FSALES  IP  F      43          DISK
FPRINT  0  F      120        PRINTER
FDISKSUM 0  F      25          DISK
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES  AA  01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7  ITEMNO
I          8  9  BRANCHL1
I         10 10  REGIONL2
I         11 25  DESC
I         26 27 0SOLDQY
I         28 34 2SOLDVA
I         35 36 0ONHAND
I         37 43 2VALUE
I*

```

<p>L1 and L2 are the defined control levels.</p>
--

Figure 133. File Description and Input Specifications for Group Printed Reports

A summary file, DISKSUM, is also produced by this program. The summary file contains a summary record of the sales data for each branch. The output specifications for DISKSUM illustrate the use of standard RPG/400 output specifications in the same program with *AUTO specifications. The output record described is written on the file, DISKSUM, when there is an L1 control break (BRANCH field changes). Because the T-*AUTO specification is conditioned by L2, automatic report does not generate fields for the L1 control level. Therefore, standard RPG/400 calculation specifications must be used to calculate the L1 totals. The L1 total fields that are written on the DISKSUM file (SOLDQ1, SOLDV1, and VALUE1) must be defined in the calculations.

Figure 134 shows the output specifications and the group printed report showing sales totals for a company. Because the T-*AUTO specification is conditioned by L2, only the totals for REGION (L2) and for the entire company (LR) are printed on the report. The totals for BRANCH (L1) are not printed.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C 01      SOLDQ1   ADD  SOLDQY   SOLDQ1  40
C 01      SOLDV1   ADD  SOLDVA   SOLDV1  92
C 01      VALUE1   ADD  VALUE    VALUE   92
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT  H                *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++*
0
0                'SALES FOR ANY COMPANY'
0                ' BY BRANCH AND REGION'
0*
0   1 T   2 L2      *AUTO
0                BRANCH      'BRANCH'
0                SOLDQY A     'NUMBER OF SALES'
0                SOLDVA A     'VALUE'
0                VALUE  A     'VALUE OF STOCK'
0                C           ' ON HAND'
0                R           'REGION'
0                REGION 2
0                2           'TOTALS'
0                R           'COMPANY TOTAL'
0
ODISKSUM T          L1
0                REGION    1
0                BRANCH    3
0                SOLDQ1 B   7
0                SOLDV1 B  16
0                VALUE1 B  25
0*
```

Figure 134. Using *AUTO to Produce a Group Printed Report Showing Data Structure to Accumulate Totals—Example 2

- 1** T in position 15 with *AUTO in positions 32 through 37 specifies a group printed report.
- 2** Because L2 is entered under output indicators, total lines are printed only for L2 and LR, although L1 is also a defined control level. In group printing, the lowest level total lines printed (L2, in this case) are single-spaced, like detail lines.

Group Printing

11/11/87	SALES FOR ANY COMPANY BY REGION			PAGE	1
REGION	NUMBER OF SALES	VALUE	VALUE OF STOCK ON HAND		
1	23	71,000.00	19,000.00	*	
3	30	70,000.00	29,000.00	*	
COMPANY TOTAL	53	141,000.00	48,000.00	**	

Figure 135. Group Printed Report Showing Region and Final Totals

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
O
O 'SALES FOR ANY COMPANY'
O ' BY BRANCH AND REGION'
O*
O T *AUTO
O BRANCH 'BRANCH'
O SOLDQY A 'NUMBER OF SALES'
O SOLDVA A 'VALUE'
O VALUE A 'VALUE OF STOCK'
O C ' ON HAND'
O 2 'REGION'
O REGION 2
O 2 'TOTALS'
O R 'COMPANY TOTALS'
O*

```

Figure 136. Using *AUTO to Produce a Group Printed Report Showing Region, Data Structure to Accumulate Totals—Example 2

When no control-level indicators are entered under output indicators, a total line is generated for each defined control-level indicator (L1 and L2, in this case) and for LR.

Figure 137 shows a group printed report similar to the one shown in Figure 135. However, the T-*AUTO specifications are not conditioned by a control-level indicator, so totals are printed for all defined control levels and for LR.

SALES FOR ANY COMPANY BY BRANCH AND REGION					PAGE	1
BRANCH	NUMBER OF SALES	VALUE	VALUE OF STOCK ON HAND			
L1 → 17	17	53,000.00	12,000.00	*		
L1 → 22	6	18,000.00	7,000.00	*		
L2 → REGION 1 TOTALS	23	71,000.00	19,000.00	**		
L1 → 25	30	70,000.00	29,000.00	*		
L2 → REGION 3 TOTALS	30	70,000.00	29,000.00	**		
LR → COMPANY TOTALS	55	141,000.00	48,000.00	***		

Figure 137. Group Printed Report Showing Region, Branch, and Final Totals

/COPY Statement Specifications

The automatic report copy function provides a way to include RPG/400 source specifications from a source-file member in an RPG/400 program. Use the copy function to include source specifications that are identical or nearly identical in several different programs, thereby reducing the need to repeatedly code specifications that are used in several programs. For example, if file description and input specifications for a particular file are similar in different programs, these specifications can be placed in a source-file member and included in any program by the copy function.

Automatic report specifications and any valid RPG/400 specifications, including arrays and tables can be copied in this manner. When compile-time arrays or tables are contained in /COPY members, you must ensure that the data is encountered in the same order as the extension specifications which declare the arrays or tables. The automatic report option specifications and other copy statements cannot be copied. See “Examples of Using Automatic Report” on page 299 for an example of using the copy function.

The specifications included in an automatic report program by the copy function are initially placed in the program immediately following the /COPY statement. When all specifications are copied from the source-file member, the entire automatic report program is sorted into the order required by the RPG/400 compiler.

To request the copy function, use the /COPY statement. This statement identifies the source-file member containing the RPG/400 specifications to be included in the source program generated by automatic report. /COPY statements must follow the automatic report option specifications, and they must precede source arrays and tables (file translation tables, alternative collating sequence tables, and compile-time arrays and tables).

/COPY Statement Specifications

The file name specified on a /COPY statement must not be changed by a control language override command. No inline data file can be specified as the file on a /COPY statement.

The automatic report /COPY specification is similar in syntax to the compiler /COPY directive.

The format of the /COPY statement is:

Position	Entry
1-5	Page and line number indicating the placement of the statement in the sequence of automatic report source specifications.
6	This position can contain any entry except H or U, or can be blank.
7-11	Enter the characters /COPY.
12	Blank.
13-44	Enter the qualified file name (library-name/file-name on the AS/400 system, or file-name.library-name in the System/38 environment), followed by a comma, followed by the member name. If the library name is not specified, the library list (*LIBL) is used to locate the file. If F1, F2, R1, or R2 is specified as the file name, the file name QRPGRSRC is assumed, and the library list is used to locate the file. If only one entry appears, it is the member name; the file name QRPGRSRC is assumed and the library list is used to locate the file. The member name must exist in QRPGRSRC.
45-49	Blank.
50-80	Enter any information or comments. The contents of these positions are not read by automatic report.

Figure 138 shows an example of the /COPY statement.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
I/COPY QGPL/SALES,SALETR  
I*  
I*  
I*   qualified   member  
I*   name       name
```

Figure 138. Example of the /COPY Auto Report Statement

Note: It is convenient to code the /COPY statement on the input specifications if input specifications are to be changed as they are copied.

Changing Copied Specifications

Statements can be included in the automatic report specifications to change file description and input-field specifications as they are copied from the source-file member. No other types of specification can be change. /COPY modifier statements from the source program that add, change, or delete entries on input field specifications are identified by an X in print position 6 of the automatic report listing.

Changing File Description Specifications

To change a file description specification that is copied from a source-file member, enter the file name in positions 7 through 14 of a file description specification (F in position 6). Then make only those entries on the line that are to replace existing entries in the copied specification or that are to be included as new entries. Blank entries in the modifier statement do not affect the copied statement.

For example, the file description specifications for a frequently used file named SALES is to be copied from a source-file member. The original specification contains an I in file type (position 15), defining SALES as an input file. (See Figure 139 on page 278.) To update the sales file, change position 15 to a U by including a modifier file description specification in the automatic report source program. The modifier statement must contain the file name, SALES, and the new file type entry, U. As a result of the modifier statement, the file type on the copied file description specification is changed from I to U.

To set an entry to blanks, enter an ampersand (&) in the first position of that entry of the modifier statement, and leave the remaining positions blank. For example, to remove the overflow indicator (positions 33 and 34) from the specification shown in Figure 139 on page 278, add an ampersand to the modifier statement in position 33, as shown in Figure 140 on page 278, and leave position 34 blank.

Modifier statements for file description specifications do not have to be in any particular order in the automatic report source program, except that they cannot immediately follow the /COPY statement if input field specifications are also being changed.

No modifications are allowed to the file description continuation specifications that accompany a copied file description. To add new continuation specifications, place them after a file description modifier statement for the file. A maximum of five continuation specifications are allowed to follow a file description specification (combined total of original and added continuation specifications).

Changing Input-Field Specifications

Only input-field specifications (those describing individual fields on the input record) can be changed. To change a copied input field specification, enter the field name in positions 53 through 58 of an input-field modifier statement (I in position 6). Modifier statements for input-field specifications must immediately follow the /COPY statement in the automatic report program that copies those specifications. The first specification following the /COPY statement that is not an input-field specification is considered the end of the input-field modifier statements for the /COPY statement. (A comment statement with an I in position 6 is not considered the end of the input field modifier statements.)

/COPY Statement Specifications

The fields that can be changed are:

- Position 43 (packed/binary)
- Positions 44-51 (field location)
- Position 52 (decimal positions)
- Positions 59-60 (control levels)
- Positions 61-62 (matching fields)
- Positions 63-64 (field record relationship)
- Positions 65-70 (field indicators).

The method of replacing, adding, or blanking entries is similar to the method used to change file description specifications. To replace or add entries, code the new entry in the proper location in the modifier statement; to set an entry to blank, place an ampersand (&) in the first position of that entry in the modifier statement.

Figure 139 shows an example of changing a copied file description specification.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* /COPY statement to copy specifications for SALES file from the
I* library QGPL. The member name is SALETR.
I/COPY QGPL/SALES,SALETR
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* File description specification as it is stored in the source-file
F* member.
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES IP F 43 OF DISK
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* Copy function modifier statement.
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES U
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* Resulting file description specification that is included in the
F* RPG/400 source program.
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES UP F 43 OF DISK
```

Figure 139. Changing a Copied File Description Specification

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES &
F*
```

Figure 140. Setting a Copied File Description Entry to Blank

The modifier statement changes all copied input-field specifications that have the same field name. If there is no input field by the same name, the modifier statement is added to the program as a new input-field specification. Modifier statements with duplicate field names are allowed (length and number of decimal positions must also be the same), but only the first is used to change a copied specification. Other field names are added as new input-field specifications. Up to 20 input-field modifier statements are allowed per /COPY statement.

For best results, first place those statements that change existing input field specifications; then place those that are to be added as new input-field specifications. This procedure is suggested because input field modifier statements that do not fit into the special main storage table for modifier statements are added to the RPG/400 source program as new input-field specifications. This order of specifying modifier statements increases the likelihood that excess statements, if any, will be valid field descriptions. Figure 141 shows examples of changing input specifications.

Input specifications as stored in a source file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES  AA  01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7  ITEMNO
I          8  9  BRANCH
I         10 10  REGION
I         11 25  DESC
I         26 27  SOLDQY
I         28 34 2SOLDVA          13
I         35 36 0ONHAND
I         37 43 2VALUE
I*
```

Figure 141 (Part 1 of 2). Changing Copied Input-Field Specifications

/COPY statement and modifier statements:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I/COPY SALETR
I          BRANCHL1 1
I          SOLDVA          & 2
I         1  43  RECORD  3
I*
```

Figure 141 (Part 2 of 2). Changing Copied Input-Field Specifications

- 1** Add an entry to BRANCH field description.
- 2** Blank out minus field indicator on SOLDVA description.
- 3** Add a new field description.

Resulting input specifications for SALES file showing:

Report Format

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES  AA  01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7 ITEMNO
I          8  9 BRANCHL1 1
I         10 10 REGION
I         11 25 DESC
I         26 270SOLDQY
I         28 342SOLDVA 2
I         35 360ONHAND
I         37 432VALUE
I          1 43 RECORD 3
```

Figure 142. Changing Copied Input-Field Specifications

- 1** Added L1 indicator.
- 2** Blanks in place of minus field indicator (positions 67 and 68).
- 3** Added field description.

Report Format

One of the advantages of automatic report is that it frees the programmer from the task of specifying the format of his report on the output specifications form. Automatic report can completely format the report by spacing, skipping, centering lines, and calculating end positions for fields and constants.

Spacing and Skipping

Spacing and skipping can be either left to automatic report or specified by you. Figure 143 on page 282 shows spacing and skipping generated by automatic report. For the specifications used to produce the report, see "Generated Specifications" on page 284. If positions 17 through 22 are blank on an H-*AUTO specification, a skip to line 06 is done before the first heading line is printed and space-two-after is done for the last heading line. If more than one heading line is specified, space-one-after is done for the first and all succeeding lines except the last. To specify spacing and skipping, follow the standard RPG/400 rules for spacing and skipping.

Column heading lines are spaced like page headings. Space-one-after is done for all except the last. Space-two-after is done for a single heading line, or for the last heading line if more than one is specified. Spacing and skipping entries cannot be specified for column headings. If spacing and skipping entries are made on a D-*AUTO record description specification, the entries apply to the detail line generated. The entries do not apply to column headings or total lines generated by automatic report from the D-*AUTO specification. Standard RPG/400 rules for spacing and skipping must be followed. Space-one-after is assumed for the generated detail line if spacing and skipping entries are not made.

Space-two-after is generated for all total lines produced by automatic report from a D-*AUTO specification. In addition, the lowest level total line and the final total line are also generated with a space-one-before.

If spacing and skipping entries are made on a T-*AUTO specification, the entries apply to the lowest level total line generated, but not to column headings or higher-level total lines. If spacing and skipping are not made, the lowest level total lines are generated with space-one-after; all higher levels are generated with space-two-after. Space-one-before is always generated for the second-to-the-lowest level total and the final total (see Figure 136 on page 274 for an example).

Placement of Headings and Fields

Automatic report generates end positions for fields and constants and centers column headings, columns, and report lines. (See Figure 143 on page 282 for an example.) If an end position is specified for a field or constant on a D/T-*AUTO field description line, that end position is used on all column heading, detail, and total specifications generated from the field description. (The specified end position may be altered slightly by automatic report when the line is centered or when the column heading and field are positioned relative to each other.) If the specified end position causes an overlay with a previous field or constant, automatic report generates a new end position.

Specify end positions only to eliminate the automatic spacing between fields or to spread out or expand a report on the page.

Page Headings

If the date and page number are printed on the first *AUTO page heading line (that is, if they are not suppressed by an N in position 27 of the option specifications or by the *NODATE option of the RPTOPT parameter in the CRTRPTPGM command), the date is always printed in positions 1 through 8. The page number is printed with an end position equal to the highest end position of the longest line in the report. When the first *AUTO page heading (including date, title, and page number) is the longest line in the report, one blank space separates the title from the date and the word PAGE from the title. If the resulting line exceeds the record length of the printer file, the excess information on the right of the line is not printed.

If a line generated from a D/T-*AUTO specification is the longest report line, that line is printed starting in print position 1, and the title portion of the first page heading line is centered relative to that line.

Additional *AUTO page headings are then centered on the first *AUTO page heading line.

If an *AUTO page heading is the longest line in the report and a D/T-*AUTO specification is present, any other *AUTO page heading lines and the line generated from the D/T-*AUTO specification are centered on the longest page heading.

Fields and constants appear in the order specified in the *AUTO output specifications from left to right. Automatic report provides one blank space before and after fields on the heading line. No spacing is provided between constants.

Reformatting *AUTO Page Headings

You can reformat an *AUTO page heading line if you do not want to use the end positions for fields and constants that are generated by automatic report. If you want to find what end positions are generated for page, date, and title information, see the listing of the generated source program that is produced by the RPG/400 compiler. See "Generated Specifications" on page 284.

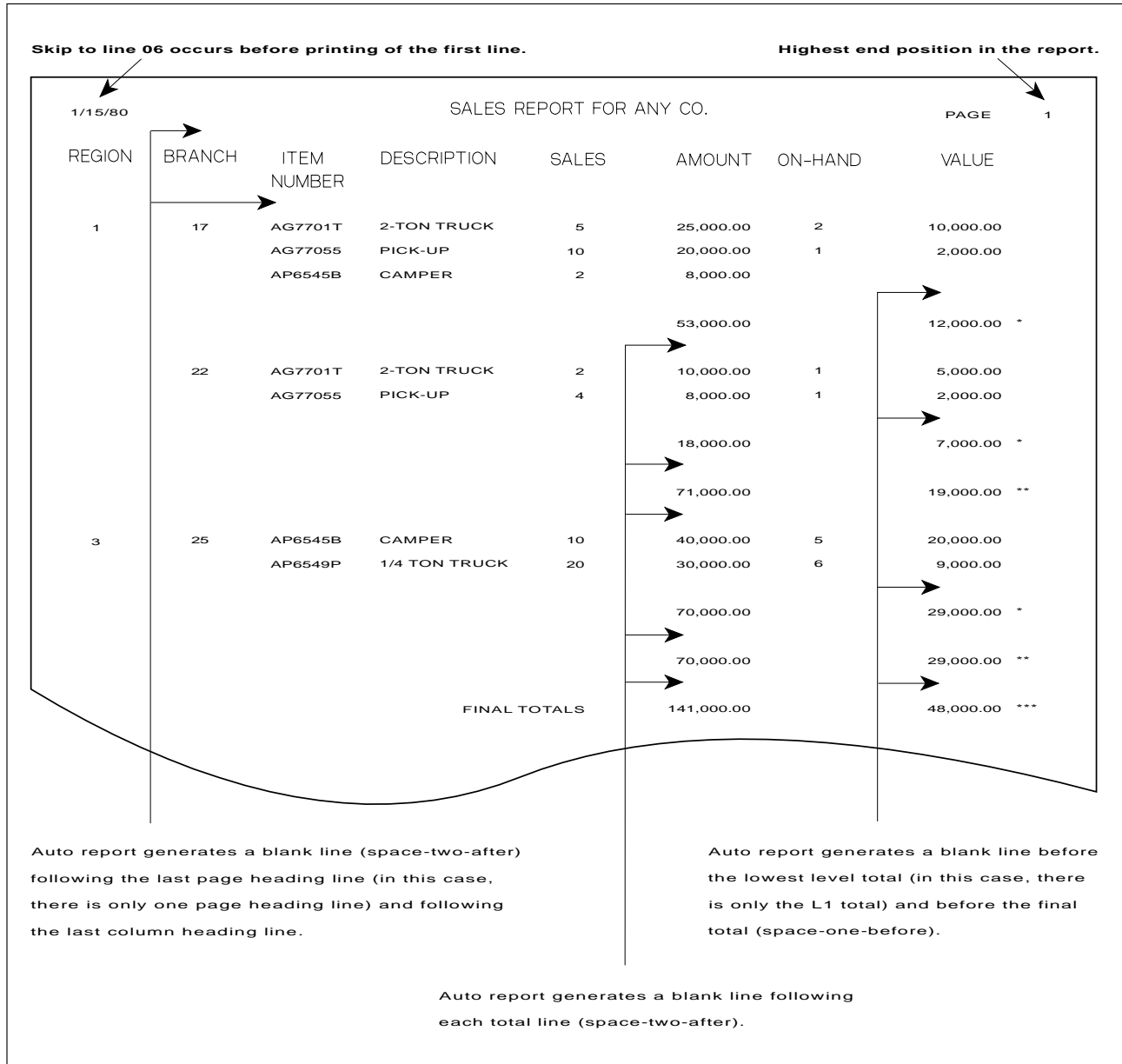


Figure 143. Report Illustrating Format Generated by Automatic report

Body of the Report

Placement of column headings above columns depends on which is longer, the heading or the associated field (including edit characters). If any column heading is longer than the associated field, the field is centered under the longest column heading constant. If, however, the field is longer than the longest-column heading constant, the column heading is left-adjusted over a character field and right-adjusted over a numeric field. When more than one column heading line is specified, shorter column headings are always centered on the longest column heading.

Fields and constants appear from left to right on a line in the order they are specified by the output specifications. At least two blank spaces appear before each field on the line. No spaces are provided before a constant; the programmer must incorporate blanks within constants to provide for additional spacing.

Total indication information (fields and constants specified with 1 through 9 or R in position 39) is placed to the left of the first total field (A in position 39) on the corresponding total line, followed by two spaces. If two or more such fields or constants are specified for a total line, they appear from left to right in the order specified on the left of the first total on the line. Each field is preceded and followed by one space. No spacing is provided for constants.

Overflow of the D/T-*AUTO Print Lines

If the lines generated from a D/T-*AUTO specification are longer than the record length specified for the printer file, a second print line (overflow line) is generated for each column heading line, detail (or group print) line, and total line. (Remember, a second print line is not generated for *AUTO page heading lines.) The excess information is placed on the overflow line in the order specified, right-adjusted.

Figure 144 on page 284 shows the result of an overflow condition.

In the output specifications for the report shown in Figure 144 on page 284, no spacing or skipping is specified. If spacing and skipping were specified, however, automatic report spaces the report as follows:

- Column heading lines and total lines are spaced as shown in Figure 144 on page 284.
- The space-before and skip-before entries specified are for the original detail (or group print) line. Automatic report generates space-one-after for this line.
- The space-after and skip-after entries specified are for the overflow line. Automatic report generates blanks for space-before and skip-before for the overflow line.

Generated Specifications

Auto report prints those columns that cannot be completely contained on the original line on overflow lines.

8 / 15 / 80		CASH RECEIPTS REGISTER					PAGE 1	
REGION	ACCOUNT NUMBER	ACCOUNT NAME	INVOICE NUMBER	INVOICE DATE	DATE PAID	AMOUNT OWED	DISCOUNT TAKEN	
					AMOUNT PAID	BALANCE DUE	EXCESS DISCOUNT	
1	11243	JONES HARDWARE	27541	7 / 11 / 80	7 / 21 / 80	23.75	.47	
					23.28			
1	11352	NU-STYLE CLOTHIERS	27987	7 / 14 / 80	7 / 26 / 80	87.07		
					40.00	47.07		
1	11886	MIDI FASHIONS INC	15771	7 / 04 / 80	7 / 14 / 80	107.22	2.14	
					105.08			
1	12874	ULOOK INTERIORS	25622	7 / 09 / 80	7 / 23 / 80	67.95		
					67.95			
1	18274	STREAMLINE PAPER INC	29703	7 / 21 / 80	7 / 30 / 80	274.03	2.38	
					170.55	101.10		
					REGION TOTALS	560.02	4.99	
					406.86	148.17		*
2	23347	RITE-BEST PENS CO	20842	7 / 18 / 80	7 / 20 / 80	15.80		
					10.00	5.80		
2	25521	IMPORTS OF NM	29273	7 / 20 / 80	7 / 27 / 80	797.40	11.93	
					585.47	200.00		
2	26723	ALRIGHT CLEANERS	19473	7 / 07 / 80	7 / 23 / 80	462.00		
					462.00			
2	28622	NORTH CENTRAL SUPPLY	17816	7 / 05 / 80	7 / 22 / 80	75.97		
					75.97			
2	29871	FERGUSON DEALERS	27229	7 / 10 / 80	7 / 22 / 80	61.91		
					61.91			
					REGION TOTALS	1,413.08	11.93	
					1,195.35	205.80		*
3	30755	FASTWAY AIRLINES	26158	7 / 06 / 80	7 / 19 / 80	742.72	16.85	
					725.87		1.90	
3	31275	ENVIRONMENT CONCERNS	20451	7 / 06 / 80	7 / 30 / 80	29.43		
					15.00	14.43		
3	32457	B SOLE SILOS	27425	7 / 10 / 80	7 / 20 / 80	110.05		
					110.05			
3	37945	HOFFTA BREAKS INC.	18276	7 / 06 / 80	7 / 23 / 80	47.23		
					47.23			
					REGION TOTALS	929.43	16.85	
					898.15	14.43	1.90	*
4	42622	EASTLAKE GRAVEL CO	16429	7 / 05 / 80	7 / 23 / 80	29.37		
					29.37			
					REGION TOTALS	29.37		
					29.37			*
					COMPANY TOTALS	2,931.90	33.77	
					2,529.73	368.40	1.90	**

Figure 144. Report Illustrating Overflow of D-AUTO Print Lines

Generated Specifications

Standard RPG/400 specifications are generated by automatic report and are combined with RPG/400 specifications included in the input to automatic report and specifications copied from the source-file member to produce the final RPG/400 source program. This section describes the generated RPG/400 specifications and the order of those specifications in the RPG/400 source program.

Figure 145 on page 286 and Figure 146 on page 287 show automatic report specifications for a sales report and the resulting RPG/400 source specifications that are generated for the report. Numbers are inserted in the figures to identify the automatic report functions and to show the specifications that are generated by each function.

Generated Calculations

Calculations are generated to accumulate totals for fields named on *AUTO field description specifications that have an A in position 39. (See Figure 147 on page 288.)

An RPG/400 subroutine is generated to accumulate the values from these fields into the lowest-level generated total fields. The name of the subroutine is always A\$\$SUM. The subroutine specifications are conditioned differently, depending on whether detail or group printing is specified:

- If detail printing is specified, as in Figure 147 on page 288, the EXSR statement is conditioned by the same indicator(s) that conditions the D-*AUTO specification (01 in this example). Each ADD statement in the subroutine is conditioned by the field indicator(s) specified with the field in its field description specification (none in this example).
- If group printing is specified, the EXSR statement and all ADD statements in the subroutine are unconditioned.

Total calculations are generated to roll the total from the lowest-level defined total field through the higher-level defined total fields and the final total. The total calculation to add the total from one level to that of the next higher level is conditioned by the control-level indicator corresponding to the field name of the lower level. As shown in Figure 147 on page 288, total calculations to accumulate L2 and LR totals are followed by the subroutine to accumulate the lowest level total, L1.

Generated total fields are defined (given length and number of decimal positions) when the total field is the result field in a generated calculation. In the input specifications, SOLDVA and VALUE are numeric fields defined with a length of seven and two decimal positions. Figure 147 on page 288 shows that the total fields generated from SOLDVA and VALUE are defined as two positions longer than the original fields, with the same number of decimal positions.

When group printing is specified (T-*AUTO specification), auto report generates total calculations to reset each of the accumulated fields (A in position 39) on the lowest level total line to zero on each cycle. A Z-ADD calculation, conditioned by L0, is generated for each accumulated field. These calculations are the first total calculations in the generated RPG/400 source program.

Generated Output Specifications

Figure 148 on page 289 shows the output specifications generated by automatic report. To identify specifications supplied by automatic report (column heading specifications, total specifications, conditioning indicators, spacing and skipping values, end position values, blank after), compare the listing with the automatic report specifications.

Automatic report generates specifications to reset accumulated fields to zero after they are printed. In this example, blank after is generated for accumulated fields.

Generated Specifications

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRINT 0 F 120 0A PRINTER 1
F*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* File description and input specifications for SALES file
I* are stored in the source-file member SALETR.
I/COPY SALETR 2
I*
I* Modifier statements follow the /COPY statement to add
I* control-level indicators.
IRcdname+....In.....*
```

I		BRANCHL1
I		REGIONL2

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
```

OPRINT H									
0			*AUTO						
0									
0	D	01	*AUTO						
0		L2	REGION						
0		L1	BRANCH						
0			ITEMNO						
0				C					
0			DESC						
0			SOLDQV						
0			SOLDVA	A					
0			ONHAND						
0			VALUE	A					
0				R					
0*									

Figure 145. Automatic Report Specifications for a Sales Transaction Report

Note: The following keys also refer to the corresponding numbers in the generated source program shown in Figure 146 on page 287.

- 1** Printer file description
- 2** Copy function and modifier statements
- 3** *AUTO page headings function
- 4** *AUTO output function
- 5** Accumulated fields

If you do not specify a control specification, automatic report generates a blank one for you.

```

1 H
2 FPRINT O F 120 OA PRINTER
3 FSALES IP F 43 DISK

4 I * /COPY SALETR
5 ISALES AA 01
6 I
7 I 1 7 ITEMNO
8 I 9 BRANCHL1
9 I 10 10 REGIONL2
10 I 11 25 DESC
11 I 26 270SOLD0Y
12 I 28 342SOLDVA
13 I 35 350ONHAND
13 I 37 432VALUE

14 C 01 EXSR ASSSUM
15 CL1 SOLDV2 ADD SOLDV1 SOLDV2 92
16 CL1 VALUE2 ADD VALUE1 VALUE2 92
17 CL2 SOLDVR ADD SOLDV2 SOLDVR 92
18 CL2 VALUER ADD VALUE2 VALUER 92
19 CSR ASSSUM BEGSR
20 CSR SOLDV1 ADD SOLDVA SOLDV1 92
21 CSR VALUE1 ADD VALUE VALUE1 92
22 CSR ENDSR

23 OPRINT H 206 1P
24 O OR OA
25 O UDATE Y 8
26 O 45 'SALES REPORT '
27 O 56 'FOR ANY CO. '
28 O 85 'PAGE '
29 O PAGE Z 89

30 OPRINT H 1 1P
31 O OR OA
32 O 6 'REGION '
33 O 14 'BRANCH '
34 O 21 'ITEM '
35 O 36 'DESCRIPTION '
36 O 47 'SALES '
37 O 62 'AMOUNT '
38 O 71 'ON -HAND '
39 O 86 'VALUE '

40 OPRINT H 2 1P
41 O OR OA
42 O 22 'NUMBER '
43 OPRINT D 1 01
44 O L2 REGION 3
45 O L1 BRANCH 12
46 O ITEMNO 23
47 O DESC 40
48 O SOLD0YK 46
49 O SOLDVAKB 62
50 O ONHANDK 69
51 O VALUE KB 86
52 OPRINT T 12 L1
53 O SOLDV1KB 62
54 O VALUE1KB 86
55 O 87 ' * '
56 OPRINT T 2 L2
57 O SOLDV2KB 62
58 O VALUE2KB 86
59 O 88 ' * * * '
60 OPRINT T 12 LR
61 O SOLDVRKB 62
62 O VALUERKB 86
63 O 47 'FINAL TOTALS '
64 O 89 ' * * * * '

```

Figure 146. RPG/400 Source Program Generated from Automatic Report Specifications

Generated Specifications

Note: These numbers refer to the corresponding numbers shown on the automatic report specifications shown in Figure 145 on page 286.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
OPRINT H *AUTO
0 'SALES REPORT '
0 'FOR ANY CO.'
0 D 01 *AUTO
0 L2 REGION 'REGION'
0 L1 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 C 'NUMBER'
0 DESC 'DESCRIPTION'
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A 'VALUE'
0 R 'FINAL TOTALS'
0*
0* Calculations are generated for fields with an A in position 39.
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C 01 EXSR A$$SUM
CL1 SOLDV2 ADD SOLDV1 SOLDV2 92
CL1 1 VALUE2 ADD VALUE1 VALUE2 92
CL2 SOLDVR ADD SOLDV2 SOLDVR 92
CL2 VALUER ADD VALUE2 VALUER 92
CSR A$$SUM BEGSR
CSR 2 SOLDV1 ADD SOLDVA SOLDV1 92 3
CSR VALUE1 ADD VALUE VALUE1 92
CSR ENDSR

```

Figure 147. Calculations Generated from Automatic Data Structure to Accumulate Totals—Example 2

- 1** Total calculations roll higher-level totals.
- 2** Subroutine accumulates the lowest level totals (L1 in this example).
- 3** Length and decimal position of generated total fields.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
OPRINT H *AUTO
0 'SALES REPORT '
0 'FOR ANY CO.'
0 D 01 *AUTO
0 L2 REGION 'REGION'
0 L1 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 C 'NUMBER'
0 DESC 'DESCRIPTION'
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A 'VALUE'
0 R 'FINAL TOTALS'
0*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
OPRINT H 206 1P
0 OR OA
0 UPDATE Y 8
0 45 'SALES REPORT '
0 56 'FOR ANY CO.'
0 85 'PAGE'
0 PAGE Z 89

```

Figure 148 (Part 1 of 3). Output Specifications Generated Data Structure to Accumulate Totals—Example 2

Generated Specifications

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
OPRINT   H 1      1P
0        OR      OA
0
0          6 'REGION'
0          14 'BRANCH'
0          21 'ITEM'
0          36 'DESCRIPTION'
0          47 'SALES'
0          62 'AMOUNT'
0          71 'ON-HAND'
0          86 'VALUE'
OPRINT   H 2      1P
0        OR      OA
0
0          22 'NUMBER'
OPRINT   D 1      01
0          L2     REGION    3
0          L1     BRANCH   12
0          ITEMNO 12
0          DESC   12
0          SOLDQYK 46
0          SOLDVAKB 62
0          ONHANDK 69
0          VALUE KB 86

```

1

Figure 148 (Part 2 of 3). Output Specifications Generated Data Structure to Accumulate Totals—Example 2

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
OPRINT   T 12     L1
0          SQLDV1KB 62
0          VALUE1KB 86
0          87 '*'
OPRINT   T 2      L2
0          SQLDV2KB 62
0          VALUE2KB 86
0          88 '**'
OPRINT   T 12     LR
0          SQLDVRKB 62
0          VALUERKB 86
0          47 'FINAL TOTALS'
0          89 '***'

```

2

Figure 148 (Part 3 of 3). Output Specifications Generated Data Structure to Accumulate Totals—Example 2

- 1** Two heading specifications are generated for column headings because ITEM NUMBER is a two-line heading.
- 2** Automatic report generates total specifications to print accumulated totals for SOLDVA and VALUE fields.

Programming Aids

The chart shown in Table 17 on page 292 should be helpful in determining valid *AUTO output entries depending on the contents of position 39.

The following programming suggestions may be helpful in specific programming situations:

- One column heading can be printed over two or more fields if automatic column spacing is taken into consideration. For example, if the heading DATE is to print over a month field and a day field as follows:

	D		A		T		E		
		M	O	N			D	A	Y
			X	X			X	X	
			X	X			X	X	

Code the output specifications as follows:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0                MONTH                'D A'
0                C                'MON'
0                DAY                'T E'
0                C                'DAY'
0*
```

- To print a constant on only the first detail line under a column heading, move the constant to a field in calculation specifications and print that field as shown in Figure 149 on page 292.
- If group printing is being done and more than one record type is present in the input file, certain precautions must be taken. If a field to be accumulated is present in all record types, but only one record type is to be processed, the correct total is not generated unless additional coding is used. The specifications shown in Figure 150 on page 293 give incorrect results because the T-*AUTO specification causes an unconditioned ADD subroutine to be generated if a field is to be added. Therefore, QTY is added when indicator 10 is on and when indicator 11 or 12 is on. Figure 151 on page 293 shows a method of obtaining the correct results.
- Figure 152 on page 294 shows the specifications for counting records. This method is especially applicable when you want to print a detail list, to take totals by control level, or to eliminate 1's from being listed down the page.

Programming Aids

*Table 17. Valid *AUTO Entries Depending on the Contents of Position 39*

39	7-22	23-31	32-37	38	40-43	44	45-70
Blank	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
	Blank	Blank or indicators	Blank	Blank	Blank or end position	Blank	Constant
B	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
A	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
C	Blank	Blank	Blank	Blank	Blank	Blank	Column heading
1-9, R	Blank	Blank	Field name	Blank or edit code	Blank	Blank	Blank or edit word
	Blank	Blank	Blank	Blank	Blank	Blank	Constant

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* Assume L1 is defined in positions 59 and 60 on input specifications.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C L1 MOVE 'CONSTANT'FLDA 8
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0 D *AUTO
0 FLDA B 'COLUMN HEADING'
0*

```

Figure 149. Printing a Constant Only on the Detail Line

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IINPUT  AA  10  1 CA
I.....PFromTo++DField+L1M1FrP1MnZr...*
I                2  27 NAME  L1
I          BB  11  1 C1
I          OR  12  1 CN
I                2  18 DESC
I                19  21QTY
I                22  262SALES
I*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT  T          L1      *AUTO
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
O                DESC          'DESCRIPTION'
O                QTY   A      'QUANTITY'
O                SALES A      'AMOUNT'
O*
```

Figure 150. Incorrect *AUTO Specifications for More Than One Record Type

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          11                Z-ADDQTY          QTYA   30
C          11                Z-ADDSALES        SALES  52
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT  T          L1      *AUTO
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
O                DESC          'DESCRIPTION'
O                QTYA  A      'QUANTITY'
O                SALES A      'AMOUNT'
O*
```

Figure 151. Correct *AUTO Specifications for More Than One Record Type

Using CRTRPTPGM to Compile an Auto Report Program

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C      1          Z-ADD0          COUNT  30
C      2      1          ADD  COUNT1  COUNT1
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
O      D          *AUTO
O.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
O      3          LR      COUNT  A      'RECORD'
O          C          'COUNT'
O*
```

Figure 152. Method of Using *AUTO for Counting Records

Calculation Specifications

- 1 This instruction is needed only to define the field COUNT for accumulation.
- 2 This instruction accumulates the total for the first control level.

Output Specifications

- 3 This instruction causes the generation of calculation and output specifications for the detail and total lines. The LR conditioning indicator prevents the generated detail calculation from occurring. It also prevents printing at detail time.

Note: If no control levels are specified in the program, a 1 is added to COUNTR rather than to COUNT1 on the calculation specifications.

Using CRTRPTPGM to Compile an Auto Report Program

To compile an RPG/400 source program that includes automatic report specifications, you must use the CL command CRTRPTPGM (Create Automatic Report Program). RPG/400 program objects are created with the public authority of *CHANGE. You may want to change this authority to maintain greater security on your system.

Automatic report does not diagnose all error conditions in the source program. Test results that are produced by the RPG/400 compiler are not duplicated by automatic report. If a program cannot be successfully generated because of errors in the automatic report specifications, automatic report ends. If automatic report stops, the escape message RPT 9001 is issued. A CL program can monitor for the escape message by using the CL command MONMSG (Monitor Message).

If an RPG/400 source program is successfully generated and the *NOCOMPILE option is not specified on the CRTRPTPGM command, automatic report calls the RPG/400 compiler.

All object names specified on the CRTRPTPGM command use the full naming convention. The length of the name cannot exceed ten characters. See the *CL Refer-*

ence for a detailed description of the OS/400 object naming rules and for a complete description of the OS/400 command syntax.

Using the CRTRPTPGM Command

You call the CRTRPTPGM compiler in three ways:

- Interactively from a display. Type the command CRTRPTPGM and then press F4
- Using keyword parameters
- Using positional parameters.

See the description of these features of the CRTRPGPGM command in Chapter 3, “Compiling an RPG/400 Program” on page 25 for details.

CRTRPTPGM Command

The CRTRPTPGM command is similar to the CRTRPGPGM command described in Chapter 3, “Compiling an RPG/400 Program” on page 25. All object names must consist of alphanumeric characters. The first character must always be alphabetic, and the length of the name cannot exceed 10 characters.

The CRTRPTPGM command recognizes all the parameters that the CRTRPGPGM command does. Some of these parameters, however, are not used by automatic report itself, but are passed on to the RPG/400 compiler. These are the PGM, OPTION, GENOPT, GENLVL, USRPRF, AUT, TEXT, PHSTRC, TGTRLS, INDENT, and REPLACE parameters. The PRTFILE parameter specifies a file that automatic report itself uses, and then passes on to the RPG/400 compiler.

The CRTRPTPGM command has the same parameters as the CRTRPGPGM command plus three others: RPTOPT, OUTFILE, and OUTMBR. The description of these parameters follows the syntax diagram for the CRTRPTPGM command. The defaults are explained first and are underlined. See “Create RPG400 Program (CRTRPGPGM) Command” on page 26 for the definition of the other parameters.

The CRTRPTPGM command can be submitted in a batch input stream, entered interactively at a work station, or in a CL or REXX program.

For information on how to read syntax diagrams, see “How to Interpret Syntax Diagrams” on page xiv.

Using CRTRPTPGM to Compile an Auto Report Program

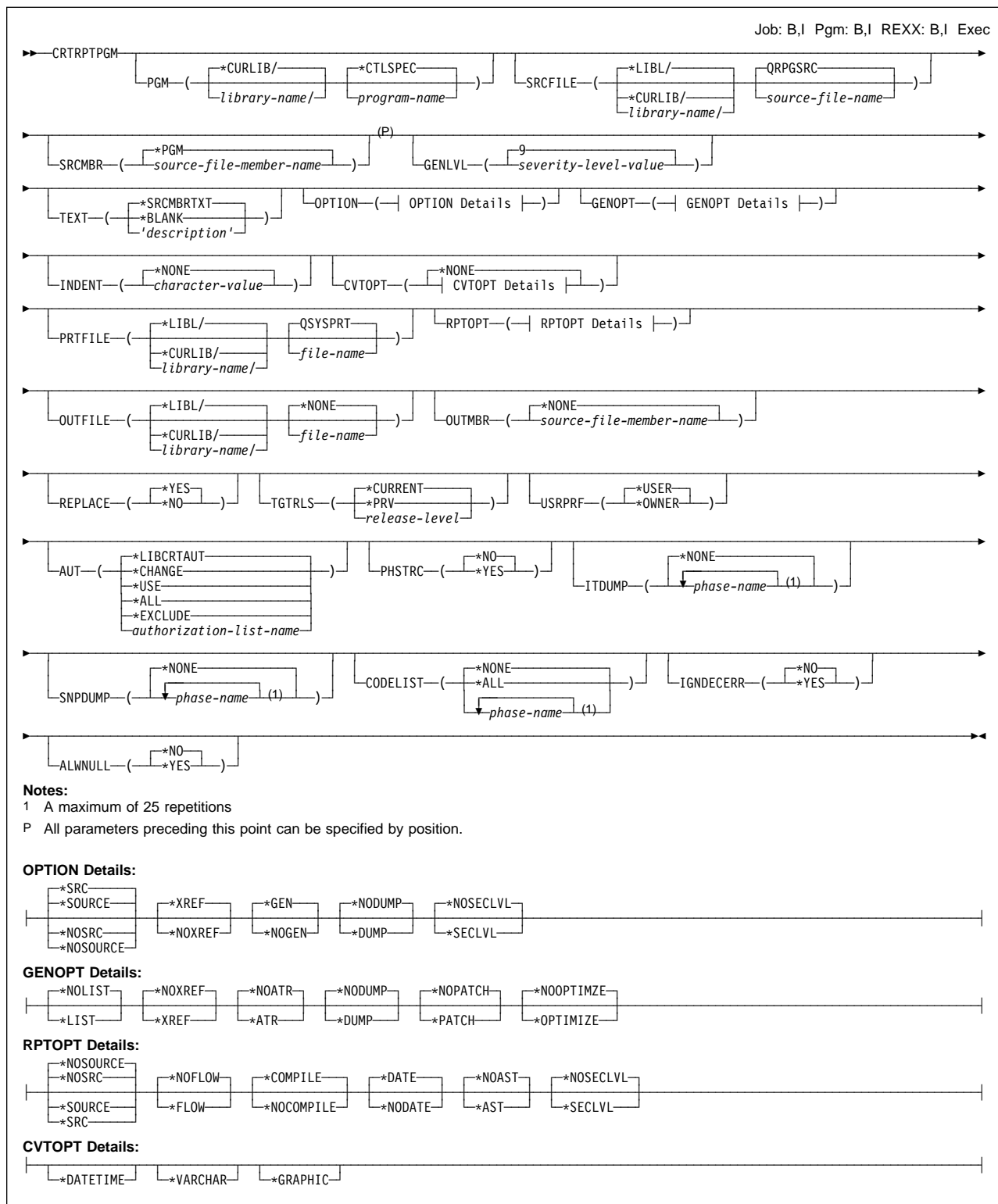


Figure 153. Syntax of the CRTRPTPGM Command

RPTOPT

Specifies the options to use when the source program is compiled. Any or all of the following keyword options can be specified in any order. Separate the keywords with a delimiter. The possible values are:

***NOSOURCE**

Do not produce a listing of the automatic report source program compile-time errors.

***SOURCE**

Produce a listing of the automatic report source program compile-time errors. The acceptable abbreviation for *NOSOURCE is *NOSRC and for *SOURCE is *SRC.

***NOFLOW**

Do not write a flow of the major routines run while the automatic report source program is compiled.

***FLOW**

Write a flow of the major routines run while the automatic report source program is compiled.

***COMPILE**

Call the RPG/400 compiler after the automatic report source statements are processed, and the complete RPG/400 source program is generated.

***NOCOMPILE**

Do not call the RPG/400 compiler after the automatic report source statements are processed.

***DATE**

Include the page number and date on the first *AUTO page heading line.

***NODATE**

Do not include the page number and date on the first *AUTO page heading line.

***NOAST**

Do not generate asterisk indication for total output lines.

***AST**

Generate asterisk indication for total output lines.

***NOSECLVL**

Do not print second-level text on the line following the first level message text.

***SECLVL**

Print second-level text on the line following the first level message text.

OUTFILE

Specifies the name of the file where the complete RPG/400 source program is to be placed and the library in which the file is located. The file is also used as the source input file to the RPG/400 compiler unless the RPTOPT parameter value *NOCOMPILE is specified.

***LIBL**

The library list is used to locate the file.

***CURLIB**

The name of the current library. If a current library is not specified, QGPL is the current library.

Using CRTRPTPGM to Compile an Auto Report Program

library-name

Enter the name of the library in which the file is located.

*NONE

Create a file in QTEMP to pass the generated RPG/400 source to the RPG/400 compiler.

file-name

Enter the name of the file to contain the complete RPG/400 source program.

OUTMBR

Specifies the name of the member of the file that will contain the output from automatic report.

*NONE

Use the first member created in or added to the file as the member name.

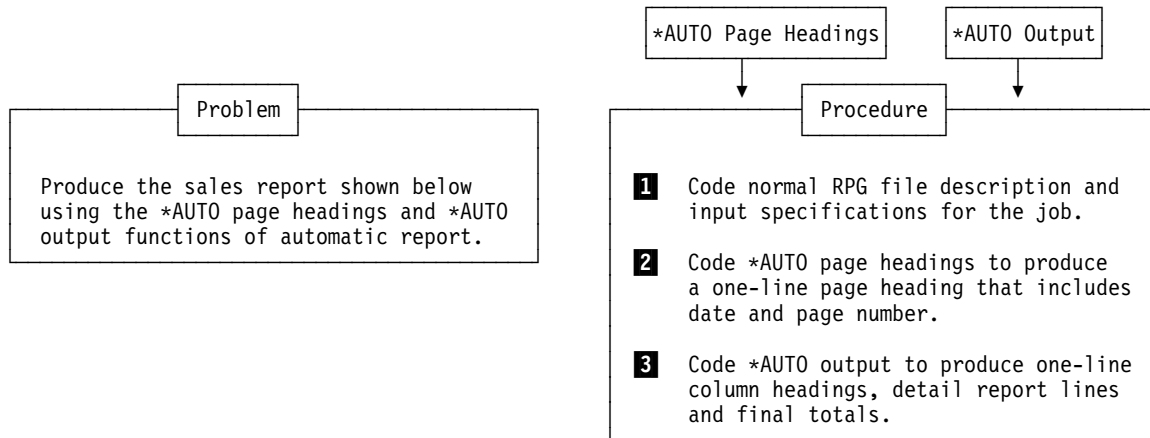
file-member-name

Enter the name of the member that is to contain the output of automatic report.

Examples of Using Automatic Report

Examples 1 through 4 explain how automatic report is used to generate report page headings and such output specifications as column headings, detail lines, and total lines. Examples 5 and 6 illustrate the use of the automatic report copy function to copy specifications from a source-file member and to change copied specifications for a particular job.

EXAMPLE 1 - Sales Report



Letters refer to fields on the following page.

10/26/80			SALES REPORT FOR ANY CO.				PAGE 1	
C	B	A	D	E	F	G	H	
REGION	BRANCH	ITEM	DESCRIPTION	SALES	AMOUNT	ON-HAND	VALUE	
1	17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00	
1	17	AG77055	PICK-UP	10	20,000.00	1	2,000.00	
1	17	AP6545B	CAMPER	2	8,000.00			
1	22	AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00	
1	22	AG77055	PICK-UP	4	8,000.00	1	2,000.00	
3	25	AG6545B	CAMPER	10	40,000.00	5	20,000.00	
3	25	AP6549P	1/4 TON TRUCK	20	30,000.00	6	9,000.00	
					141,000.00		48,000.00 *	

Examples of Using Auto Report

1 RPG/400 file description and input specifications.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
```

```
FSALES IP F 43 DISK
FPRINT 0 F 120 PRINTER
```

F*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
```

```
ISALES AA 01
```

```
I.....PFromTo++DField+L1M1FrP1MnZr...*
```

```
I 1 7 ITEMNO
```

```
I 8 9 BRANCH
```

```
I 10 10 REGION
```

```
I 11 25 DESC
```

```
I 26 270SOLDQY
```

```
I 28 342SOLDVA
```

```
I 35 360ONHAND
```

```
I 37 432VALUE
```

I*

	Field Name	Contents
A	ITEMNO	Item number
B	BRANCH	Number of the branch office where the item was sold
C	REGION	Sales region in which the branch office is located
D	DESC	Description of the sales item
E	SOLDQY	Quantity of the item sold
F	SOLDVA	Total value of the items sold
G	ONHAND	Quantity of the item remaining on hand
H	VALUE	Total value of the items remaining on hand.

2 *AUTO page heading specifications.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H A C D *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0                                     'SALES REPORT ' B
0                                     'FOR ANY CO.'
0*
```

- A** Enter an H in position 15 and *AUTO in positions 32 through 36 to request an automatic report page heading. Up to five page heading lines can be described. The system date is printed on the left and the page number on the right of the first heading line on each page. To suppress the date and page, enter an N in position 27 of the automatic report option specifications or use the *NODATE option on the RPTOPT parameter in the CRTRPTPGM command.
- B** The title information is centered by automatic report; do not enter end positions in positions 40 through 43. Fields and array/table elements can also be used.
- C** When space and skip entries (positions 17 through 22) are left blank, skip to line 06 is assumed for the first heading line; single spacing is done between heading lines, double spacing after the last heading line. (See "Example 4" for an example of multiple page heading lines.)
- D** When output indicators (positions 23 through 31) are left blank, automatic report page headings are printed on each page (conditioned by 1P or overflow). If no overflow indicator is defined for the printer file, automatic report assigns an unused overflow indicator to the printer line.

Examples of Using Auto Report

3 Code *AUTO output specifications to produce:

REGION	BRANCH	ITEM	DESCRIPTION	SALES	AMOUNT ON-HAND	VALUE
1	17	A97701T	2-TON TRUCK	8	25,000.00	2 10,000.00
1	17	A97708B	PICK-UP	10	20,000.00	1 2,000.00
1	17	AP8846B	CAMPER	2	8,000.00	
1	22	A97701T	2-TON TRUCK	2	10,000.00	1 5,000.00
1	22	A97708B	PICK-UP	4	8,000.00	1 2,000.00
3	25	AG8846B	CAMPER	10	40,000.00	5 20,000.00
3	25	AP8849P	1/4 TON TRUCK	20	30,000.00	8 9,000.00
				141,000.00		48,000.00 *

- A** Detail report lines
- B** Column headings
- C** Final totals

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
0.....N01N02N03Field+YBEnd+PCConstant/editword+++++...*
0
0 'SALES REPORT '
0 'FOR ANY CO.'
0 A D 01 *AUTO
0 REGION 'REGION'
0 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 DESC 'DESCRIPTION' B
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A C 'VALUE'
0*

```

- A** Enter D in position 15 and *AUTO in positions 32 through 36 to describe an automatic report with detail lines. The record-identifying indicator 01 conditions printing of the detail lines.
- B** Column headings are entered on the same line as the fields over which they appear in the report.
- C** Enter an A in position 39 to cause fields to be accumulated. Automatic report generates (1) total fields and calculations to accumulate the totals, and (2) total output specifications to print the totals.

10/26/80

SALES REPORT FOR ANY CO.

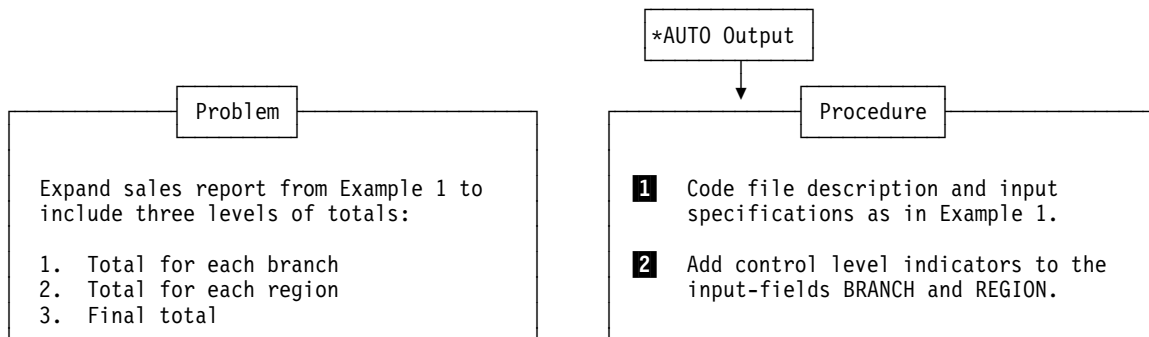
PAGE 1

REGION	BRANCH	ITEM	DESCRIPTION	SALES	AMOUNT ON - HAND	VALUE
1	17	AG7701T	2-TON TRUCK	5	25,000.00	2 10,000.00
1	17	AG77055	PICK-UP	10	20,000.00	1 2,000.00
1	17	AP6545B	CAMPER	2	8,000.00	
1	22	AG7701T	2-TON TRUCK	2	10,000.00	1 5,000.00
1	22	AG77055	PICK-UP	4	8,000.00	1 2,000.00
3	25	AG6545B	CAMPER	10	40,000.00	5 20,000.00
3	25	AP6549P	1/4 TON TRUCK	20	30,000.00	8 9,000.00
					141,000.00	48,000.00 *

Automatic report formats the report so that column headings and data are neatly spaced and centered on each other.

All numeric fields for which a blank, B, or A is specified in position 39 are edited by the K edit code unless a different edit code is specified.

EXAMPLE 2 - Sales Report with Three Levels of Totals



Note: The *AUTO output function can also be used to produce a group printed report. See “Group Printing” on page 271 for a discussion and examples of group printing.

1 RPG/400 file description and input specifications.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES AA 01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7 ITEMNO
I          8  9 BRANCHL1 2
I         10 10 REGIONL2
I         11 25 DESC
I         26 270SOLDQY
I         28 342SOLDVA A
I         35 360ONHAND
I         37 432VALUE
I*
```

A Because two control levels are defined, the SOLDVA and VALUE fields (see following page) are accumulated to two levels of totals (branch and region) and a final total (LR).

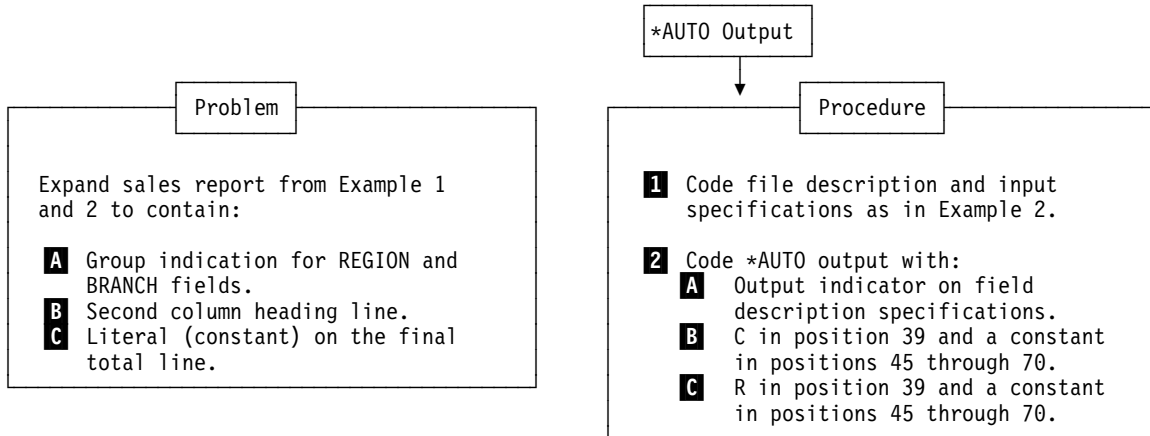
```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H                      *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0                               'SALES REPORT '
0                               'FOR ANY CO.'
0      D  A      01      *AUTO
0                               REGION      'REGION'
0                               BRANCH     'BRANCH'
0                               ITEMNO     'ITEM'
0                               DESC        'DESCRIPTION'
0                               SOLDQY     'SALES'
0                               SOLDVA A  B 'AMOUNT'
0                               ONHAND     'ON-HAND'
0                               VALUE  A   'VALUE'
0*

```

- A** Automatic report places a blank line after each total line and an additional blank line before the lowest level total and before the final total. If you enter spacing and skipping values on the D-*AUTO specification, they apply to the detail print line only.
- B** As in “EXAMPLE 1 - Sales Report,” an A in position 39 of the output specification causes SOLDVA and VALUE to be accumulated.
- C** Total fields are always two positions longer with the same number of decimal positions as the original fields.
- D** Automatic report prints asterisks (*) to the right of the generated total lines to aid in identifying them. If you want to suppress the asterisks, enter N in position 28 of the automatic report option specifications or use the *NOAST option on the RPTOPT parameter in the CRTRPTPGM command.

EXAMPLE 3 - Sales Report with Group Indication



1 RPG/400 file description and input specifications.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES  IP  F      43                DISK
FPRINT  0  F      120                PRINTER
  
```

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES  AA  01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7  ITEMNO
I          8  9  BRANDHL1
I         10 10  REGIONL2
I         11 25  DESC
I         26 27 0SOLDQY
I         28 34 2SOLDVA
I         35 36 0ONHAND
I         37 43 2VALUE
I*
  
```

Examples of Using Auto Report

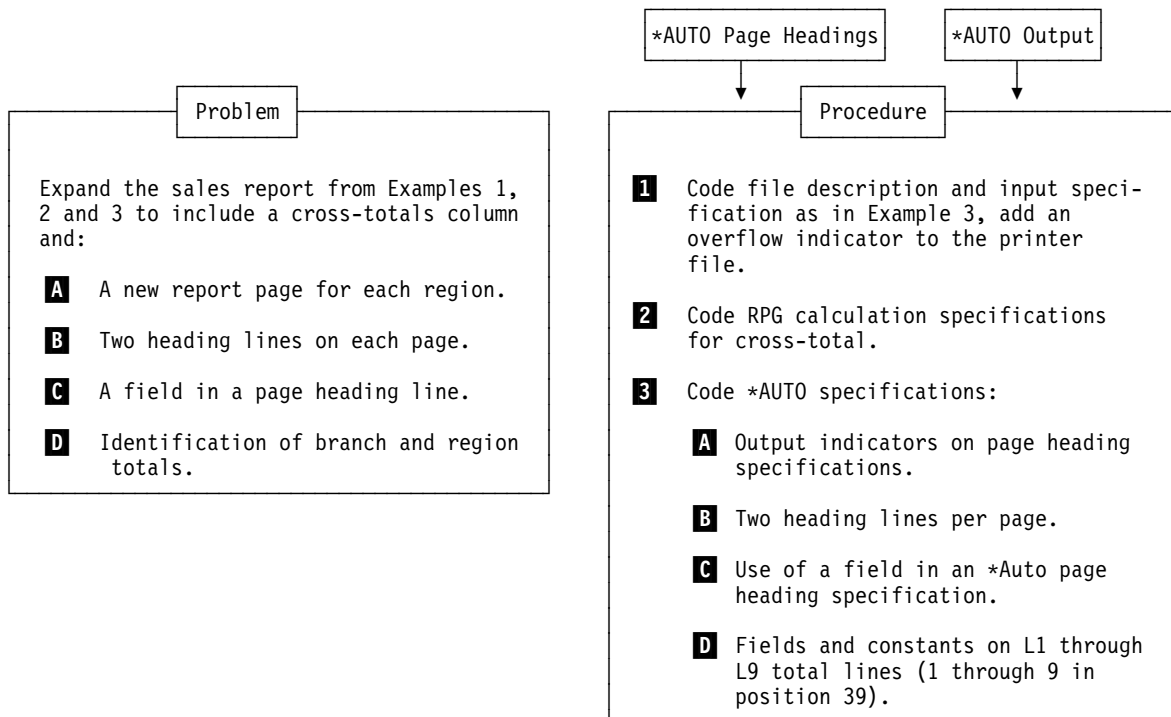
2 *AUTO output.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
0.....N01N02N03Field+YBEnd+PCConstant/editword+++++++...*
0 'SALES REPORT '
0 'FOR ANY CO.'
0 D 01 *AUTO
0 A L2 REGION 'REGION'
0 L1 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 B C 'NUMBER'
0 DESC 'DESCRIPTION'
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A 'VALUE'
0 C R 'FINAL TOTALS'
0*
```

- A** Output indicators can be used on field description specifications. In this example, control-level indicators condition BRANCH and REGION so that they are printed only for the first record of the corresponding control group. This print suppressing of common fields (group indication) reduces repetitive information.
- B** One or two additional column heading lines can be specified by a C entry in position 39 with the heading information in positions 45 through 70.
- C** The literal FINAL TOTALS makes that line easy to find. To specify information to appear on the final total line, enter R in position 39 with a constant in positions 45 through 70 or a field name/table name/indexed array name in positions 32 through 37. The information is printed two spaces to the left of the leftmost total on the line. If more than one such specification is used, the constants and fields are printed from left to right in the order they are specified in the program.

10/26/80		B		SALES REPORT FOR ANY CO.			PAGE 1		
REGION	BRANCH	ITEM NUMBER	DESCRIPTION	SALES	AMOUNT ON-HAND		VALUE		
A	1	17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00	
			AG77055	PICK-UP	10	20,000.00	1	2,000.00	
			AP6545B	CAMPER	2	8,000.00			
						53,000.00		12,000.00 *	
		22		AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00
				AG77055	PICK-UP	4	8,000.00	1	2,000.00
									18,000.00
						71,000.00		19,000.00 **	
	3	25		AG6545B	CAMPER	10	40,000.00	5	20,000.00
AP6549P				1/4 TON TRUCK	20	30,000.00	6	9,000.00	
								70,000.00	29,000.00 *
					70,000.00		29,000.00 **		
C FINAL TOTALS					141,000.00		48,000.00 ***		

EXAMPLE 4 - Sales Report with Cross-Column Totals



11/11/80		SALES REPORT FOR ANY CO.					PAGE 1
		REGION 1					
BRANCH	ITEM NUMBER	DESCRIPTION	SALES QUANTITY	SALES VALUE	ON HAND	ON-HAND VALUE	TOTAL
17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00	35,000.00
	AG77055	PICK-UP	10	20,000.00	1	2,000.00	22,000.00
	AP6545B	CAMPER	2	8,000.00			8,000.00
		D BRANCH 17 TOTALS		53,000.00		12,000.00	65,000.00 *
22	AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00	15,000.00
	AG77055	PICK-UP	4	8,000.00	1	2,000.00	10,000.00
		BRANCH 22 TOTALS		18,000.00		7,000.00	25,000.00 *
		D REGION 1 TOTALS		71,000.00		19,000.00	90,000.00 **
							A
11/11/80		SALES REPORT FOR ANY CO.					PAGE 2
		REGION 3					
BRANCH	ITEM NUMBER	DESCRIPTION	SALES QUANTITY	SALES VALUE	ON HAND	ON-HAND VALUE	TOTAL
25	AG6545B	CAMPER	10	40,000.00	5	20,000.00	60,000.00
	AG6549P	1/4 TON TRUCK	20	30,000.00	6	9,000.00	39,000.00
		BRANCH 25 TOTALS		70,000.00		29,000.00	99,000.00 *
		REGION 3 TOTALS		70,000.00		29,000.00	99,000.00 **
		COMPANY TOTALS		141,000.00		48,000.00	189,000.00 ***

Note: Compare matching letters (**B**) on this and the following pages to see the automatic report coding to obtain this report.

2 RPG/400 calculations can be among the input statements for automatic report. This specification calculates a cross-total of the sales and on-hand values. The placement of the calculation relative to calculations generated by automatic report is described under "Generated Specifications" on page 284.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C 01      SOLDVA      ADD VALUE      TOTVAL 82
C*
```

Examples of Using Auto Report

3 *AUTO specifications.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaNO1N02N03Excnam.....*
OPRINT   H       A L2       *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0         OR         OFNL2
0
0         'SALES REPORT '
0         'FOR ANY CO.'
0         H           *AUTO
0         B           'REGION'
0         REGION C
0         D         01    *AUTO
0         L1          BRANCH 'BRANCH'
0         ITEMNO      'ITEM'
0         C          'NUMBER'
0         DESC       'DESCRIPTION'
0         SOLDQY     'SALES'
0         C          'QUANTITY'
0         SOLDVA A  'SALES VALUE'
0         ONHAND   'ON'
0         C          'HAND'
0         VALUE A  'ON-HAND VALUE'
0         TOTVAL A  'TOTAL'
0         1          'BRANCH'
0         BRANCH 1   'TOTALS'
0         1          'REGION'
0         2
0         REGION 2   'TOTALS'
0         2          'COMPANY TOTALS'
0         R
0*

```

- A** The headings are printed on a new page when the region number changes (L2) or when overflow occurs (OF). (OF must be defined for the printer file in file description specifications).
- B** A second automatic report page heading is specified. Because spacing is not specified, space-one is done after the first and space-two after the second. Because no output indicators are specified, the second heading is conditioned like the first.
- C** The contents of the REGION field are printed on the second page heading.
- D** Fields and constants can be printed on generated total lines if you enter the number of the control level in position 39.

EXAMPLE 5 - Sales Report Using Copied Specifications

COPY

Problem

Use the copy function to obtain specifications for the sales report below (same as in Example 1).

Procedure

- 1** Save the file description and input specifications for the SALES file in a source file member.
- 2** Code the /COPY statement in the specifications for auto report.

10/26/80		SALES REPORT FOR ANY CO.				PAGE 1	
REGION	BRANCH	ITEM	DESCRIPTION	SALES	AMOUNT	ON-HAND	VALUE
1	17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00
1	17	AG77055	PICK-UP	10	20,000.00	1	2,000.00
1	17	AP6545B	CAMPER	2	8,000.00		
1	22	AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00
1	22	AG77055	PICK-UP	4	8,000.00	1	2,000.00
3	25	AG6545B	CAMPER	10	40,000.00	5	20,000.00
3	25	AP6549P	1/4 TON TRUCK	20	30,000.00	6	9,000.00
					141,000.00		48,000.00 *

1 Specifications for the SALES file are stored in a source-file member.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FSALES IP F 43 DISK A
FPRINT 0 F 120 PRINTER
```

A These specifications could be replaced by a single statement as shown on the following page.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES AA 01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  7 ITEMNO
I          8  9 BRANCH
I         10 10 REGION
I         11 25 DESC
I         26 270SOLDQY
I         28 342SOLDVA
I         35 360ONHAND
I         37 432VALUE
I*
```

A

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
O.....N01N02N03Field+YBEnd+PCconstant/editword+++++...*
O          'SALES REPORT '
O          'FOR ANY CO.'
O      D      01      *AUTO
O          REGION      'REGION'
O          BRANCH      'BRANCH'
O          ITEMNO      'ITEM'
O          DESC        'DESCRIPTION'
O          SOLDQY      'SALES'
O          SOLDVA A    'AMOUNT'
O          ONHAND      'ON-HAND'
O          VALUE A    'VALUE'
O*
```

Examples of Using Auto Report

- 2** Code the /COPY statement to include the file description and input specifications. (For a detailed description of the copy function, see “/COPY Statement Specifications” on page 275.)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRINT 0 F 120 PRINTER
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
O/COPY SALETR B
0*
```

- A** Position 6 of a /COPY statement must not contain a U or an H.
- B** The /COPY statement copies file description and input specifications for the SALES file from the member named SALETR.

The /COPY statement can appear anywhere among the automatic report specifications following the automatic report option statement and preceding array and table input records. It is convenient to code the /COPY on the input specifications when you want to override copied input specifications, as in “Example 6”. After specifications are copied, all specifications are sorted into the order required by the RPG/400 compiler.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0 'SALES REPORT '
0 'FOR ANY CO.'
0 D 01 *AUTO
0 REGION 'REGION'
0 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 DESC 'DESCRIPTION'
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A 'VALUE'
0*
```

EXAMPLE 6 - Override Copied Input Specifications

COPY

Problem

Override copied input specifications to produce a report (below) that includes subtotals for branch and region.

Procedure

- 1** Save specifications for the SALES file, as in *Example 5*.
- 2** Code the /COPY statement.
- 3** Code /COPY modifier statements to add control level indicators to BRANCH and REGION fields on copied specifications.

10/26/80		SALES REPORT FOR ANY CO.				PAGE 1	
REGION	BRANCH	ITEM	DESCRIPTION	SALES	AMOUNT	ON-HAND	VALUE
1	17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00
1	17	AG77055	PICK-UP	10	20,000.00	1	2,000.00
1	17	AP6545B	CAMPER	2	8,000.00		
					53,000.00		12,000.00 *
1	22	AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00
1	22	AG77055	PICK-UP	4	8,000.00	1	2,000.00
					18,000.00		7,000.00 *
					71,000.00		19,000.00 **
3	25	AG6545B	CAMPER	10	40,000.00	5	20,000.00
3	25	AP6549P	1/4 TON TRUCK	20	30,000.00	6	9,000.00

Examples of Using Auto Report

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ISALES AA 01
I.....PFfromTo++DField+L1M1FrP1MnZr...*
I          1  7 ITEMNO
I          8  9 BRANCH A
I         10 10 REGION
I         11 25 DESC
I         26 270SOLDQY
I         28 342SOLDVA
I         35 360ONHAND
I         37 432VALUE
I*
```

- A** To produce a report that has subtotals for branch and region, L1 must be assigned to **BRANCH** and L2 to **REGION** as the specifications are copied from the source-file member.

2 and **3** Code /COPY and modifier statements. As a result of the modifier statements, three levels of totals are accumulated for the SOLDVA and VALUE fields (L1, L2 and LR).

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRINT 0 F 120 PRINTER
F*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I/COPY SALETR
I.....PFromTo++DField+L1M1FrP1MnZr...*
I BRANCHL1 A
I REGIONL2 B
I*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OPRINT H *AUTO
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0 'SALES REPORT '
0 'FOR ANY CO.'
0 D 01 *AUTO
0 REGION 'REGION'
0 BRANCH 'BRANCH'
0 ITEMNO 'ITEM'
0 DESC 'DESCRIPTION'
0 SOLDQY 'SALES'
0 SOLDVA A 'AMOUNT'
0 ONHAND 'ON-HAND'
0 VALUE A 'VALUE'
0*
```

A Entries on the modifier statements override the corresponding entries in the copied specifications.

B The field names, BRANCH and REGION, identify the input-field specifications that are to be changed.

Saved file description or input specifications are overridden as follows (see “/COPY Statement Specifications” on page 275 for examples):

- Entries in a modifier statement override corresponding entries in a copied file description or input field specification.
- Blank entries in a modifier statement remain unchanged in a copied specification.
- Ampersand (&) in the leftmost position of an entry in the modifier statement sets the entry to blanks in the copied specification.

Examples of Using Auto Report

- New fields can be added to input specifications by new input-field specifications added as modifier statements.
- Modifier statements do not change the saved specifications. The modification is only for the program into which the specifications are copied.

Chapter 13. RPG/400 Sample Programs

This chapter contains a sample application consisting of a series of RPG/400 programs that could run on the OS/400 system. The sample programs are scaled in such a way that you can use the *RPG Debugging Template*, GX21-9129 to check the coding in the programs.

A time reporting application has been chosen for the sample programs. The design does not attempt to provide a complete time reporting system, but is designed to illustrate RPG/400 programs. The chapter consists of:

- Application scope and objectives
- System Overview
- Database design
- Technical design including:
 - Master file maintenance
 - Data area control file maintenance
 - Transaction entry
 - Weekly processing
 - Monthly processing
 - Year end processing.

The following sample programs are cited throughout this guide and the *RPG/400 Reference*.

Note: Before the sample application will run successfully, the physical files must contain initial data and the data area CTLFIL must be created and initialized.

Checklist of Program Examples

All RPG/400 functions, operation codes, and features that are included in the program examples are shown in Table 18. Beside each function or operation code are the program names. Where a function is used in more than one program, all occurrences are listed.

Note: Refer to Table 19 on page 324 for a list of the programs in the order they appear in this chapter.

<i>Table 18 (Page 1 of 3). Functions, Operation Codes, and Features of RPG/400 Sample Programs</i>		
Specification Form	Function/Operation Code Description	Programs
File Description	Program-described files	PRG02 PRG09
	Externally described files	PRG01 PRG03 PRG04 PRG05 PRG06 PRG07 PRG08
	Disk files	PRG01 PRG03 PRG04 PRG05 PRG06 PRG07 PRG08 PRG09
	Work station files	PRG01 PRG02 PRG03
	Printer files	PRG06 PRG07 PRG08 PRG09
	Table files	PRG09
Extension	Array	PRG01 PRG02 PRG03 PRG05 PRG06 PRG07 PRG08
	Table	PRG09

Checklist of Program Examples

<i>Table 18 (Page 2 of 3). Functions, Operation Codes, and Features of RPG/400 Sample Programs</i>		
Specification Form	Function/Operation Code Description	Programs
Input	Program-described Externally described Data structures Named constants	PRG02 PRG09 PRG01 PRG03 PRG04 PRG05 PRG06 PRG07 PRG08 PRG02 PRG03 PRG05 PRG06 PRG07 PRG08 PRG09 PRG02 PRG03 PRG09
Calculation	Operation codes: ADD ANDXX BEGSR CABXX CALL CASXX CAT CHAIN CLEAR CLOSE COMP DEFN DELET DIV DOUXX DOWXX DSPLY ELSE END ENDSR EXCPT EXFMT EXSR FREE GOTO IFXX IN KFLD KLIST LOKUP	PRG03 PRG04 PRG06 PRG07 PRG08 PRG09 PRG01 PRG02 PRG03 PRG09 PRG01 PRG02 PRG03 PRG04 PRG06 PRG07 PRG08 PRG09 PRG01 PRG05 PRG04 PRG08 PRG01 PRG03 PRG06 PRG07 PRG08 PRG09 PRG08 PRG05 PRG02 PRG05 PRG03 PRG02 PRG06 PRG07 PRG04 PRG03 PRG05 PRG01 PRG02 PRG03 PRG05 PRG06 PRG01 PRG02 PRG03 PRG04 PRG05 PRG06 PRG07 PRG08 PRG09 PRG01 PRG02 PRG03 PRG04 PRG06 PRG07 PRG08 PRG09 PRG09 PRG01 PRG03 PRG01 PRG02 PRG03 PRG04 PRG06 PRG07 PRG08 PRG09 PRG05 PRG01 PRG03 PRG01 PRG02 PRG03 PRG05 PRG06 PRG07 PRG08 PRG09 PRG05 PRG03 PRG03 PRG09

Table 18 (Page 3 of 3). Functions, Operation Codes, and Features of RPG/400 Sample Programs		
Specification Form	Function/Operation Code Description	Programs
Calculation	Operation codes: MOVE MOVEL MULT MVR OPEN ORXX OUT PARM PLIST READ READC READE REDPE RESET RETRN SCAN SETGT SETLL SETOF SETON SUB SUBST TAG TIME UNLCK UPDAT WRITE XFOOT Z-ADD Z-SUB	PRG01 PRG02 PRG03 PRG04 PRG05 PRG06 PRG09 PRG09 PRG06 PRG07 PRG02 PRG05 PRG02 PRG03 PRG05 PRG05 PRG05 PRG04 PRG03 PRG03 PRG05 PRG08 PRG03 PRG08 PRG05 PRG03 PRG03 PRG09 PRG03 PRG09 PRG07 PRG08 PRG01 PRG03 PRG06 PRG05 PRG01 PRG03 PRG04 PRG01 PRG03 PRG06 PRG07 PRG08 PRG06 PRG07 PRG08 PRG01 PRG02 PRG03 PRG04 PRG06 PRG07 PRG08 PRG09 PRG07
Output	Printer files Program-described Externally described Exception output	PRG06 PRG07 PRG09 PRG02 PRG09 PRG01 PRG03 PRG04 PRG05 PRG06 PRG07 PRG09
Other Features Matching Record	Structured programming techniques Level breaks SAA compatible Function keys Subfile processing External indicators Initialization subroutine	PRG01 PRG02 PRG03 PRG04 PRG05 PRG06 PRG07 PRG08 PRG09 PRG06 PRG09 PRG09 PRG01 PRG02 PRG03 PRG03 PRG04 PRG05 PRG08

Sample Programs Design

Table 19 is a list of the sample programs in the order they appear in this chapter.

<i>Table 19. List of Sample Programs</i>	
Program	Refer to
PRG01	Figure 172 on page 364
PRG02	Figure 176 on page 386
PRG03	Figure 181 on page 401
PRG05	Figure 186 on page 421
PRG09	Figure 188 on page 428
PRG06	Figure 194 on page 453
PRG07	Figure 197 on page 468
PRG08	Figure 200 on page 479
PRG04	Figure 201 on page 487

Database Design

The time reporting application consists of three master files, two transaction history files, and a data area control file. The design of each of the files is listed below:

Employee Master File

The employee master file contains information about employees enrolled in the time reporting system. Data elements include:

ACREC Active record code
EMPNO Employee number
ENAME Employee name
EMCAT Employee Category
EDEPT Employee department
ELOCN Employee location
EUSRI Employee USRID (user identification)
ENHRS Employee normal week hours
EPHRC Employee project hours current month
EPHRY Employee project hours year-to-date
EPHRP Employee project prior year
ENHRC Employee non-project hours current month
ENHRY Employee non-project hours year-to-date
ENHRP Employee non-project hours prior year.

Project Master File

The project master file contains information on projects that are used in the time reporting system. Data elements include:

ACREC Active record code
PRCDE Project code

PRDSC Project description
 PRRSP Project responsibility
 PRSTR Project start date
 PREND Project estimated end date
 PRCMP Project completion date
 PREST Project estimated total hours
 PRHRC Project hours current month
 PRHRY Project hours year-to-date
 PRHRP Project hours prior year.

Reason-Code Master File

The reason-code master file contains information on non-project-related tasks, such as statutory holidays and personal time off. Data elements include:

ACREC Active record code
 RSCDE Reason code
 RSDSC Reason-code description
 RSHRC Reason-code hours current month
 RSHRY Reason-code hours year-to-date
 RSHRP Reason-code hours prior year

Transaction History Files

The transaction history files contain detail information entered by the user in a time entry display. The weekly transaction file contains all entries for the current week. When weekly reports are produced, this file is rolled into the monthly file. Both files have identical layouts. Data elements include:

ACREC Active record code
 EMPNO Employee number
 EUSRI Employee USRID (user identification)
 ACDAT Actual date worked (optional)
 CWKDT Week ending date
 CMTDT Month ending date
 PRCDE Project code
 RSCDE Reason code
 EHWRK Hours worked
 TFRRN Transaction file relative record number.

Data Area Control File

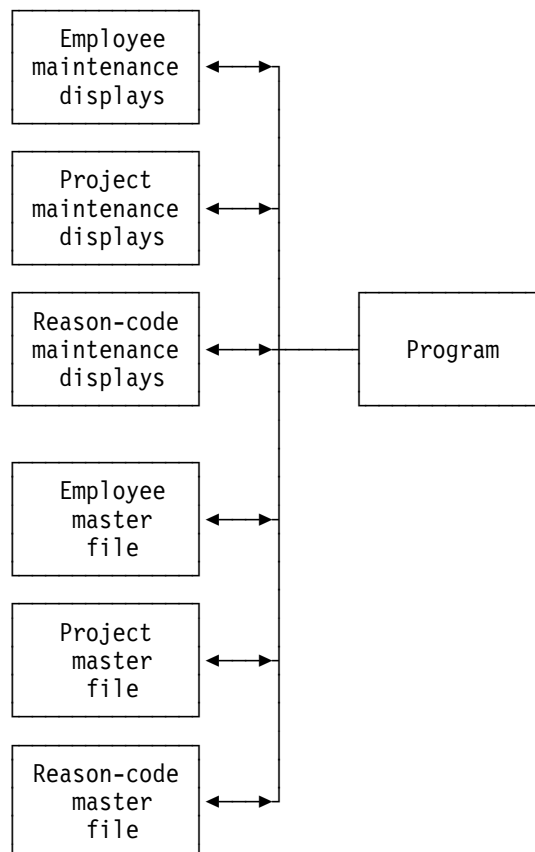
This data area control file contains control information for the time reporting system. Data elements include:

- ACREC Active record code
- CWKDT Week ending date
- CMTDT Month ending date
- CALLE All entries made flag.

Master File Maintenance

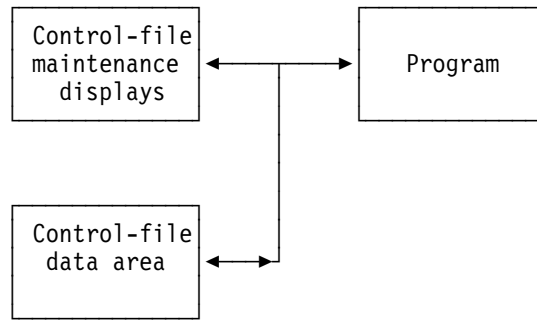
The master files are all maintained using workstation programs. All screens are designed using Screen Design Aid (SDA) and are externally defined. Flowcharts for the master file maintenance process follow:

The master file maintenance process allows additions, changes and deletions to the employee master file, project master file and reason-code master file.



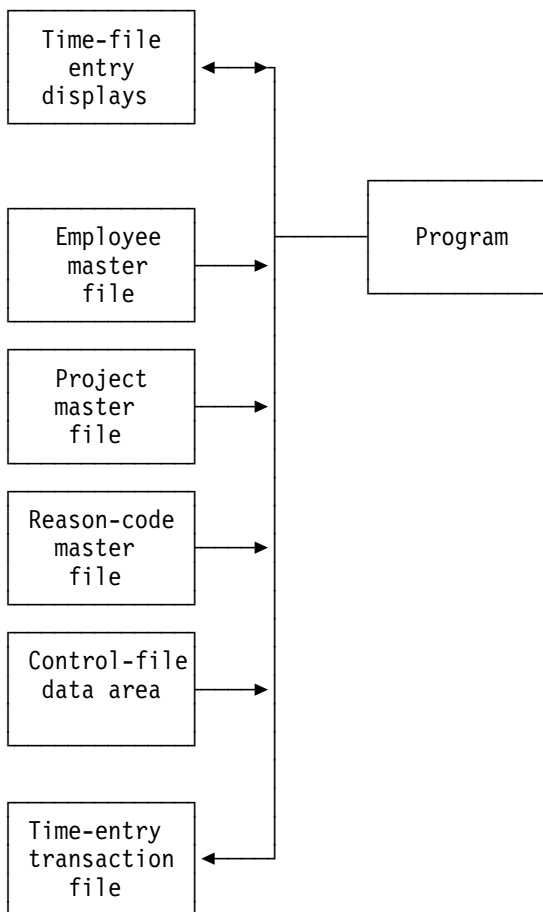
Data Area Control File Maintenance

The data area control file maintenance process allows update to the data area control file. A program-described workstation program maintains the data area control file.



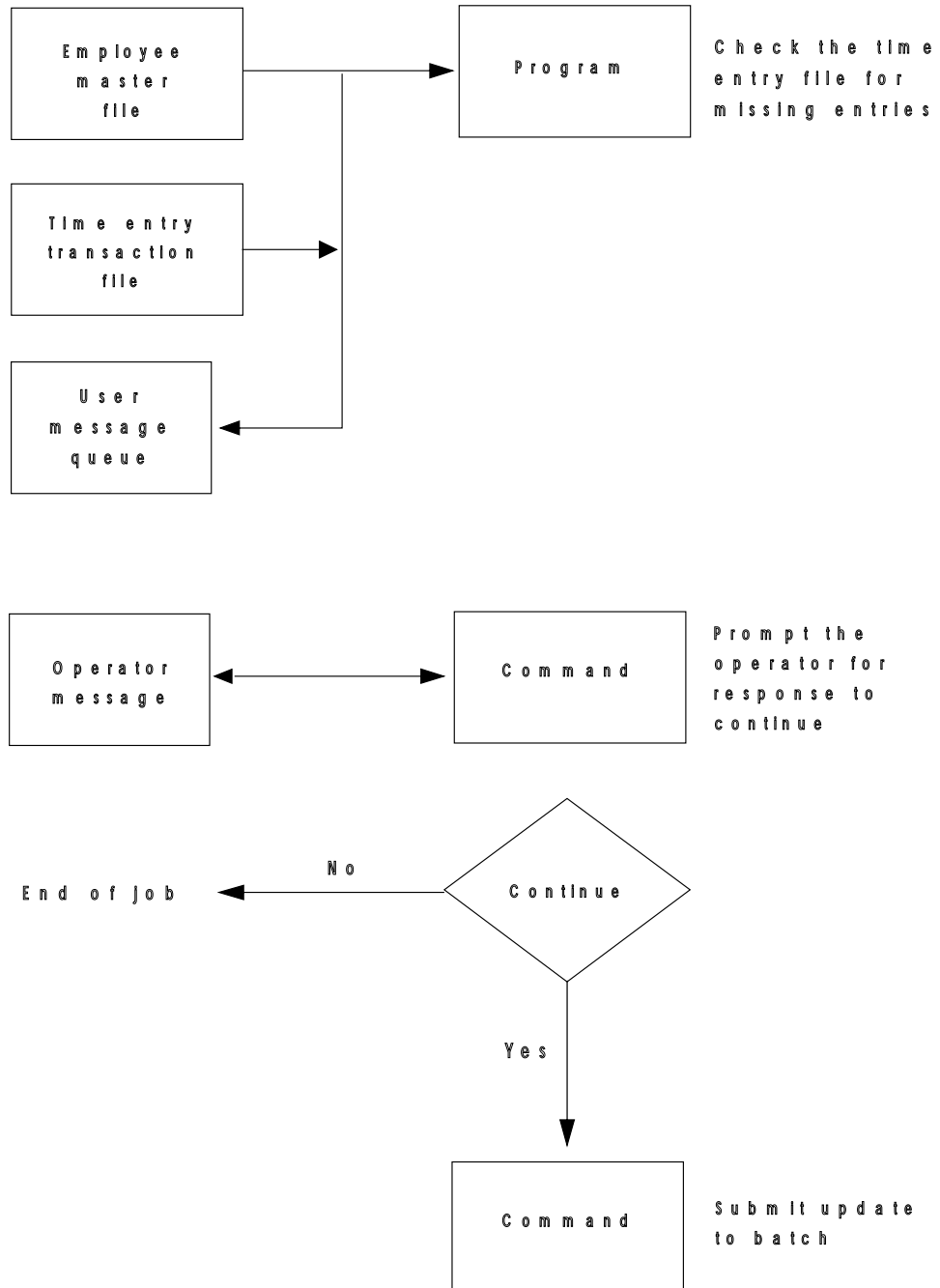
Time-File Entry

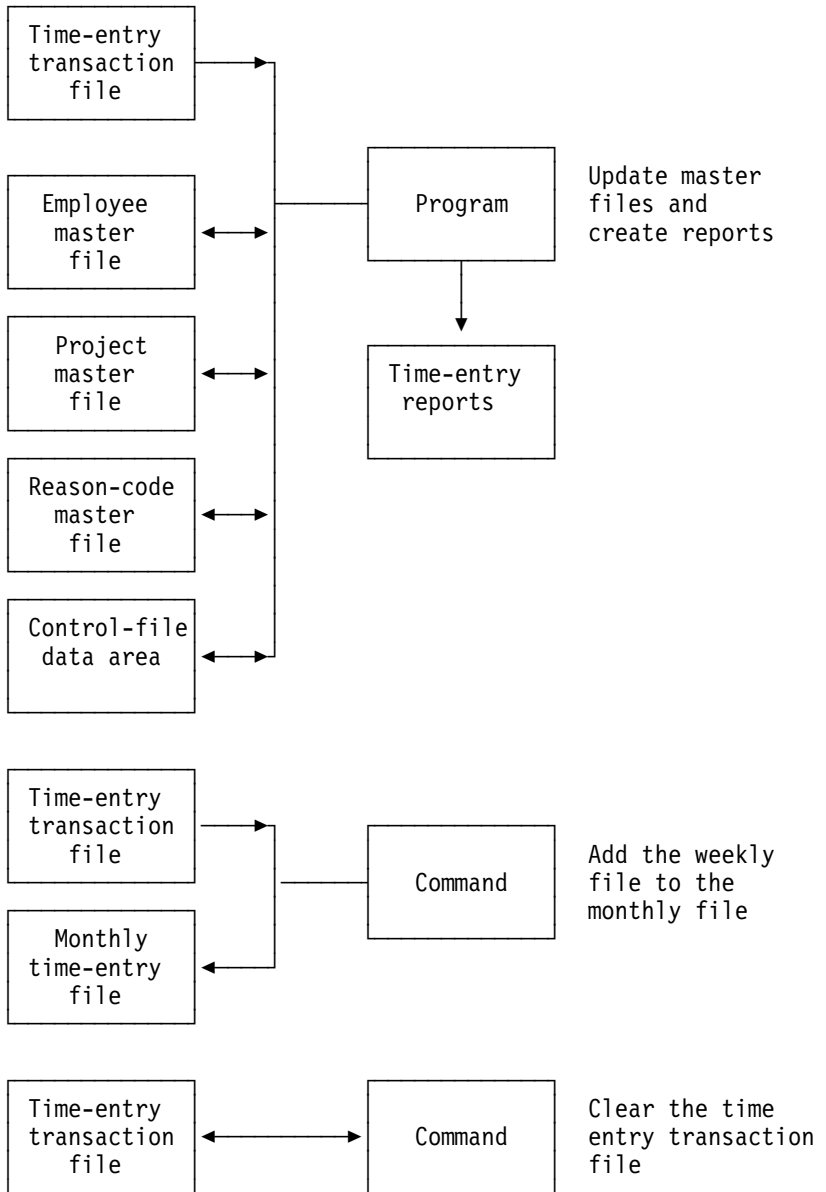
The time-file entry process is performed using workstation subfile processing. The screens are designed using SDA and are externally defined. Verification of data entered is done to the master files. The time-file entry process allows additions, changes and deletions to the transaction file with all fields maintainable. The data entry file is used for the weekly reporting and file update process. The time-file entry flowchart follows:



Weekly Time-File Update

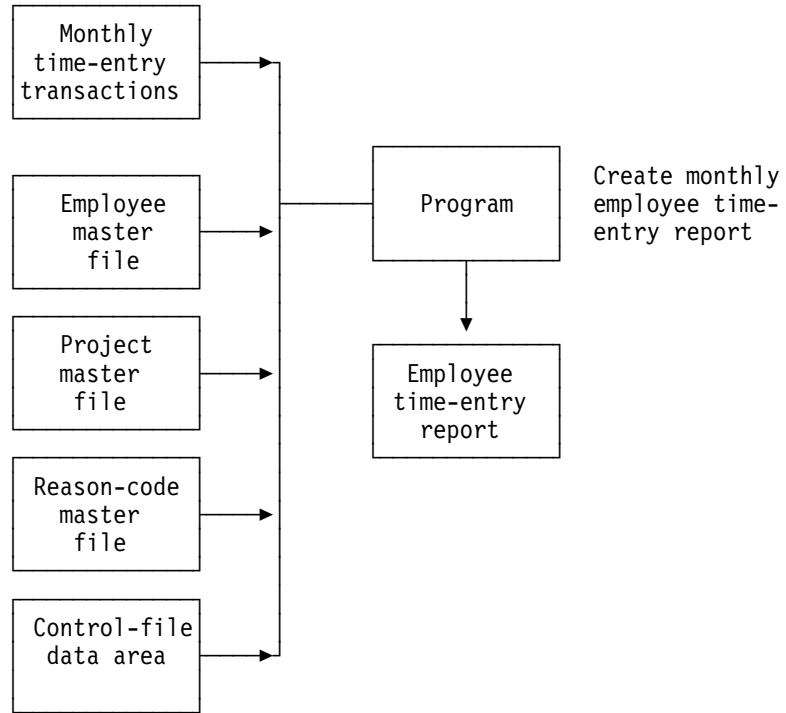
On a weekly basis the time-entry transaction file is processed to produce time-sheet reports and to update the master files with time-entry hours. The weekly time-file update process determines whether or not all required time entries have been made. If entries are missing, the employee is notified that his or her time entries are missing and the person who asked for the update is also notified. The person can cancel the update or continue. After all entries have been made or the person who asked for the update elects to continue, the reports are produced and the files updated. The weekly transaction file is added to the monthly file and then cleared. The following flowchart illustrates this process.

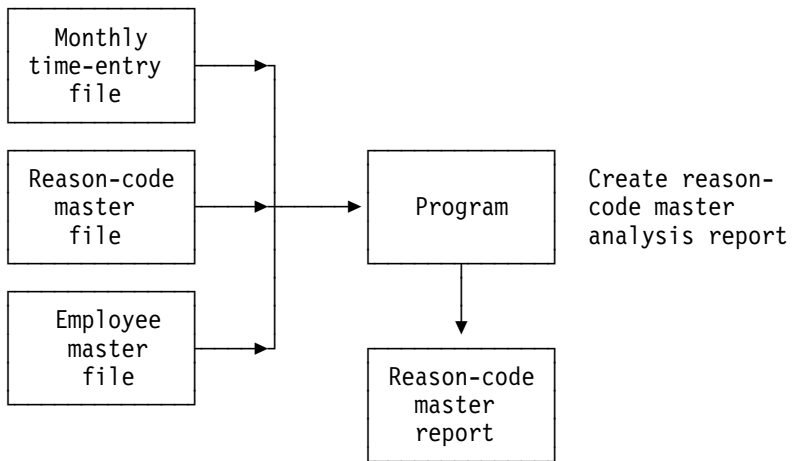
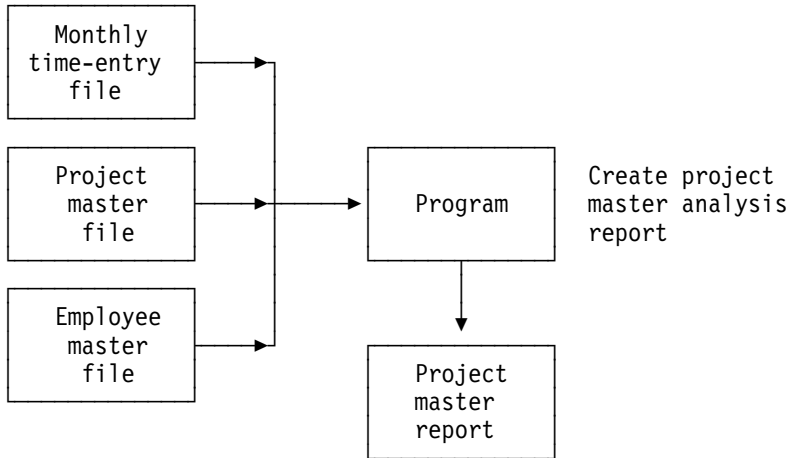


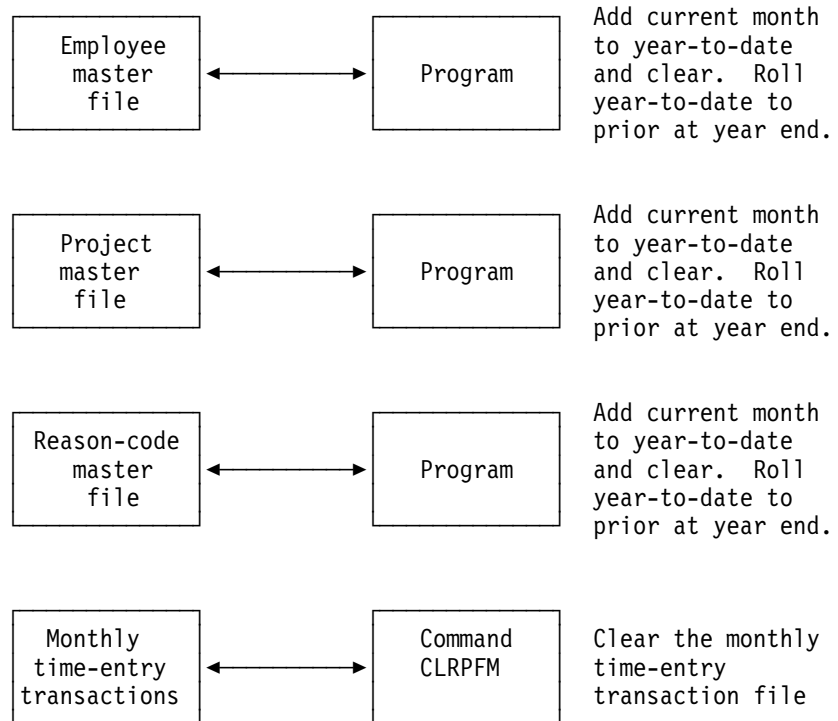


Monthly Time-Entry File Reporting and Update

After the final weekly run for the month, the monthly time-entry transaction file is processed to produce month end reports and to update the master files in preparation for new monthly data. The following flowchart illustrates this process.







Database Field Definition

This section contains the database field definition and field attributes for the time reporting system. A database reference file, REFMST, has been created that contains all detailed field definitions for all files. We could have defined the fields in each file as part of its own data description specifications (DDS), however, when a field is used in more than one file, the field ends up being defined multiple times. If a change is required to the definition of that field, it must be done for every occurrence. By defining a field reference file, we eliminate multiple definitions and simplify the task of redefining the field in the future. See Figure 154 on page 333.

Database Reference Master File - REFMST

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   REFMST - Database Reference Master File
A*   DESCRIPTION - A file containing field-level information for the
A*                   time reporting system files. This field-level
A*                   information is referenced when the specific
A*                   physical and logical files are created. The field-
A*                   level information is also referenced by SDA when
A*                   display formats are created.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A       R RCREF                TEXT('DATA BASE REFERENCE')
A* COMMON FIELDS
A       ACREC                   1          TEXT('ACTIVE RECORD CODE')
A                                   COLHDG('ACREC')
A       DATFL                   6  0      TEXT('DATE FIELD')
A                                   COLHDG('DATE' 'MDDYY')
A* EMPLOYEE MASTER RELATED FIELDS
A       EMPNO                   6  0      TEXT('EMPLOYEE NUMBER')
A                                   COLHDG('EMPLOYEE' 'NUMBER')
A       ENAME                   30        TEXT('EMPLOYEE NAME')
A                                   COLHDG('EMPLOYEE' 'NAME')
A       EMCAT                   1          TEXT('EMPLOYEE CATEGORY')
A                                   COLHDG('EMP' 'CAT')
A       EDEPT                   5          TEXT('EMPLOYEE DEPARTMENT')
A                                   COLHDG('EMPL' 'DEPT')
A       ELOCN                   30        TEXT('EMPLOYEE LOCATION')
A                                   COLHDG('EMPLOYEE' 'LOCATION')
A       EUSRI                   8          TEXT('EMPLOYEE USRID')
A                                   COLHDG('EMPLOYEE' 'USRID')
A       ENHRS                   3  1      TEXT('EMPLOYEE NORMAL WEEK HOURS')
A                                   COLHDG('NORMAL' 'WK HRS')
A       EPHRC                   5  1      TEXT('PROJECT HOURS CURRENT MONTH')
A                                   COLHDG('PRJ HRS' 'CUR MTH')
A       EPNRC                   5  1      TEXT('NON PROJECT HOURS CURR MONTH')
A                                   COLHDG('NON PRJ HRS' 'CUR MTH')

```

Figure 154 (Part 1 of 3). Database Reference Master File

Technical Design

```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A          EPHRY          7 1          TEXT('PROJECT HOURS YEAR TO DATE')
A          COLHDG('PRJ HRS' 'YTD')
A          EPHRP          7 1          TEXT('PROJECT HOURS PRIOR YEAR')
A          COLHDG('PRJ HRS' 'PRIOR YR')
A          EPNRY          7 1          TEXT('NON PROJECT HOURS YTD')
A          COLHDG('NON PRJ' 'HRS YTD')
A          EPNRP          7 1          TEXT('NON PROJECT HOURS PRIOR YEAR')
A          COLHDG('NON PRJ HRS' 'PRIOR YR')
A          EHWRK          5 1          TEXT('EMPLOYEE HOURS WORKED')
A          COLHDG('EMP HRS' 'WORKED')
A* PROJECT MASTER RELATED FIELDS
A          PRCDE          8          TEXT('PROJECT CODE')
A          COLHDG('PROJECT' 'CODE')
A          PRDSC          50         TEXT('PROJECT DESCRIPTION')
A          COLHDG('PROJECT' 'DESCRIPTION')
A          PRRSP          30         TEXT('PROJECT RESPONSIBILITY')
A          COLHDG('PROJECT' 'RESPONSIBILITY')
A          PRSTR          R          REFFLD(DATFL)
A          TEXT('PROJECT START DATE')
A          COLHDG('PRJ START' 'DATE')
A          PREND          R          REFFLD(DATFL)
A          TEXT('PROJECT ESTIMATED END DATE')
A          COLHDG('PRJ EST' 'END DATE')
A          PRCMP          R          REFFLD(DATFL)
A          TEXT('PROJECT COMPLETION DATE')
A          COLHDG('PRJ CMP' 'DATE')
A          PREST          9 1          TEXT('PROJECT ESTIMATED TOTAL HRS')
A          COLHDG('PRJ EST' 'TOT HRS')
A          PRHRC          7 1          TEXT('PROJECT HOURS CURRENT MONTH')

```

Figure 154 (Part 2 of 3). Database Reference Master File


```

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A          COLHDG('PRJ HRS' 'CUR MTH')
A          PRHRY          9  1      TEXT('PROJECT HOURS YEAR TO DATE')
A          COLHDG('PRJ HRS' 'YTD')
A          PRHRP          9  1      TEXT('PROJECT HOURS PRIOR YEAR')
A          COLHDG('PRJ HRS' 'PRIOR YR')
A* REASON CODE MASTER RELATED FIELDS
A          RSCDE          8          TEXT('REASON CODE')
A          COLHDG('REASON' 'CODE')
A          RSDSC          50         TEXT('REASON CODE DESCRIPTION')
A          COLHDG('REASON CODE' 'DESCRIPTION')
A          RSHRC          7  1      TEXT('REASON CODE HRS CURR MONTH')
A          COLHDG('RSN CDE HRS' 'CUR MTH')
A          RSHRY          9  1      TEXT('REASON CODE HRS YEAR TO DATE')
A          COLHDG('RSN CDE' 'HRS YTD')
A          RSHRP          9  1      TEXT('REASON CODE HOURS PRIOR YEAR')
A          COLHDG('RSN CDE HRS' 'PRIOR YR')
A* CONTROL FILE RELATED FIELDS
A          CTCDE          6          TEXT('CONTROL RECORD CODE')
A          COLHDG('CTL REC' 'CODE')
A          CWKDT          6S 0      TEXT('WEEK ENDING DATE')
A          COLHDG('WEEK END' 'DATE')
A          CMTDT          6S 0      TEXT('MONTH ENDING DATE')
A          COLHDG('MTH END' 'DATE')
A          CALLE          1          TEXT('ALL ENTRIES MADE FLAG')
A          COLHDG('ENTRIES' 'FLAG')

```

Figure 154 (Part 3 of 3). Database Reference Master File

Data Area Control File - CTLFIL

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   CTLFIL - Data Area Control File
A*   DESCRIPTION - A data area control file containing control-level
A*                   information for the time reporting system. The
A*                   data area contains one record format.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++*****
A                   REF(REFMST)
A           R RCCTL           TEXT('CONTROL FILE')
A           CTCDE           R
A           CWKDT           R
A           CMTDT           R
A           CALLE           R
A           K CTCDE
```

Figure 155. Data Area Control File

Employee Master File - EMPMST

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   EMPMST - Employee Master File
A*   DESCRIPTION - A file containing one record for each employee
A*                   enrolled in the time reporting system. Current
A*                   month, year-to-date, prior year project, and
A*                   non-project-related activity are maintained.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A
A
A           R RCEMP
A           ACREC      R
A           EMPNO      R
A           ENAME      R
A           EMCAT      R
A           EDEPT      R
A           ELOCN      R
A           EUSRI      R
A           ENHRS      R
A           EPHRC      R
A           EPHRY      R
A           EPHRP      R
A           EPNRC      R
A           EPNRY      R
A           EPNRP      R
A           K EMPNO

```

Figure 156. Employee Master File

Project Master File - PRJMST

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   PRJMST - Project Master File
A*   DESCRIPTION - A file containing information related to project
A*                   activity. Current month, year-to-date, and prior
A*                   year activity are maintained. One record exists
A*                   for each project code.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A                                     UNIQUE
A                                     REF(REFMST)
A           R RCPRJ                   TEXT('PROJECT MASTER')
A           ACREC                     R
A           PRCDE                     R
A           PRDSC                     R
A           PRRSP                     R
A           PRSTR                     R
A           PREND                     R
A           PRCMP                     R
A           PREST                     R
A           PRHRC                     R
A           PRHRY                     R
A           PRHRP                     R
A           K PRCDE
```

Figure 157. Project Master File

Reason-Code Master File - RSNMST

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   RSNMST - Reason-Code Master File
A*   DESCRIPTION - A file containing information related to non-project
A*                   activity. Current month, year-to-date, and prior
A*                   year activity are maintained. One record exists
A*                   for each reason code.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A                   UNIQUE
A                   REF(REFMST)
A           R RCRSN           TEXT('REASON CODE MASTER')
A           ACREC           R
A           RSCDE           R
A           RSDSC           R
A           RSHRC           R
A           RSHRY           R
A           RSHRP           R
A           K RSCDE

```

Figure 158. Reason-Code Master File

Weekly Transaction Entry File - TRWEEK

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   TRWEEK - Weekly Transaction Entry File
A*   DESCRIPTION - A file containing all entries made to the time
A*                   reporting system for the week.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A                   REF(REFMST)
A           R RCWEEK           TEXT('TRANSACTION ENTRY WEEKLY')
A           ACREC             R
A           EMPNO             R
A           EUSRI             R
A           ACDAT             6S 0
A           CWKDT             R
A           CMTDT             R
A           PRCDE             R
A           RSCDE             R
A           EHWRK             R
A           TFRRN             3 0

A*****
A*   TRWEEKL - Logical View of Weekly Transaction Entry File
A*   DESCRIPTION - The transaction entry program uses this file to
A*                   allow redisplay of existing employee entries and
A*                   update records added or changed in the subfile
A*                   entry.
A*****
A.....T.Name+++++.Len++TDpB.....Functions+++++
A           R RCWEEK           PFILE(TRWEEK)
A           K EMPNO
A           K TFRRN

```

Figure 159. Weekly Transaction Entry File

Monthly Transaction Entry File - TRMNTH

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   TRMNTH - Monthly Transaction Entry File
A*   DESCRIPTION - A file containing all entries made to the time
A*                   reporting system for the month.
A*****
A.....T.Name+++++RLen++TDpB.....Functions+++++
A                                     REF(REFMST)
A           R RCMNTH                   TEXT('TRANSACTION ENTRY MONTHLY')
A           ACREC                       R
A           EMPNO                       R
A           EUSRI                       R
A           CWKDT                       R
A           CMTDT                       R
A           PRCDE                       R
A           RSCDE                       R
A           EHWRK                       R
```

```
A*****
A*   TRMNTHL - Logical View of Monthly Transaction Entry File
A*   DESCRIPTION - This file is used by the time-entry employee
A*                   monthly reporting system.
A*****
A.....T.Name+++++.Len++TDpB.....Functions+++++
A           R RCMNTH                   PFILE(TRMNTH)
A           K CWKDT
A           K EMPNO
```

Figure 160 (Part 1 of 2). Monthly Transaction Entry File

Technical Design

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   TRMNTHR - Logical View of Monthly Transaction Entry File
A*   DESCRIPTION - This file is used by the time-entry project
A*                   monthly reporting system.
A*****
A.....T.Name+++++.Len++TDpB.....Functions+++++
A      R RCMNTH                PFILE(TRMNTH)
A      K PRCDE
A      K CWKDT
A      K EMPNO
A      O PRCDE                COMP(EQ '      ')

A*****
A*   TRMNTHN - Logical View of Monthly Transaction Entry File
A*   DESCRIPTION - This file is used by the time-entry reason-code
A*                   monthly reporting system.
A*****
A      R RCMNTH                PFILE(TRMNTH)
A      K RSCDE
A      K CWKDT
A      K EMPNO
A      O RSCDE                COMP(EQ '      ')
```

Figure 160 (Part 2 of 2). Monthly Transaction Entry File

Time Reporting Menu Design

Figure 161 shows the Time Reporting System Main Menu. The Main Menu allows you to perform file maintenance, control-file maintenance, transaction entry, weekly update, and monthly update. See Figure 162 on page 344 for the DDS for the TMENU.

Each menu option is described in detail in the remainder of this chapter. The Main Menu is repeated for each option, and the option being described is highlighted. An explanation of each option includes the control-level program called, the RPG/400 program called, or the command processed.

```

TMENU                               Time Reporting System
                                     Main Menu

      1. Master file maintenance      (PRG01)
      2. Control file maintenance     (PRG02)
      3. Time file transaction entry  (PRG03)
      4. Weekly time file update      (PROC1)
      5. Monthly time file update & reporting (PROC3)

      8. Display messages              (DSPMSG)
      9. Sign off                      (SIGNOFF)

Selection or command
====>
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel
F13=User support  F16=System main menu

```

Figure 161. Time Reporting System Main Menu Layout

Time Reporting Menu Design

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*  TMENU - Time Reporting System Main Menu Data Descriptions
A*  DESCRIPTION - A display file describing the formats that the
A*                  program uses to allow workstation maintenance
A*                  of the time reporting system.
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          DSPSIZ(24 80 *DS3
A                    27 132 *DS4)
A          CHGINPDT
A          INDARA
A          PRINT(*LIBL/QSYSPRT)
A          R  TMENU
A          DSPMOD(*DS3)
A          LOCK
A          SLNO(01)
A          CLRL(*ALL)
A          ALWROL
A          CF03
A          HELP
A          HOME
A          HLPRTN
A          1  2'TMENU'
A          COLOR(BLU)

```

Figure 162 (Part 1 of 3). TMENU Data Description Specifications

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          019  2'Selection or command          -
A          '
A          5  7'1.  Master file maintenance'
A          6  7'2.  Control file maintenance'
A          7  7'3.  Time file transaction entry'
A          8  7'4.  Weekly time file update'
A          9  7'5.  Monthly time file update &-
A                   reporting'
A         12  7'8.  Display messages'
A         13  7'9.  Sign off'
A          1 28'Time Reporting System'
A          2 34'Main Menu'
A          5 63'(PRG01)'
A          6 63'(PRG02)'
A          7 63'(PRG03)'
A          8 63'(PROC1)'
A          9 63'(PROC3)'
A         12 63'(DSPMSG)'
A         13 63'(SIGNOFF)''
A*

```

Figure 162 (Part 2 of 3). TMENU Data Description Specifications

```

* ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
TMENUQQ,1
0001 CALL PGM(PRG01)
0002 CALL PGM(PRG02)
0003 CALL PGM(PRG03)
0004 CALL PGM(PROC1)
0005 CALL PGM(PROC3)
0008 DSPMSG
0009 SIGNOFF

```

Figure 162 (Part 3 of 3). TMENU Data Description Specifications

Note: The TMENUQQ,1 portion of the DDS above begins in column 1.

Master File Maintenance

You select option 1 (Master file maintenance) on the Time Reporting System Main Menu to perform additions, changes, or deletions in the employee master file, project master file, or reason-code master file. You make these changes to the master file before doing your time entry transactions. The time entry process verifies the data you enter against these three master files. Figure 163 shows the Time Reporting System Main Menu. Option 1 calls program PRG01 by using the CALL PGM(PRG01) command.

```
TMENU                                Time Reporting System
                                      Main Menu

    1. Master file maintenance                (PRG01)
    2. Control file maintenance           (PRG02)
    3. Time file transaction entry        (PRG03)
    4. Weekly time file update            (PROC1)
    5. Monthly time file update & reporting (PROC3)

    8. Display messages                   (DSPMSG)
    9. Sign off                           (SIGNOFF)

Selection or command
====>
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel
F13=User support  F16=System main menu
```

Figure 163. Time Reporting System Main Menu

Employee Master Maintenance - EMPMNT Format

Figure 166 shows the display for the Employee Master Maintenance entry.

```

PRG01                                Time Reporting System                MM/DD/YY
                                      Employee Master Maintenance          TT:TT:TT

Number 666666 Name BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB 1
Category B 2
Department BBBBB 3
Location BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB 4
USRID BBBBBBB 5
Normal week hours 999- 6

                                      Time Reporting History

Project Related Current Month Year To Date Prior Year 7
Non Project Related 66666 6666666 6666666 6666666

0000000000000000000000000000000000000000000000000000000 8

F3-End of Job      F4-Maintenance Selection      F5-Employee Selection
    
```

Figure 166. Employee Master Maintenance - EMPMNT Format

The current values from the employee master are displayed on a change request or delete request. All fields are blank on an add request. On a delete request, all fields are protected. You can type the information for employee name **1**, category **2**, department **3**, location **4**, user identification **5**, and normal week hours **6**. The time reporting history **7** is not maintainable.

When you press Enter, the employee master file is updated and the employee maintenance selection format EMPSEL appears. You can press F3 to end the job, F4 to return to the maintenance selection format SELECT, or F5 to return to the employee maintenance selection format EMPSEL. The program does not update the employee master file when you use these function keys. The program does not validate data entered on this format. If an Add is requested and the employee master record was previously deleted, the program displays a warning message **8**.

Possible error message:

Warning - Record was previously deleted.

Project Master Maintenance - PRJMNT Format

Figure 168 shows the display for the Project Master Maintenance entry.

```

PRG01                                Time Reporting System                MM/DD/YY
                                     Project Master Maintenance          TT:TT:TT

                                     Project Code  00000000                1
Description BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
                                     2
Responsibility  BBBB...BBBBB...BBBBB...BBBBB...BBBBB...BBBBB...
Project Start Date  999999-                3
Project Estimated End Date  999999-        4
Project Completion Date  999999-        5
Project Estimated Hours  99999999-        6

                                     Project History
Current   Year To   Prior
Month    Date      Year
6666666  66666666  66666666          7

0000000000000000000000000000000000000000000000000000000000000000  8

F3-End of Job    F4-Maintenance Selection    F6-Project Code Selection
    
```

Figure 168. Project Master Maintenance - PRJMNT Format

The current values from the project master are displayed on a change request or delete request. All fields are blank on an add request. On a delete request, all fields are protected. You can type information for project description **1**, responsibility **2**, start date **3**, estimated end date **4**, completion date **5**, and estimated hours **6**. The time reporting history **7** is not maintainable.

When you press Enter, the project master file is updated and the project maintenance selection format PRJSEL appears. You can press F3 to end the job, F4 to return to the maintenance selection format SELECT, or F6 to return to the project maintenance selection format PRJSEL. The program does not update the project master file if you use these function keys. The program does not validate any data entered on this format. If an Add is requested and the project master record was previously deleted, the program displays a warning message **8**.

Possible error message:

Warning - Record was previously deleted.

Master File Maintenance Data Descriptions - PRG01FM

Figure 171 on page 355 shows the DDS for the PRG01FM Master File Maintenance display file. There are seven record formats, identified by R in position 17 followed by the format name in positions 19 through 28. The following keywords have been used:

ALARM	Activates the audible alarm.
BLINK	Blinks the cursor.
CAnn	Makes the function key specified in the keyword available for use.
DATE	Displays the current job date as a constant.
DSPATR	Specifies a display attribute for the field.
DSPSIZ	Specifies the display size to which the program can open this file.
EDTCDE	Specifies editing on an output capable numeric field.
INDARA	Removes option and response indicators from the buffer and places them in a 99-byte separate indicator area.
REFFLD	References the attributes of a previously defined field.
TIME	Displays the current system time as a constant.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* PRG01FM - Master File Maintenance Data Descriptions
A* DESCRIPTION - A display file describing the formats that the
A*                RPG/400 program PRG01 uses to allow work station
A*                maintenance of the following time reporting master
A*                files: EMPMST - Employee master file
A*                PRJMST - Project master file
A*                RSNMST - Reason code master file
A*****
A* The following code contains keywords that describe the overall
A* display file.
A*
AAN01N02N03T.Name+++++Rlen++TDpBlinPosFunctions+++++*****
A                DSPSIZ(24 80 *DS3)
A                PRINT
A                INDARA
A                CA03(03 'end of job')
A                CA04(04 'return to maintenance sele-
A                ction')
A                R SELECT
A*
A* The SELECT format describes the literals and fields that you
A* use to enter the maintenance selection code to determine which
A* time reporting master file you want to maintain.
A*
A                BLINK
A 60                ALARM
A                2 5'PRG01'
A                2 30'Time Reporting System'
A                2 70DATE
A                EDTCDE(Y)
A                3 30'Maintenance Selection'
A                3 70TIME
A                6 14'Enter an X beside the application -
A                you want to maintain'

```

Figure 171 (Part 1 of 9). Master File Maintenance Data Descriptions - PRG01FM

PRG01FM (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A      EMPAPL      1A B 9 25
A      9 28'Employee Master Maintenance'
A      PRJAPL      1A B 10 25
A      10 28'Project Master Maintenance'
A      RSNAPL      1A B 11 25
A      11 28'Reason Code Master Maintenance'
A      EMESS      50A 0 21 16
A 60      DSPATR(HI)
A      23 7'F3-End of Job'
A      R EMPSEL
A*
A* The EMPSEL format describes the literals and fields that you
A* use to enter the employee number and the maintenance action code
A* for selecting an employee master record.
A*
A      BLINK
A 60      ALARM
A      2 5'PRG01'
A      2 30'Time Reporting System'
A      2 70DATE
A      EDTCDE(Y)
A      3 27'Employee Master Maintenance'
A      3 70TIME
A      6 27'Employee Number'
A      EMPNO      R      B 6 44REFFLD(RCEMP/EMPNO *LIBL/EMPMST)
A      8 27'Action Code'
A      ACODE      1A B 8 44
A      8 48'A-Add'
A      9 48'C-Change'
A      10 48'D-Delete'

```

Figure 171 (Part 2 of 9). Master File Maintenance Data Descriptions - PRG01FM

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A          EMESS          50A  0 21 16
A 60          DSPATR(HI)
A          23  7'F3-End of Job'
A          23 25'F4-Maintenance Selection'
A          R EMPMNT
A*
A* The EMPMNT format describes the literals and fields that you
A* use to enter employee master maintenance.
A*
A          BLINK
A 61          ALARM
A          CA05(05 'return to employee selecti-
A          on')
A          2  5'PRG01'
A          2 30'Time Reporting System'
A          2 70DATE
A          EDTCDE(Y)
A          3 70TIME
A          3 27'Employee Master Maintenance'
A          5 14'Number'
A          EMPNO      R      0  5 22REFFLD(RCEMP/EMPNO *LIBL/EMPMST)
A          5 31'Name'
A          ENAME      R      B  5 37REFFLD(RCEMP/ENAME *LIBL/EMPMST)
A 90          DSPATR(PR)
A          7 18'Category'
A          EMCAT      R      B  7 37REFFLD(RCEMP/EMCAT *LIBL/EMPMST)
A 90          DSPATR(PR)
A          8 18'Department'
A          EDEPT      R      B  8 37REFFLD(RCEMP/EDEPT *LIBL/EMPMST)
A 90          DSPATR(PR)
A          9 18'Location'
A          ELOCN      R      B  9 37REFFLD(RCEMP/ELOCN *LIBL/EMPMST)

```

Figure 171 (Part 3 of 9). Master File Maintenance Data Descriptions - PRG01FM

PRG01FM (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A 90                                DSPATR(PR)
A                                10 18'USRID'
A                                B 10 37REFFLD(RCEMP/EUSRI *LIBL/EMPMST)
A 90                                DSPATR(PR)
A                                11 18'Normal week hours'
A                                B 11 37REFFLD(RCEMP/ENHRS *LIBL/EMPMST)
A 90                                DSPATR(PR)
A                                13 30'Time Reporting History'
A                                15 40'Current Year To Prior'
A                                16 40' Month      Date      Year'
A                                17 19'Project Related'
A                                EPHRC      R      0 17 41REFFLD(RCEMP/EPHRC *LIBL/EMPMST)
A                                EPHRY      R      0 17 49REFFLD(RCEMP/EPHRY *LIBL/EMPMST)
A                                EPHRP      R      0 17 58REFFLD(RCEMP/EPHRP *LIBL/EMPMST)
A                                18 19'Non Project Related'
A                                EPNRC      R      0 18 41REFFLD(RCEMP/EPNRC *LIBL/EMPMST)
A                                EPNRY      R      0 18 49REFFLD(RCEMP/EPNRY *LIBL/EMPMST)
A                                EPNRP      R      0 18 58REFFLD(RCEMP/EPNRP *LIBL/EMPMST)
A                                EMESS      50 0 21 16
A 61                                DSPATR(HI)
A                                23 7'F3-End of Job'
A                                23 25'F4-Maintenance Selection'
A                                23 55'F5-Employee Selection'
A                                11 44'(eg. 40.0 enter 400)'

```

Figure 171 (Part 4 of 9). Master File Maintenance Data Descriptions - PRG01FM


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A          R PRJSEL
A*
A* The PRJSEL format describes the literals and fields that you use
A* to enter the project code and the maintenance action code for
A* selecting a project master record.
A*
A          BLINK
A 60          ALARM
A          2 5'PRG01'
A          2 30'Time Reporting System'
A          2 70DATE
A          EDTCDE(Y)
A          3 27'Project Master Maintenance'
A          3 70TIME
A          6 27'Project Code'
A          PRCDE      R      B 6 44REFFLD(RCPRJ/PRCDE *LIBL/PRJMST)
A          8 27'Action Code'
A          ACODE      1A  B 8 44
A          8 48'A-Add'
A          9 48'C-Change'
A          10 48'D-Delete'
A          EMESS      50  0 21 16
A 60          DSPATR(HI)
A          23 7'F3-End of Job'
A          23 25'F4-Maintenance Selection'

```

Figure 171 (Part 5 of 9). Master File Maintenance Data Descriptions - PRG01FM

PRG01FM (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A          R PRJMNT
A*
A* The PRJMNT format describes the literals and fields that you
A* use to enter project master maintenance.
A*
A          BLINK
A 61      ALARM
A          CA06(06 'return to project selectio-
A          n')
A          2 5'PRG01'
A          2 30'Time Reporting System'
A          2 70DATE
A          EDTCDE(Y)
A          3 70TIME
A          3 27'Project Master Maintenance'
A          5 29'Project Code'
A          PRCDE      R      0 5 43REFFLD(RCPRJ/PRCDE *LIBL/PRJMST)
A          7 10'Description'
A          PRDSC      R      B 7 23REFFLD(RCPRJ/PRDSC *LIBL/PRJMST)
A 90      DSPATR(PR)
A          9 13'Responsibility'
A          PRRSP      R      B 9 41REFFLD(RCPRJ/PRRSP *LIBL/PRJMST)
A 90      DSPATR(PR)
A          10 13'Project Start Date'
A          PRSTR      R      B 10 41REFFLD(RCPRJ/PRSTR *LIBL/PRJMST)
A 90      DSPATR(PR)
A          10 53'(MMDDYY)'
A          11 13'Project Estimated End Date'
A          PREND      R      B 11 41REFFLD(RCPRJ/PREND *LIBL/PRJMST)
A 90      DSPATR(PR)
A          11 53'(MMDDYY)'

```

Figure 171 (Part 6 of 9). Master File Maintenance Data Descriptions - PRG01FM

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A          12 13'Project Completion Date'
A          PRCMP      R      B 12 41REFFLD(RCPRJ/PRCMP *LIBL/PRJMST)
A 90          DSPATR(PR)
A          12 53'(MMDDYY)'
A          13 13'Project Estimated Hours'
A          PREST      R      B 13 41REFFLD(RCPRJ/PREST *LIBL/PRJMST)
A 90          DSPATR(PR)
A          15 33'Project History'
A          17 27'Current'
A          17 37'Year To      Prior'
A          18 28'Month'
A          18 39'Date      Year'
A          PRHRC      R      0 19 27REFFLD(RCPRJ/PRHRC *LIBL/PRJMST)
A          PRHRY      R      0 19 36REFFLD(RCPRJ/PRHRY *LIBL/PRJMST)
A          PRHRP      R      0 19 47REFFLD(RCPRJ/PRHRP *LIBL/PRJMST)
A          EMESS          50  0 21 16
A 61          DSPATR(HI)
A          23  7'F3-End of Job'
A          23 25'F4-Maintenance Selection'
A          23 54'F6-Project Code Selection'

```

Figure 171 (Part 7 of 9). Master File Maintenance Data Descriptions - PRG01FM

PRG01FM (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A           R RSNSEL
A* The RSNSEL format describes the literals and fields that you
A* use to enter the reason code and maintenance action code for
A* selecting a reason code master record.
A           BLINK
A 60        ALARM
A           2 5'PRG01'
A           2 30'Time Reporting System'
A           2 70DATE
A           EDTCDE(Y)
A           3 26'Reason Code Master Maintenance'
A           3 70TIME
A           6 27'Reason Code'
A           RSCDE      R      B 6 44REFFLD(RCRSN/RSCDE *LIBL/RSNMST)
A           8 27'Action Code'
A           ACODE      1A  B 8 44
A           8 48'A-Add'
A           9 48'C-Change'
A           10 48'D-Delete'
A           EMESS      50  0 21 16
A 60        DSPATR(HI)
A           23 7'F3-End of Job'
A           23 25'F4-Maintenance Selection'
A           R RSNMNT
A* The RSNMNT format describes the literals and fields that you
A* use to enter reason code master maintenance.

```

Figure 171 (Part 8 of 9). Master File Maintenance Data Descriptions - PRG01FM

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++*****
A          BLINK
A 61      ALARM
A          CA07(07 'return to reason code sele-
A          ction')
A          2 5'PRG01'
A          2 30'Time Reporting System'
A          2 70DATE
A          EDTCDE(Y)
A          3 70TIME
A          3 27'Employee Master Maintenance'
A          5 30'Reason Code'
A          RSCDE R 0 5 43REFFLD(RCRSN/RSCDE *LIBL/RSNMST)
A          7 9'Description'
A          RSDSC R B 7 22REFFLD(RCRSN/RSDSC *LIBL/RSNMST)
A 90      DSPATR(PR)
A          9 31'Reason Code History'
A          11 26'Current'
A          11 36'Year To Prior'
A          12 27'Month'
A          12 38'Date Year'
A          RSHRC R 0 13 26REFFLD(RCRSN/RSHRC *LIBL/RSNMST)
A          RSHRY R 0 13 35REFFLD(RCRSN/RSHRY *LIBL/RSNMST)
A          RSHRP R 0 13 46REFFLD(RCRSN/RSHRP *LIBL/RSNMST)
A          EMESS 50 0 21 16
A 61      DSPATR(HI)
A          23 7'F3-End of Job'
A          23 25'F4-Maintenance Selection'
A          23 54'F7-Reason Code Selection'

```

Figure 171 (Part 9 of 9). Master File Maintenance Data Descriptions - PRG01FM

Master File Maintenance RPG/400 program - PRG01

Figure 172 shows the RPG/400 program PRG01. The program contains embedded comments to explain the logic flow and the use of RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG01 - Master File Maintenance RPG/400 Program
F* DESCRIPTION - Time reporting master file maintenance using
F*                externally described workstation processing.
F*****
F* INDICATORS USED:
F* 50 - No record found on CHAIN operation
F* 60 - General error condition
F* 90 - Protect display on delete request
F* KC - End of job requested
F* KD - Return to application selection
F* KE - Return to employee selection
F* KF - Return to project selection
F* KG - Return to reason code selection
F* LR - Last record
F*****
F* SUBROUTINES USED:
F* EDITSL - Edit application selection display (SELECT)
F* ACDESR - Edit action code for all maintenance requests
F*****
F* This program uses all externally described files. Files used
F* are: PRG01FM - Maintenance display file
F*       EMPMST - Employee master file
F*       PRJMST - Project master file
F*       RSNMST - Reason code master file
F*****
```

Figure 172 (Part 1 of 19). Sample RPG/400 Program - PRG01

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
```

```
FPRG01FM CF E WORKSTN
FEMPMST UF E K DISK A
FPRJMST UF E K DISK A
FRSNMST UF E K DISK A
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E* Compile time array containing error descriptions.
```

```
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++
E ERR 1 10 50
```

E*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
```

C* MAINLINE CALCULATIONS

```
C*****
```

C* This mainline routine controls display file processing and editing. Using the function keys described on each display format, you can transfer from one maintenance application to another. The action code you select on the selection formats determines if the program adds a new record to the file or updates an existing record in the file.

```
C*****
```

C* The program contains several TAG operations. The program will branch to these TAGs based on the action you take or function key you press on the various display formats. The BEGIN TAG provides a label to which the program branches if you press F4 on any of the maintenance formats.

C* The term 'housekeeping' used in this program refers to the initialization of indicators, temporary work fields, and display fields. Housekeeping ensures that information from previous input or calculation operations that may affect the operations the program performs next is not kept. Indicator 60 (*IN60) is set off, and blanks are moved to the SELECT format display fields as part of housekeeping.

Figure 172 (Part 2 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          BEGIN      TAG
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C          MOVE *BLANKS  EMPAPL
C          MOVE *BLANKS  PRJAPL
C          MOVE *BLANKS  RSNAPL
C*
C* The SELTAG TAG provides a label to which the program branches
C* if errors are found in the maintenance selection format SELECT.
C* The SELECT format is written to the work station using EXFMT.
C* The EXFMT causes a write to and a read from the display. If
C* you press F3 (*INKC = 1), the program branches to the END TAG.
C* If you do not press F3 (*INKC = 0), the program processes the
C* EDITSL subroutine to edit the SELECT format input.
C*
```

Figure 172 (Part 3 of 19). Sample RPG/400 Program - PRG01

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          SELTAG     TAG
C          EXFMTSELECT
C          *INKC      CABEQ'1'      END
C          EXSR EDITSL
C*
C* IF the general error indicator *IN60 is on (equal to 1), the
C* program branches back to the SELTAG.
C*
C          *IN60      CABEQ'1'      SELTAG
C*
C* At this point, the SELECT format has been verified and the program
C* displays the maintenance entry format for the application selected.
C* The application selection fields from the SELECT format are tested
C* and the program branches to the section specific to the application.
C* If EMPAPL (employee maintenance) equals X, the program branches to
C* label EMPTAG. If PRJAPL (project maintenance) equals X, the
C* program branches to label PRJTAG. If the previous two tests were
C* not successful, you chose reason code maintenance. The program
C* continues with the next operation.
C*
```

Figure 172 (Part 4 of 19). Sample RPG/400 Program - PRG01


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          EMPAPL    CABEQ'X'          EMPTAG
C          PRJAPL    CABEQ'X'          PRJTAG
C*
C*****
C* Reason Code Maintenance: The RSNTAG TAG provides a label to
C* which the program branches if you press F7 from the reason-code
C* maintenance entry format RSNMNT. The program branches to this
C* tag when the maintenance request completes successfully.
C* Housekeeping: Initialize general error indicator 60 and clear
C* RSNSEL format display fields.
C*
C          RSNTAG    TAG
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C          MOVE *BLANKS  RSCDE
C          MOVE *BLANKS  ACODE
C*
C* The program branches to the RSNERR TAG if errors are found
C* when editing the RSNSEL format input.
C* Housekeeping: Initialize RSNMNT format display fields.
C* The RSNSEL format is written using the EXFMT operation. When
C* you press Enter or a function key, the program continues with
C* the next operation. If you press F4 (*INKD = 1), the program
C* branches to the BEGIN TAG.
C*

```

Figure 172 (Part 5 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          RSNERR      TAG
C          MOVE *BLANKS  RSDSC
C          EXFMTRSSEL
C          *INKD       CABEQ'1'      BEGIN
C*
C* If you press F3 (*INKC = 1), the program branches to the END TAG.
C* If you do not press F3, the reason code master file is accessed
C* using the reason code (RSCDE) that you entered and the CHAIN
C* operation. If the record is not found, resulting indicator 50
C* (positions 54 and 55) is set on. The ACDESR subroutine is
C* processed to edit your request.
C*
C          *INKC       CABEQ'1'      END
C          RSCDE      CHAINRSNMST      50
C          EXSR ACDESR
C*
C* If editing processed by the ACDESR subroutine detects errors
C* in your request, general error indicator 60 is on and the
C* program branches back to the RSNERR TAG.
C*
C          *IN60      CABEQ'1'      RSNERR
C*
C* The RSNMNT format is written using the EXFMT operation. If you
C* press F4 (*INKD), the program branches back to the BEGIN TAG. If
C* you press F7 (*INKG), the program branches back to the RSNTAG TAG.
C*
C          EXFMTRSNMNT
C          *INKD      CABEQ'1'      BEGIN
C          *INKG      CABEQ'1'      RSNTAG
```

Figure 172 (Part 6 of 19). Sample RPG/400 Program - PRG01

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code performs the reason-code file update
C* operations. The code contains five levels of nested IF
C* statements. The beginning of each level is indicated by Bnn,
C* where nn is the nested level. The calculations in each level
C* are indicated by nn. The end of each level is indicated by Enn.
C* In each level, the ACREC (active record code) field is updated
C* to A for active or D for deleted.
C*
C* First level - if you press F3 for end of job, the program
C* processes the 01 level calculations and branches to the END TAG.
C* Second level - if you enter action code A, and the record
C* does not already exist (*IN50 = 1), the program WRITES the record.
C* Third level - if you enter action code A, and the record already
C* exists (*IN50 = 0) with an active record code of D (ACREC = D),
C* then the program updates the existing record.
C* Fourth level - if you enter action code D, the record is updated.
C* Fifth level - if you enter action code C, the record is updated.
C*

```

Figure 172 (Part 7 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *INKC      IFEQ '0'                      B01
C          ACODE      IFEQ 'A'                      B02
C          *IN50      ANDEQ'1'                      02
C                               MOVE 'A'            ACREC      02
C                               WRITERCRSN          02
C                               ELSE                02
C          ACODE      IFEQ 'A'                      B03
C          *IN50      ANDEQ'0'                      03
C          ACREC      ANDEQ'D'                      03
C                               MOVE 'A'            ACREC      03
C                               UPDATRCRSN          03
C                               ELSE                03
C          ACODE      IFEQ 'D'                      B04
C                               MOVE 'D'            ACREC      04
C                               UPDATRCRSN          04
C                               ELSE                04
C          ACODE      IFEQ 'C'                      B05
C                               UPDATRCRSN          05
C                               END                  E05
C                               END                  E04
C                               END                  E03
C                               END                  E02
C                               ELSE                01
C                               GOTO END              01
C                               END                  E01
C*
C* Your maintenance request is completed and the program branches
C* back to the RSNTAG TAG.
C*
C                               GOTO RSNTAG

```

Figure 172 (Part 8 of 19). Sample RPG/400 Program - PRG01

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* Employee-master maintenance routine performs the same steps as
C* done in the reason code routine. Refer to RSNTAG for further
C* explanation of the following processing steps.
C*****
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C          EMPTAG      TAG
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C          Z-ADD0        EMPNO
C          MOVE *BLANKS  ACODE
C          EMPERR      TAG
C          MOVE *BLANKS  ENAME
C          MOVE *BLANKS  EMCAT
C          MOVE *BLANKS  EDEPT
C          MOVE *BLANKS  ELOCN
C          MOVE *BLANKS  EUSRI
C          Z-ADD0        ENHRS
C* Display employee selection format
C          EXFMTEMPSEL
C          *INKD      CABEQ'1'      BEGIN
C* Access employee master to validate action code request
C          *INKC      CABEQ'1'      END
C          EMPNO      CHAINEMPMST          50
C          EXSR ACDESR
C          *IN60      CABEQ'1'      EMPERR
C* Display employee maintenance format
C          EXFMTEMPMNT
C          *INKD      CABEQ'1'      BEGIN
C          *INKE      CABEQ'1'      EMPTAG

```

Figure 172 (Part 9 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *INKC      IFEQ '0'                B01
C          ACODE      IFEQ 'A'                B02
C          *IN50      ANDEQ '1'              02
C          MOVE 'A'    ACREC                  02
C          WRITERCEMP                                02
C          ELSE                                02
C          ACODE      IFEQ 'A'                B03
C          *IN50      ANDEQ '0'              03
C          ACREC      ANDEQ 'D'              03
C          MOVE 'A'    ACREC                  03
C          UPDATRCEMP                                03
C          ELSE                                03
C          ACODE      IFEQ 'D'                B04
C          MOVE 'D'    ACREC                  04
C          UPDATRCEMP                                04
C          ELSE                                04
C          ACODE      IFEQ 'C'                B05
C          MOVE 'A'    ACREC                  05
C          UPDATRCEMP                                05
C          END                                E05
C          END                                E04
C          END                                E03
C          END                                E02
C          ELSE                                01
C          GOTO END                                01
C          END                                E01
C          GOTO EMPTAG

```

Figure 172 (Part 10 of 19). Sample RPG/400 Program - PRG01

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* Project-master maintenance routine performs the same steps as
C* in the reason code routine. Refer to RSNTAG for further
C* explanation of the following processing steps.
C*****
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C          PRJTAG      TAG
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C          MOVE *BLANKS  PRCDE
C          MOVE *BLANKS  ACODE
C          PRJERR      TAG
C          MOVE *BLANKS  PRDSC
C          MOVE *BLANKS  PRRSP
C          Z-ADD0       PRSTR
C          Z-ADD0       PREND
C          Z-ADD0       PRCMP
C          Z-ADD0       PREST
C* Display project selection format
C          EXFMTPRJSEL
C          *INKD      CABEQ'1'      BEGIN
C* Access project master to validate action code request
C          *INKC      CABEQ'1'      END
C          PRCDE      CHAINPRJMST          50
C          EXSR ACDESR
C          *IN60      CABEQ'1'      PRJERR
C* Display project maintenance format
C          EXFMTPRJMNT
C          *INKD      CABEQ'1'      BEGIN
C          *INKF      CABEQ'1'      PRJTAG

```

Figure 172 (Part 11 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C* Determine update mode and perform record add or update
C      *INKC      IFEQ '0'      B01
C      ACODE      IFEQ 'A'      B02
C      *IN50      ANDEQ'1'      02
C      MOVE 'A'      ACREC      02
C      WRITERCPRJ      02
C      ELSE      02
C      ACODE      IFEQ 'A'      B03
C      *IN50      ANDEQ'0'      03
C      ACREC      ANDEQ'D'      03
C      MOVE 'A'      ACREC      03
C      UPDATRCPRJ      03
C      ELSE      03
C      ACODE      IFEQ 'D'      B04
C      MOVE 'D'      ACREC      04
C      UPDATRCPRJ      04
C      ELSE      04
C      ACODE      IFEQ 'C'      B05
C      MOVE 'A'      ACREC      05
C      UPDATRCPRJ      05
C      END      E05
C      END      E04
C      END      E03
C      END      E02
C      ELSE      01
C      GOTO END      01
C      END      E01
C      GOTO PRJTAG
C* End of job requested. Control is passed to here when you press
C* F3 (*INKC). The last record indicator *INLR is set on and the
C* program ends.
C      END      TAG
C      MOVE '1'      *INLR

```

Figure 172 (Part 12 of 19). Sample RPG/400 Program - PRG01


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*  EDITSL subroutine verifies the time reporting application
C*  selection display input.
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          EDITSL    BEGSR
C*
C*  Housekeeping: The general error indicator *IN60 is set off and
C*  the error message field EMESS is set to blanks.
C          MOVE *BLANKS    EMESS    50
C          MOVE '0'        *IN60
C*
C*  The following IF AND OR combination checks the application
C*  selection fields to ensure that only one application has been
C*  selected.  If more than one is selected, the general error
C*  indicator *IN60 is set on (equal to 1) and the error message
C*  established by moving array element 2 (ERR,2) to the EMESS field.
C*
C          Z-ADD0          SELCNT  10
C          EMPAPL         IFEQ 'X'
C          ADD 1           SELCNT
C          END
C          PRJAPL         IFEQ 'X'
C          ADD 1           SELCNT
C          END
C          RSNAPL         IFEQ 'X'
C          ADD 1           SELCNT
C          END
C          SELCNT         IFGT 1
C          MOVE '1'        *IN60
C          MOVE ERR,2      EMESS
C          ELSE
C          MOVE '0'        *IN60
C          END

```

Figure 172 (Part 13 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following IF AND combination ensures that at least one
C* application is selected. The application selection fields are
C* checked and if they are all equal to ' ' (blank), *IN60 is set
C* on and array element 3 moved to the error message field.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C      EMPAPL   IFEQ ' '
C      PRJAPL   ANDEQ' '
C      RSNAPL   ANDEQ' '
C              MOVE '1'          *IN60
C              MOVE ERR,3        EMESS
C              END
C* The following code checks each application selection field to
C* ensure that it is either ' ' (blank) or equal to 'X'. If any
C* of the three selection fields contains a value other than ' '
C* or 'X', *IN60 is set on and array element 1 is moved to the
C* error message field.
C      EMPAPL   IFNE ' '
C      EMPAPL   ANDNE'X'
C              MOVE '1'          *IN60
C              MOVE ERR,1        EMESS
C              END
C      PRJAPL   IFNE ' '
C      PRJAPL   ANDNE'X'
C              MOVE '1'          *IN60
C              MOVE ERR,1        EMESS
C              END
C      RSNAPL   IFNE ' '
C      RSNAPL   ANDNE'X'
C              MOVE '1'          *IN60
C              MOVE ERR,1        EMESS
C              END
C      ENDSR
```

Figure 172 (Part 14 of 19). Sample RPG/400 Program - PRG01

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* ACDESR subroutine verifies the time reporting action codes for
C* all maintenance selections.
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C           ACDESR   BEGSR
C*
C* Housekeeping: The error indicators *IN60 and *IN61 are set off
C* and the error message field EMESS is set to blanks. Indicator
C* *IN90 is defined in the maintenance display formats to protect
C* the display on a delete request. It is set off here (equal to 0)
C* as part of housekeeping.
C*
C           MOVE *BLANKS   EMESS
C           MOVE '0'       *IN60
C           MOVE '0'       *IN61
C           MOVE '0'       *IN90
C*
C* The following compare and branch (CABEQ) statements perform two
C* functions. They determines the type of maintenance requested
C* and branches to the appropriate label, and they determine if the
C* maintenance code entered is incorrect. The CABEQ operation
C* checks the ACODE (action code) field for a value of 'A' (add)
C* and, if true, branches to the ADDCDE TAG. ACODE is also checked
C* for 'C' and sent to CHGCDE TAG and for 'D' and sent to DELCDE TAG.
C* If the ACODE field does not equal A, C, or D, *IN60 is set on and
C* array element 4 moved to the error message field. The program
C* then branches to the end of the subroutine ACDEND label on the
C* ENDSR statement.
C*

```

Figure 172 (Part 15 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          ACODE      CABEQ'A'      ADDCDE
C          ACODE      CABEQ'C'      CHGCDE
C          ACODE      CABEQ'D'      DELCDE
C*
C* Not valid action code
C*
C          MOVE '1'      *IN60
C          MOVE ERR,4    EMESS
C          GOTO ACEND
C*
C* The following code verifies the add request. Indicator *IN50
C* equals to '0' indicates the record is found on the CHAIN
C* operation. If the record already exists (*IN50 equals 0) and
C* the record status field ACREC for the record is 'A' for active,
C* *IN60 is set on, and array element 5 is moved to EMESS. If the
C* record already exists and the record status field ACREC is 'D'
C* for deleted, array element 6 is moved to EMESS. In the last
C* error, we do not set on the error indicator because error
C* message 6 is a warning error that is displayed on the
C* maintenance format indicating that the record has already been
C* deleted. You must then decide whether to reactivate the record
C* or to return to the selection display to change the request.
C*
```

Figure 172 (Part 16 of 19). Sample RPG/400 Program - PRG01

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          ADDCDE    TAG
C          *IN50     IFEQ '0'
C          ACREC     ANDEQ'A'
C                      MOVE '1'          *IN60
C                      MOVE ERR,5        EMESS
C                      ELSE
C          *IN50     IFEQ '0'
C          ACREC     ANDEQ'D'
C                      MOVE ERR,6        EMESS
C                      END
C                      END
C                      GOTO ACDEND

```

C*

C* The following code verifies the change request. If the first
C* check verifies that the record is not found on the CHAIN
C* operation (*IN50 equals 1), *IN60 is set on and array element is
C* moved to EMESS. The second check verifies that the record does
C* exist (*IN50 equals 0) but that the record status field ACREC
C* equals 'D' for deleted, *IN60 is set on and array element 8 is
C* moved to EMESS.

```

C          CHGCDE    TAG
C          *IN50     IFEQ '1'
C                      MOVE '1'          *IN60
C                      MOVE ERR,7        EMESS
C                      ELSE
C          *IN50     IFEQ '0'
C          ACREC     ANDEQ'D'
C                      MOVE '1'          *IN60
C                      MOVE ERR,8        EMESS
C                      END
C                      END
C                      GOTO ACDEND

```

Figure 172 (Part 17 of 19). Sample RPG/400 Program - PRG01

PRG01 (Master File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code verifies the delete request. When field
C* protect indicator *IN90 is first set on (equal to 1), changes to
C* existing data on a delete request are not allowed. When the
C* program checks that the record is not found on the CHAIN
C* operation (*IN50 equals 1), *IN60 is set on and array element 9
C* is moved to EMESS. A second check determines that the record
C* does exist (IN50 equals 0) but that the record status field
C* ACREC equals 'D' indicating it is already deleted, *IN60 is set
C* on and array element 10 is moved to EMESS.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          DELCDE      TAG
C          MOVE '1'          *IN90
C          *IN50      IFEQ '1'
C          MOVE '1'          *IN60
C          MOVE ERR,9      EMESS
C          ELSE
C          *IN50      IFEQ '0'
C          ACREC      ANDEQ'D'
C          MOVE '1'          *IN60
C          MOVE ERR,10      EMESS
C          END
C          END
C          ACDEND      ENDSR
```

Figure 172 (Part 18 of 19). Sample RPG/400 Program - PRG01

```

* ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
  0*
  0* The compile time array ERR is entered below. The array is
  0* preceded by "** " to denote the beginning of the array and
  0* begins in column 1 of the output specification.
  0*
** Array ERR - Error descriptions
MAINTENANCE SELECTION CODE NOT EQUAL TO "X"
MORE THAN ONE APPLICATION SELECTED FOR MAINTENANCE
NO APPLICATION SELECTED FOR MAINTENANCE
ACTION CODE NOT EQUAL TO "A", "C" OR "D"
ADD REQUESTED BUT RECORD ALREADY EXISTS IN FILE
WARNING - RECORD WAS PREVIOUSLY DELETED
CHANGE REQUESTED BUT RECORD DOES NOT EXIST
FLAGGED FOR DELETE BUT CHANGE REQUESTED
DELETE REQUESTED BUT RECORD DOES NOT EXIST
DELETE REQUESTED BUT RECORD ALREADY DELETED

```

Figure 172 (Part 19 of 19). Sample RPG/400 Program - PRG01

Control File Maintenance

You select option 2 (Control file maintenance) on the Time Reporting System Main Menu to change the week ending date, month ending date, or the all time entries made flag. You make the changes before running your weekly update and your monthly update. Figure 173 shows the Time Reporting System Main Menu. Option 2 calls program PRG02 by using the CALL PGM(PRG02) command.

```
TMENU                                Time Reporting System
                                      Main Menu

    1. Master file maintenance          (PRG01)
    2. Control file maintenance       (PRG02)
    3. Time file transaction entry      (PRG03)
    4. Weekly time file update          (PROC1)
    5. Monthly time file update & reporting (PROC3)

    8. Display messages                 (DSPMSG)
    9. Sign off                         (SIGNOFF)

Selection or command
====>
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel
F13=User support  F16=System main menu
```

Figure 173. Time Reporting System Main Menu

Control File Maintenance Data Descriptions - PRG02FM

Figure 175 on page 385 shows the DDS for the PRG02FM Control File Maintenance display file. The data descriptions describe the function and appearance of the display file formats. Comments have been included in the display file to describe the code.

The following keywords are used:

BLINK	Blinks the cursor.
CAnn	Specifies the function key, identified by nn, is available for use.
DSPATR	Specifies the display attributes for the field.
DSPSIZ	Specifies the display size to which the program can open the file.
EDTCDE	Specifies the edit code for an output-capable field.
INDARA	Removes option and response indicators from the buffer and places them in a 99-byte separate indicator area.
REFFLD	References the attributes of a previously defined field.
TIME	Displays the current system time as a constant field.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* This display file provides maintenance to the time reporting
A* control file.
A*****
AAN01N02N03T.Name+++++Rlen++TDpBLinPosFunctions+++++
A          DSPSIZ(24 80 *DS3)
A          PRINT
A          INDARA
A          CA03(03 'end of job')
A          R CTLMNT
A*
A* The CTLMNT format describes the layout of the control file
A* maintenance entry.
A*
A          CA12(12 'DATE WARNING - ACCEPT')
A          BLINK
A          2 5'PRG02'
A          2 30'Time Reporting System'
A          2 70DATE
A          EDTCDE(Y)
A          3 70TIME
A          3 23'Control File Data Area Maintenance'
A          6 17'Week Ending Date'
A          CWKDT      R      B 6 45REFFLD(RCCTL/CWKDT *LIBL/CTLFIL)
A          7 17'Month Ending Date'
A          CMTDT      R      B 7 45REFFLD(RCCTL/CMTDT *LIBL/CTLFIL)
A          8 17'All time entries made flag'
A          CALLE      R      B 8 45REFFLD(RCCTL/CALLE *LIBL/CTLFIL)
A          ERMESS      50A 0 21 16
A 60          DSPATR(HI)
A          23 7'F3-End of Job'
A          6 55'(MMDDYY)'
A          7 55'(MMDDYY)'
A          8 55'(Y or N)'

```

Figure 175. Control File Maintenance Data Descriptions - PRG02FM

Control File Maintenance RPG/400 Program - PRG02

Figure 176 shows the RPG/400 specifications for the control file maintenance program. Comments are included as part of the program to describe the various sections of the code.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG02 - Control File Maintenance RPG/400 Program
F* DESCRIPTION - Time reporting control file maintenance using
F*                program-described work station processing.
F*****
F* INDICATORS USED:
F* 01 - Control file maintenance display input
F* 50 - Leap year
F* 51 - Invalid date entered
F* 52 - Invalid time entry flag
F* 53 - Year in date entered does not equal current year
F* 60 - General error condition
F* KC - End of job requested
F* KL - Accept warning error
F* LR - Last record
F*****
F* SUBROUTINES USED:
F* EDITSR - Edit input fields from CTLMNT format
F* DATESR - Edit date format
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRG02FM CP F 100 WORKSTN
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E*
E* Compile time array containing error descriptions.
E ERR 1 3 50
E* Compile time arrays containing days per month for non-leap
E* year and leap year.
E ARM 12 12 2 0
E ARL 12 12 2 0
```

Figure 176 (Part 1 of 8). Sample RPG/400 Program - PRG02

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* Program-described display file input for control file maintenance.
I* Input fields are:
I*   C WEEK - Week ending date
I*   C MMTH - Month ending date
I*   C ENTR - All time entries made flag
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IPRG02FM NS 01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I           1 60C WEEK
I           7 120C MMTH
I           13 13 C ENTR
I*
I* The following named constant defines the record format name
I* for the WORKSTN file.
I*
I           'CTLMNT'           C           RECFMT
I*
I* Externally described control file data area
I*
IDname....NODsExt-file++.....OccrLen+.....*
ICTLFIL EUDS
I*
I* Data structure used for date editing: The data structure
I* contains a 6-position date field with three 2-position
I* subfields. This provides the program with individual
I* reference to the month, day, and year.

```

Figure 176 (Part 2 of 8). Sample RPG/400 Program - PRG02

PRG02 (Control File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IDsnam.....NODsExt-file++.....OccrLen+.....*
I          DS
I.....PFromTo++DField+L1M1FrPlMnZr...*
I          1   60CDATE
I          1   20CDTMM
I          3   40CDTDD
I          5   60CDTYT

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C*  MAINLINE CALCULATIONS
C*****
C*  This program uses the RPG/400 program cycle to handle input and
C*  output to the display file and the data area data structure.
C*  The MAINLINE routine first checks for an end-of-job request
C*  indicated by function key indicator F3 (*INKC).  If *INKC is
C*  off (equal to 0), the general error indicator *IN60 is set off
C*  (equal to 0), and the error message field is filled with blanks.
C*  The subroutine EDITSR is processed to validate input from the
C*  display file.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C          *INKC    IFEQ '0'
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C          EXSR EDITSR
C*
C*  Control returns to the statement following the EXSR line above
C*  and the general error indicator *IN60 is checked to see if it is
C*  off (equal to 0), and the display file input is moved to the
C*  data area data structure fields.  The last record indicator *INLR
C*  is set on (equal to 1) and the program ends.
C*
```

Figure 176 (Part 3 of 8). Sample RPG/400 Program - PRG02

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN60      IFEQ '0'
C          MOVE CWEEK      CWKDT
C          MOVE CMNTH      CMTDT
C          MOVE CENTR      CALLE
C          MOVE '1'        *INLR
C          END
C*
C* The preceding END statement denotes the end of the second IF
C* statement.
C*
C* The following ELSE statement is associated with the initial
C* IF statement checking for end-of-job requested *INKC. The
C* statements that follow the ELSE perform the *INKC indicator on
C* condition (equal to 1) by setting on the last record indicator
C* *INLR.
C*
C          ELSE
C          MOVE '1'        *INLR
C          END
C*
C* The preceding END statement denotes the end of the initial
C* IF statement.
C*
C*****
C* EDITSR subroutine verifies the week ending and month
C* ending dates entered and the all entries made flag.
C*****
C*
C          EDITSR      BEGSR

```

Figure 176 (Part 4 of 8). Sample RPG/400 Program - PRG02

PRG02 (Control File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The display input field CWEEK is moved to the data structure
C* field to provide separate reference to the month, day, and year.
C* The subroutine DATESR is processed to verify the date format
C* and control returns to the statement following the EXSR. If no
C* errors are found in the week ending date (*IN51 and *IN53 both
C* equal 0), the same process is done using CMNTH month ending date.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          MOVE CWEEK      CDATE
C          EXSR DATESR
C          *IN51   IFEQ '0'
C          *IN53   ANDEQ '0'
C          MOVE CMNTH     CDATE
C          EXSR DATESR
C          END
C*
C* The all entries made flag CENTR is checked to ensure only the
C* values Y or N are entered. Error indicator *IN52 is set off
C* (equal to 0). If the field is equal to Y or N, ELSE *IN52 is
C* set on (equal to 1).
C*
C          CENTR   IFEQ 'Y'
C          CENTR   OREQ 'N'
C          MOVE '0'      *IN52
C          ELSE
C          MOVE '1'      *IN52
C          END
```

Figure 176 (Part 5 of 8). Sample RPG/400 Program - PRG02


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code checks the error indicators and moves the
C* appropriate error message from the compile-time array. The
C* general error indicator *IN60 is set on (equal to 1) if any of
C* the three error indicators are on and the error message moved.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *IN51      IFEQ '1'
C          *IN52      OREQ '1'
C          *IN53      OREQ '1'
C                      MOVE '1'          *IN60
C                      END
C*
C  51                MOVE ERR,1      EMESS  50
C  52                MOVE ERR,2      EMESS
C  53                MOVE ERR,3      EMESS
C*
C                      ENDSR
C*****
C* DATESR subroutine verifies the date format entered. The date
C* has been moved to the program data structure before this routine
C* is processed.
C*****
C          DATESR      BEGSR
C*
C* The year entered is processed to determine if it is a leap year.
C* The year is divided by 4 and the remainder moved to a separate
C* work field. Resulting indicator 50, positions 58 and 59, is
C* set on (equal to 1) if the remainder is zero. This indicates
C* a leap year and is used to condition subsequent calculations.
C*
C          CDTYY      DIV  4          LEAPYR  30
C          MVR                LEAPRM  30      50

```

Figure 176 (Part 6 of 8). Sample RPG/400 Program - PRG02

PRG02 (Control File Maintenance)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The error indicator *IN51 is set off (equal to 0) and then the
C* month is checked. If the month is greater than 12 or less than 1,
C* the error indicator *IN51 is set on (equal to 1).
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          MOVE '0'          *IN51
C          CDTMM          IFGT 12
C          CDTMM          ORLT 1
C          MOVE '1'          *IN51
C          END
C* The following code verifies the day entered. The month entered
C* is used as the index to the compile time array's ARL and ARM.
C* The ARL array contains the number of days in a month during a
C* leap year; the ARM array contains the number of days in a month
C* for a non-leap year. If the number of days entered is greater
C* than the array element, indicator 51 is set on.
C          *IN51          IFEQ '0'
C          Z-ADDCDTMM          M          20
C          *IN50          IFEQ '1'
C          CDTDD          COMP ARL,M          51
C          ELSE
C          CDTDD          COMP ARM,M          51
C          END
C          END
C* The year entered is compared to the system year reserved word
C* UYEAR. If they are not equal, *IN53 equals 1, a warning message
C* is issued. If you press F12 (*INKL) to accept the value entered,
C* the verification is bypassed.
C          MOVE '0'          *IN53
C          *INKL          IFEQ '0'
C          CDTYY          IFNE UYEAR
C          MOVE '1'          *IN53
C          END
C          END
C          ENDSR
```

Figure 176 (Part 7 of 8). Sample RPG/400 Program - PRG02

Time File Transaction Entry

You select option 3 (Time file transaction entry) on the Time Reporting System Main Menu to enter employee time file transactions. The entries can be made at any time before you begin your weekly update. Figure 177 shows the Time Reporting System Main Menu. Option 3 calls program PRG03 by using the CALL PGM(PGM03) command.

```
TMENU                               Time Reporting System
                                      Main Menu

      1. Master file maintenance      (PRG01)
      2. Control file maintenance     (PRG02)
      3. Time file transaction entry (PRG03)
      4. Weekly time file update      (PROC1)
      5. Monthly time file update & reporting (PROC3)

      8. Display messages              (DSPMSG)
      9. Sign off                      (SIGNOFF)

Selection or command
====>
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel
F13=User support  F16=System main menu
```

Figure 177. Time Reporting System Main Menu

Time Reporting Transaction Entry - PRG03FM

The transaction entry application of the time reporting system consists of two displays. On the first display, you enter an employee number and change the week ending date. On the second display, you enter the detail transactions. The second display consists of three formats: a subfile control record format, a subfile record format, and a format containing valid function key descriptions and error messages.

Employee Selection Display

Figure 178 shows the first part of the Employee Time Entry display of the time reporting system.

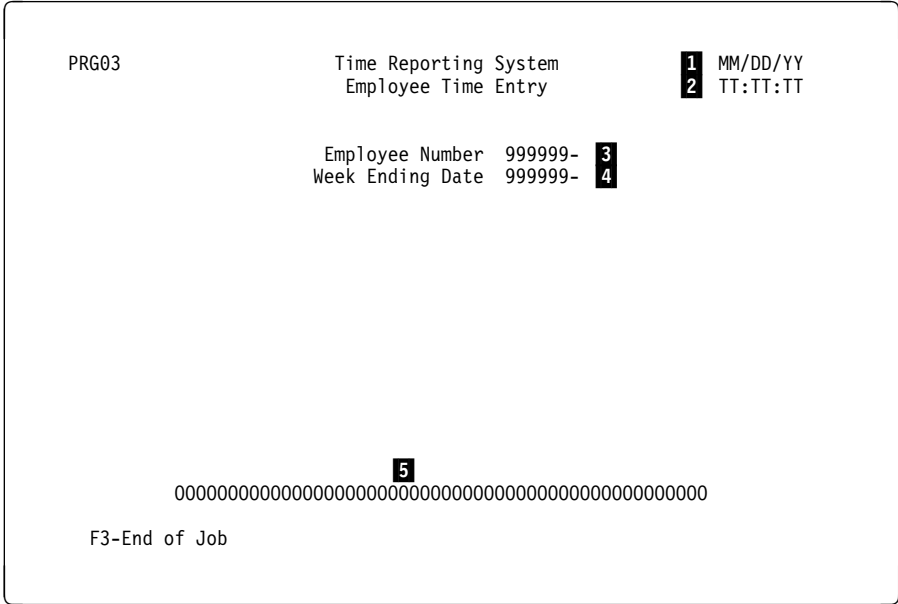


Figure 178. Time Reporting System - Employee Time Entry Display (Part 1)

The first part of the Employee Time Entry display consists of one format, EMPSEL. The date **1** and time of day **2** are updated each time the you press enter. You must enter an employee number **3** and can optionally change the week ending date **4**. When you press Enter, the program verifies that the employee number exists in the employee master file and has an active status code. If no errors are found, the second part of the Employee Time Entry display appears (see Figure 179 on page 396). If errors are found, the program returns format EMPSEL with an error message **5**. You can press F3 to end the job.

Possible error messages are:

- Employee master record not found.
- Employee master record not active.

validated. When all entries pass the validation, the employee time entry hours **4** is updated.

The **C** EMPERR record format displays the function keys allowed and any error messages **12** from the program. You can press F3 to end the job, or F1 to return to the employee selection display. If you press F5, the program rebuilds the subfile and the transaction entry display appears. If you press any of these function keys before you press Enter, no updates are performed on the transaction file. However, entries made and passed to the program by pressing Enter, before pressing the function key, are kept.

Possible error messages are:

- A project code or a reason code is required.
- Invalid project code.
- Invalid reason code.
- No hours entered on this transaction.

Time Reporting Transaction Entry Data Descriptions - PRG03FM

Figure 180 on page 398 shows the DDS for the PRG03FM Time Reporting Transaction Entry display file. Four record formats, identified by R in position 17, are followed by the format name in positions 19 through 28. The following keywords are used:

ALARM	Activates the audible alarm.
BLINK	Blinks the cursor.
CAnn	Makes the function key specified in the keyword available for use.
DATE	Displays the current job date as a constant.
DSPATR	Specifies a display attribute for the field.
DSPSIZ	Specifies the display size to which the program can open this file.
EDTCDE	Specifies editing on an output capable numeric field.
EDTWRD	Specifies an edit word on an output capable numeric field.
OVERLAY	Specifies that the record format you are defining appears on the display without the entire display being erased first.
REFFLD	Refers to the attributes of a previously defined field.
SFL	Record level keyword specifying that this record format is to be a subfile record format.
SFLCLR	Used in the subfile control record format so that your program can clear the subfile of all records.
SFLCTL	Record level keyword specifying that this record format is to be a subfile control record format.
SFLDSP	Used in the subfile control record format so that the OS/400 system displays the subfile when your program sends an output operation to the subfile control record format.
SFLDSPSTL	Used in the subfile control record format so that the OS/400 system displays fields in the subfile control record format when your program sends an output operation to the subfile control record format.
SFLPAG	Used in the subfile control record format to specify the number of records in the subfile to be displayed at the same time.
SFLSIZ	Used in the subfile control record format to specify the number of records in the subfile.
TIME	Display the current system time as a constant.

PRG03FM (Transaction Entry)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A*   PRG03FM - Transaction Entry Data Descriptions
A*   DESCRIPTION - A file containing record formats for the employee
A*                   time file transctions.
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A                               DSPSIZ(24 80 *DS3)
A           R EMPSEL
A                               CA03(03 'End of job')
A                               BLINK
A 60                               ALARM
A                               2 5'PRG03'
A                               2 30'Time Reporting System'
A                               2 71DATE
A                               EDTCDE(Y)
A                               3 31'Employee Time Entry'
A                               3 71TIME
A                               6 29'Employee Number'
A           EMPNO       R           B 6 46REFFLD(RCEMP/EMPNO *LIBL/EMPMST)
A                               8 28'Week Ending Date'
A           CWKDTX     R           B 8 46REFFLD(RCCTL/CWKDT *LIBL/CTLFIL)
A           EMESS           50A 0 21 15
A 60                               DSPATR(HI)
A                               23 6'F3-End of Job'
A           R EMPFIL
A           RECNO           3S 0B 11 2DSPATR(ND)
A                               DSPATR(PR)
A           STATUS           1A  B 11 10
A           PRFLAG           2   0 11 14
A 60                               DSPATR(HI)
A 60                               DSPATR(BL)
A           PRCDEX       R           B 11 17REFFLD(RCWEK/PRCDE *LIBL/TRWEEK)
A           RSFLAG           2   0 11 27
A 60                               DSPATR(HI)
A 60                               DSPATR(BL)
A           RSCDEX       R           B 11 30REFFLD(RCWEK/RSCDE *LIBL/TRWEEK)
A           HRFLAG           2   0 11 40

```

Figure 180 (Part 1 of 3). Time Reporting Transaction Entry Data Descriptions - PRG03FM


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A 60                                DSPATR(HI)
A 60                                DSPATR(BL)
A          EHWRKX      R          B 11 43REFFLD(RCWEK/EHWRK *LIBL/TRWEEK)
A          ACDATX      R          B 11 54REFFLD(RCWEK/ACDAT *LIBL/TRWEEK)
A          TFRRN       R          B 11 67REFFLD(RCWEK/TFRRN *LIBL/TRWEEK)
A                                DSPATR(PR)
A          R EMPCTL      SFLCTL(EMPFIL)
A                                SFLSIZ(0050)
A                                SFLPAG(0010)
A 30                                SFLCLR
A N30                               SFLDSP
A N30                               SFLDSPCTL
A                                OVERLAY
A                                CA03(03 'End of job')
A                                CA05(05 'Restart employee display a-
A                                t beginning')
A                                CA01(01 'Employee selection display-
A                                ')
A                                BLINK
A 60                                ALARM
A          2 5'PRG03'
A          2 30'Time Reporting System'
A          2 71DATE
A          EDTCDE(Y)
A          3 31'Employee Time Entry'
A          3 71TIME
A          5 9'Emp1 #'
A          EMPNO      R          0 5 16REFFLD(RCEMP/EMPNO *LIBL/EMPMST)
A                                DSPATR(CS)
A          5 25'Name'

```

Figure 180 (Part 2 of 3). Time Reporting Transaction Entry Data Descriptions - PRG03FM

PRG03FM (Transaction Entry)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A      ENAME      R      0  5 30REFFLD(RCEMP/ENAME *LIBL/EMPMST)
A      DSPATR(CS)
A      5 63'Dept'
A      EDEPT      R      0  5 68REFFLD(RCEMP/EDEPT *LIBL/EMPMST)
A      DSPATR(CS)
A      7 15'Employee time entry hours in-
A      weekly file'
A      CURHRS      5 10  7 57EDTWRD(' . ')
A      DSPATR(CS)
A      9  8'Action  Project  Reason  -
A      Hours  Actual date'
A      9 66'Relative'
A      10 9'Code  Code  Code  W-
A      orked  Worked'
A      10 66'Record #'
A      R EMPERR
A      EMESS      50  0 22 17
A 60      DSPATR(HI)
A      23 6'F3-End of Job'
A      23 24'F1-Employee Selection'
A      23 50'F5-Restart Employee Display'

```

Figure 180 (Part 3 of 3). Time Reporting Transaction Entry Data Descriptions - PRG03FM

Time Reporting Transaction Entry RPG/400 Program - PRG03

Figure 181 shows the RPG/400 program for the time reporting transaction entry. Comments are included as part of the program to describe the various sections of the code and the RPG/400 logic.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG03 - Time Reporting Transaction Entry RPG/400 Program
F* DESCRIPTION - Time reporting transaction entry using subfile
F*                work station processing.
F*****
F* INDICATORS USED:
F* 31 - Record read does not match factor 1 on READE operation
F* 32 - Subfile is full
F* 35 - No more changed records in subfile on READC operation
F* 41 - Record found on SETLL operation equal to factor 1
F* 45 - Record not found in TRWEEK file on CHAIN operation
F* 60 - General error condition
F* 64 - Record not found in EMPMST file on CHAIN operation
F* 65 - Record not found in PRJMST file on CHAIN operation
F* 66 - Record not found in RSNMST file on CHAIN operation
F* KA - Return to employee selection
F* KC - End of job requested
F* KE - Restart employee transaction display
F* LR - Last record
F*****
F* SUBROUTINES USED:
F* EMPEDT - Verifies the employee requested in EMPSEL format
F* SFLEDT - Verifies the subfile entries in EMPFIL format
F*****
```

Figure 181 (Part 1 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F* This program uses all externally described files. Files
F* used are: PRG03FM - transaction entry display file
F*           EMPMST - employee master file
F*           PRJMST - project master file
F*           RSNMST - reason code master file
F*           TRWEEKL - logical view of the weekly transaction
F*                   file by employee number EMPNO and
F*                   relative record number TFRRN.
F*****
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FPRG03FM CF E                                WORKSTN
F                                             RECNO KSFIL EMPFIL
F*
F* The continuation line for the WORKSTN file identifies the record
F* format (EMPFIL) that is to be used as a subfile. The relative
F* record number field (RECNO) controls which record within the
F* subfile is being accessed.
F*
FEMPMST IF E           K           DISK
FPRJMST IF E           K           DISK
FRSNMST IF E           K           DISK
FTRWEEKL UF E         K           DISK           A

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* Compile time array containing error descriptions.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E           ERR       1   6 50
E*

```

Figure 181 (Part 2 of 17). Sample RPG/400 Program - PRG03

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The following input specification defines a named constant field
I* FLAG. This constant is used to indicate transaction errors on
I* the subfile display.
I*
I.....Constant+++++++C.....Field+.....*
I          '->'          C          FLAG
I*
I* Externally described control file data area
I*
IDsnam.....NODsExt-file++.....OccrLen+.....*
ICTLFIL    EUDS

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* MAINLINE CALCULATIONS
C*****
C* This mainline routine controls display file processing and
C* editing. You must first select an employee number from the
C* EMPSEL display. The number is verified against the employee
C* master. The transaction entry display consists of three formats:
C* EMPCTL subfile control record, EMPFIL subfile record, and EMPERR
C* error display and message record.
C* You can return to the employee selection display, end the job,
C* or restart the subfile display by using the function keys.
C* The roll keys scroll through the existing entries. You can add
C* new records, and change or delete existing records. The weekly
C* transaction file is updated with valid entries when you press
C* Enter.
C*****

```

Figure 181 (Part 3 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The mainline routine begins by moving the week ending date to the
C* display file week ending date. This protects the week ending
C* date in the data area from being updated since you can change
C* the date on the EMPSEL display. As part of housekeeping,
C* general error indicator *IN60 is set off, blanks are moved to
C* the error message display field, and the employee number is set
C* to zeros before writing the EMPSEL format. The BEGIN TAG
C* provides a label to which the program can return when requested
C* from the entry display using function key 1 *INKA.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          MOVE CWKDT      CWKDTX
C          BEGIN          TAG
C          MOVE '0'        *IN60
C          MOVE *BLANKS    EMESS
C          Z-ADD0          EMPNO
C*
C* The SELTAG TAG provides a label the program returns to if an
C* error is found in the employee selection display EMPSEL. The
C* EMPSEL format is written to the work station using EXFMT. The
C* EXFMT causes a write to and a read from the display. The
C* function key F3 (*INKC) is checked to see if end-of-job is
C* requested, and if not (*INKC equals 0), the employee edit EMPEDT
C* subroutine is processed. If end-of-job is requested by F3,
C* which sets on indicator 03 (see display file), the last record
C* indicator LR is set on and the RETRN operation starts.
C*
C          SELTAG      TAG
C          EXFMTEMPSEL
C 03          SETON          LR
C 03          RETRN
C          EXSR EMPEDT
```

Figure 181 (Part 4 of 17). Sample RPG/400 Program - PRG03

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* IF the general error indicator *IN60 is on (equal to 1), the
C* program branches back to the SELTAG and redisplay the EMPSEL
C* format with the appropriate error message.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *IN60          IFEQ '1'
C          GOTO SELTAG
C          END
C*
C* At this point, the EMPSEL format has been verified and the
C* program displays the time entry formats. Existing time entry
C* transaction records for the employee are displayed. The
C* REPEAT TAG provides a label to which the program can branch if
C* F5 (*INKE) has been requested or when all subfile entries have
C* been verified and applied to the transaction file. The resulting
C* indicators that are used when building and processing the subfile
C* are set off (equal to 0) before processing is done.
C*
C          REPEAT      TAG
C          SETOF              313235

```

Figure 181 (Part 5 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code positions the program at the employee number
C* requested within the transaction file. If a record is found
C* that matches EMPNO, resulting indicator 41 is set on.
C* Indicator 30 is defined in the subfile control format to allow
C* clearing of the subfile display format. Indicator 30 is set on
C* and the EMPCTL format is written. The EMPERR format is also
C* written to display valid function keys on the bottom of the
C* screen. Indicator 30 is set off and work fields are initialized.
C* RECNO is defined on the file description continuations line;
C* LSRRN is used to store the last relative record number from the
C* transaction file; CURHRS is used to display the total hours that
C* are entered for the employee, and STATUS is used to allow
C* deletion of existing subfile records.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          EMPNO      SETLLTRWEEKL                41
C          MOVE '1'      *IN30
C          WRITEEMPCTL
C          WRITEEMPERR
C          MOVE '0'      *IN30
C          Z-ADD0        RECNO    30
C          Z-ADD0        LSRRN    50
C          Z-ADD0        CURHRS   51
C          MOVE *BLANKS  STATUS
```

Figure 181 (Part 6 of 17). Sample RPG/400 Program - PRG03


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following DOWEQ operation is processed until indicator 31
C* (*IN31) is set on by the READE operation or by the subfile
C* being filled. The READE reads all existing entries in the
C* transaction file that are equal to the employee number (EMPNO).
C* If an entry is found (*IN31 equals 0), the fields from the
C* transaction file are moved to the subfile display fields.
C* Three error flags in each subfile record are used to point out
C* not valid entries; these flags are blanked: The subfile relative
C* record number RECNO is incremented by 1, the hours from the
C* transaction record added to the total for the employee, and the
C* subfile record is written. When the subfile is full, resulting
C* indicator 32 on the WRITE operation is set on.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C          *IN31      DOWEQ '0'
C          EMPNO      READERC WEEK          31
C*
C          *IN31      IFEQ '0'
C          MOVE PRCDE  PRCDEX
C          MOVE RSCDE  RSCDEX
C          Z-ADDEHWRK  EHWRKX
C          Z-ADDACDAT  ACDATX
C          Z-ADDTFRRN  LSRRN
C          MOVE *BLANKS PRFLAG
C          MOVE *BLANKS RSFLAG
C          MOVE *BLANKS HRFLAG
C          ADD 1       RECNO
C          ADD EHWRK   CURHRS
C          WRITEEMPFIL          32
C          END

```

Figure 181 (Part 7 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* If the subfile full indicator (*IN32) is on, indicator 31 is
C* set on to end the DOWEQ operation. If 31 is off, the preceding
C* code is processed again.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN32      IFEQ '1'
C          MOVE '1'      *IN31
C          END
C          END
C* The preceding END denotes the end of the Do While operation.
C*
C* The following code determines if the subfile is filled. If
C* indicator 30 is off (*IN30 equals 0), then the DOWEQ operation
C* processes until the remainder of the subfile is filled with
C* blank records.
C*
C          *IN32      DOWEQ'0'
C          MOVE *BLANKS STATUS
C          MOVE *BLANKS PRCDEX
C          MOVE *BLANKS RSCDEX
C          Z-ADD0      EHWRKX
C          Z-ADD0      ACDATX
C          Z-ADD0      TFRRN
C          ADD 1      RECNO
C          WRITEEMPFIL          32
C          END
C*
C* The preceding END denotes the end of the Do While operation.
C*
```

Figure 181 (Part 8 of 17). Sample RPG/400 Program - PRG03

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The SFDISP TAG provides a label to which the program can branch
C* if errors are found in the subfile records. If indicator 60 is
C* on, the EMPERR format is written to display the error message
C* followed by the EXFMT operation to write the EMPCTL subfile
C* control format. If F1 (*INKA) is pressed from the transaction
C* entry display, indicator 01 is set on (see display file) and the
C* program returns to the BEGIN tag and displays the employee
C* selection format EMPSEL. If F3 is entered (*INKC) requesting
C* end-of-job, indicator 03 is set on. This in turn sets on
C* the last record indicator LR and the RETRN operation. If F5
C* (*INKE) is entered requesting a redisplay of employee time
C* entries, the program branches back to the REPEAT TAG and the
C* subfile is rebuilt. No file updates are performed if any of
C* these functions keys are used.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          SFDISP      TAG
C  60          WRITEEMPERR
C          EXFMTEMPCTL
C  01          GOTO BEGIN
C  03          SETON                      LR
C  03          RETRN
C  05          GOTO REPEAT
C*

```

Figure 181 (Part 9 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The following code prepares the program for processing the
C* subfile. The subfile relative record number is set to a value
C* of 1 using the Z-ADD operation, and the error indicator *IN60
C* is set off. The transaction file key is established using the
C* KLIST and KFLD operation codes. The KLIST and KFLD operation are
C* declarative (cannot be processed) operations indicating the search
C* argument for the file. They could have appeared anywhere within
C* the calculations but are coded here for documentation purposes.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          Z-ADD1          RECNO
C          MOVE '0'        *IN60
C          TRKEY          KLIST
C          KFLD            EMPNO
C          KFLD            TFRRN
C*
C* The following DOWEQ operation processes the transaction file
C* and all changed records in the subfile. The Do While operation
C* continues until indicator 35 (*IN35) is set on. Indicator 35 is
C* defined as a resulting indicator on the READC (read changed
C* records) operation, which is set on when all changed subfile
C* records are read. If the display field STATUS is blank and
C* indicator 35 is off (equal to 0), the subfile edit subroutine
C* SFLEDT is processed. The program returns from the edit and if
C* the error indicator 60 is on, the program branches back to
C* the SFDISP TAG, writes the EMPERR format, and displays the
C* EMPCTL format again. If indicator 35 is on, the program
C* branches back to the REPEAT TAG and rebuilds the subfile.
C*
```

Figure 181 (Part 10 of 17). Sample RPG/400 Program - PRG03

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN35      DOWEQ'0'
C          READCEMPFIL                                35
C          STATUS    IFEQ ' '
C          *IN35      ANDEQ'0'
C          EXSR SFLEDT
C          END
C 60          GOTO SFDISP
C 35          GOTO REPEAT
C*
C* The following code is still part of the Do While operation.
C* Using the TRKEY field built by the KLIST operation, the trans-
C* action file is accessed using the CHAIN operation. If the
C* record does not exist in the file, resulting indicator 45 is
C* set on. If indicator 45 is on, the record must be added to the
C* transaction file. The last relative record number, which is
C* stored in field LSRRN, is incremented by 1 and the display fields
C* are moved to the transaction record. The new record is then
C* written using the WRITE operation and record format RCWEEK.
C*
C          TRKEY      CHAINTRWEEKL                    45
C          *IN45      IFEQ '1'
C          ADD 1          LSRRN
C          Z-ADDLSRRN    TFRRN
C          MOVE PRCDEX    PRCDE
C          MOVE RSCDEX    RSCDE
C          Z-ADDEHWRKX    EHWRK
C          Z-ADDACDATX    ACDAT
C          WRITERCWEEK
C          END

```

Figure 181 (Part 11 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* If indicator 45 is off (*IN45 equals 0) and the display file
C* field STATUS equals D, the operator requests the deletion of the
C* record. The DELET operation code is processed and the record
C* is removed from the file. The record cannot be retrieved.
C*
CL0N01N02N03Factor1+++OpdcFactor2+++ResultLenDHHiLoEqComments+++++*
C      *IN45      IFEQ '0'
C      STATUS    ANDEQ'D'
C                      DELETRCWEK
C                      END
C*
C* If indicator 45 is off and the display file field STATUS is not
C* equal to D, the operator has changed an existing record.
C* The display file fields are moved to the transaction file fields
C* and the UPDAT operation is processed to update the file.
C*
C      *IN45      IFEQ '0'
C      STATUS    ANDNE'D'
C                      MOVE PRCDEX    PRCDE
C                      MOVE RSCDEX    RSCDE
C                      Z-ADDEHWRKX    EHWRK
C                      Z-ADDACDATX    ACDAT
C                      UPDATRCWEK
C                      END
C                      END
C*
C* The preceding END denotes the end of the Do While operation.
C*
```

Figure 181 (Part 12 of 17). Sample RPG/400 Program - PRG03

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* EMPEDT subroutine verifies the employee number requested.
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          EMPEDT      BEGSR
C*
C* The error indicator 60 (*IN60) is set off (equal to 0) and the
C* error message field EMESS is filled with blanks.
C*
C          MOVE '0'      *IN60
C          MOVE *BLANKS  EMESS
C*
C* Using the employee number entered (EMPNO) and the CHAIN operation,
C* the employee master file is accessed. If the record is found,
C* resulting indicator 64 is set on. If *IN64 is on, error indicator
C* 60 is set on and error array element 1 is moved to EMESS. The
C* ELSE operation indicates the record is found (*IN60 equals 0)
C* and the record status is checked for a value of D for deleted.
C* If true, error indicator 60 is set on and error array element 2
C* is moved to EMESS.
C*
C          EMPNO      CHAINEMPST          64
C          *IN64      IFEQ '1'
C                   MOVE '1'          *IN60
C                   MOVE ERR,1        EMESS
C                   ELSE
C          ACREC      IFEQ 'D'
C                   MOVE '1'          *IN60
C                   MOVE ERR,2        EMESS
C                   END
C                   END
C                   ENDSR

```

Figure 181 (Part 13 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* SFLEDT subroutine verifies the subfile entries.
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          SFLEDT      BEGSR
C*
C* The subfile error flags PRFLAG, RSFLAG, and HRFLAG are set to
C* blanks, the error indicator is set off and the error message
C* field EMESS set to blanks. The subfile error flags contain ->
C* (named constant field FLAG) when an error is found. The flag
C* blinks on the display in highlighted mode beside the field(s)
C* in error.
C*
C          MOVE *BLANKS    PRFLAG
C          MOVE *BLANKS    RSFLAG
C          MOVE *BLANKS    HRFLAG
C          MOVE '0'        *IN60
C          MOVE *BLANKS    EMESS
C* The first check determines if both the project code and reason
C* code subfile fields are blank. If they are both blank, FLAG is
C* moved to project code flag and reason code flag, the error
C* indicator is set on, and error array element 3 is moved to EMESS.
C*
C          PRCDEX      IFEQ *BLANKS
C          RSCDEX      ANDEQ*BLANKS
C          MOVE FLAG    PRFLAG
C          MOVE FLAG    RSFLAG
C          MOVE '1'     *IN60
C          MOVE ERR,3   EMESS
C          ELSE
```

Figure 181 (Part 14 of 17). Sample RPG/400 Program - PRG03


```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
```

```
C*
```

```
C* The second check determines if a project code is entered
C* (not equal to blanks), and using the code entered, accesses the
C* project master. Resulting indicator 65 is set on by the CHAIN
C* operation if the record is not found. If indicator 65 is on,
C* or if it is off but the record status field ACREC in the record
C* contains D for deleted, FLAG is moved to the project code error
C* flag, error indicator 60 is set on, and error array element 4 is
C* moved to EMESS.
```

```
C*
```

```
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
```

```
C          PRCDEX    IFNE *BLANKS
C          PRCDEX    CHAINPRJMST          65
C          *IN65     IFEQ '1'
C          *IN65     OREQ '0'
C          ACREC     ANDEQ'D'
C          MOVE FLAG      PRFLAG
C          MOVE '1'       *IN60
C          MOVE ERR,4     EMESS
C          END
C          ELSE
```

```
C*
```

```
C* The third check determines if a reason code is entered (not equal
C* to blanks), and using the code entered, accesses the reason code
C* master. Resulting indicator 66 is set on by the CHAIN operation
C* if the record is not found. If indicator 66 is on, or if it is
C* off but the record status field ACREC in the record contains D
C* for deleted, FLAG is moved to the reason code error flag, error
C* indicator 60 is set on and error array element 5 is moved to EMESS.
```

```
C*
```

Figure 181 (Part 15 of 17). Sample RPG/400 Program - PRG03

PRG03 (Transaction Entry)

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          RSCDEX    IFNE *BLANKS
C          RSCDEX    CHAINRSNMST                66
C          *IN66     IFEQ '1'
C          *IN66     OREQ '0'
C          ACREC     ANDEQ'D'
C          MOVE FLAG      RSFLAG
C          MOVE '1'      *IN60
C          MOVE ERR,5    EMESS
C          END
C          END
C          END
C          END
C* The fourth check verifies that hours are entered on the subfile
C* transaction.  If the hours worked field is equal to zero, FLAG
C* is moved to the hours worked error flag, error indicator 60 is
C* set on, and error array element 6 is moved to EMESS.
C*
C          EHWRKX    IFEQ *ZEROS
C          MOVE FLAG      HRFLAG
C          MOVE '1'      *IN60
C          MOVE ERR,6    EMESS
C          END
C*
C* If errors are found in the subfile entry edit, error indicator
C* 60 is set on, and the subfile record is updated.  This is
C* required since we moved values to one or more of the error flags.
C* These error flags are then displayed when the EMPCTL format is
C* written.
C*
C          *IN60     IFEQ '1'
C          UPDATEMPFIL
C          END
C          ENDSR

```

Figure 181 (Part 16 of 17). Sample RPG/400 Program - PRG03

```
* ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
0*
0* The compile time array ERR is entered below. The array is
0* preceded by "** " to denote the beginning of the array
0* and begins in column 1 of the output specification.
0*
** Array ERR - Error descriptions
    EMPLOYEE MASTER RECORD NOT FOUND
    EMPLOYEE MASTER RECORD NOT ACTIVE
A PROJECT CODE OR A REASON CODE IS REQUIRED
    INVALID PROJECT CODE
    INVALID REASON CODE
NO HOURS ENTERED ON THIS TRANSACTION
```

Figure 181 (Part 17 of 17). Sample RPG/400 Program - PRG03

Weekly Time File Update

Once a week the time entry transaction file TRWEEK is processed to: determine if all employees enrolled in the time reporting system entered their time transactions; update the master files with transactions entered; and prepare the transaction files for new week processing. The weekly application consists of two RPG/400 programs and two control procedures.

Figure 182 shows the Time Reporting System Main Menu. The first step in the weekly update is to change the week ending date in the control file by selecting option 3 (Time file transaction entry). After the control file is updated, call the weekly update by selecting option 4 (Weekly time file update). Option 4 calls PROC1 (See Figure 183 on page 419) by using the CALL PGM(PROC1) command.

```
TMENU                                Time Reporting System
                                      Main Menu

      1. Master file maintenance              (PRG01)
      2. Control file maintenance            (PRG02)
      3. Time file transaction entry          (PRG03)
      4. Weekly time file update             (PROC1)
      5. Monthly time file update & reporting (PROC3)

      8. Display messages                    (DSPMSG)
      9. Sign off                            (SIGNOFF)

Selection or command
====>
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel
F13=User support  F16=System main menu
```

Figure 182. Time Reporting System Main Menu

The weekly time file update of the time reporting system consists of three control-level programs:

1. The first CL program, PROC1, runs interactively to determine if all employees have made their time entries. See Figure 183 on page 419.
2. CL program PROC1 calls RPG/400 program PRG05. Within PRG05 is the RPG/400 operation code CALL, which calls the second CL program PROC5. For employees who have not entered a time reporting transaction, PROC5 sends messages to their message queues. See Figure 184 on page 419.
3. CL program PROC1 submits the third CL program, PROC2, to batch. PROC2 produces the weekly employee transaction reports, updates the master files and prepares the transaction files for the new week. See Figure 185 on page 420.

```

/* Weekly Time File Update: */
/* This procedure is run weekly to process the weekly time */
/* entry transaction file. The file is reviewed for missing */
/* entries. Both the person who asks for this procedure and the */
/* employee whose entries are missing are notified that entries */
/* are missing. The batch update procedure PROC2 is then */
/* submitted for processing. */
/* */
/* Program PRG05 reads the employee master file and checks for */
/* an entry in the weekly transaction file. If an entry is not */
/* found, the program calls procedure PROC5 to issue a message */
/* to the employee. At end of job, the person who asked is */
/* issued a message stating if all entries have been made or not.*/
/* */
BEGIN:      PGM
            CHGJOB      SWS('00000000')
            CALL       PGM(PRG05)
            IF         COND(%SWITCH(XXXXXXX1)) THEN(DO)
                SBMJOB      CMD(CALL PGM(PROC2)) JOB(*JOB)
            ENDDO
ENDIT:      ENDPGM

```

Figure 183. CL Program PROC1

```

/* This procedure sends an information message to the employees' */
/* message queues stating their time entries are missing. */
/* */
            PGM          PARM(&EUSRI)
            DCL         VAR(&EUSRI) TYPE(*CHAR) LEN(8)
            SNDUSRMSG   MSG('Your time entries are missing for +
                        prior week') MSGTYPE(*INFO) TOMSGQ(&EUSRI)
            MONMSG     MSGID(CPF2559)
            ENDPGM

```

Figure 184. CL Program PROC5

Weekly Time File Update

```
/* Weekly Time File Update */
/* This procedure is run weekly to produce the weekly */
/* employee transaction report and to update the time */
/* reporting master files. */
/* */
/* Program PRG09 reads the weekly transaction entry file */
/* to produce the weekly report and update the Tonth */
/* to date hours in the Taster files. */
/* */
BEGIN: PGM
CALL PGM(PRG09)

/* */
/* STEP2 adds the weeks time entry transactions to the */
/* monthly transaction file. */
/* */
STEP2: CPYF FROMFILE(TRWEEK) TOFILE(TRMNTH) MBROPT(*ADD) +
FMTOPT(*MAP *DROP)

/* */
/* STEP3 clears the weekly transaction file in preparation*
/* for new weeks entry. */
/* */
STEP3: CLRPFM FILE(TRWEEK)

/* */
ENDIT: ENDPGM
```

Figure 185. CL Program PROC2

Time File Entry Edit RPG/400 Program - PRG05

This program processes the employee master file and uses the employee number to access the weekly transaction entry file to determine if at least one transaction record exists for the employee. If no entries are found, the program calls a control language program to issue a message to the employee who has not made entries.

Figure 186 shows program PRG05 with embedded comments to explain the logic flow, and the use of RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG05 - Time Reporting Time File Entry Edit
F* DESCRIPTION - This program edits the weekly transaction entry
F*                file and the employee master to determine if all
F*                employees enrolled have entered their weekly
F*                transactions.
F*****
F* This program uses externally described files. Files
F* used are: EMPMST - employee master file
F*           TRWEEKL - logical view of weekly transaction entry
F*                file by employee number
F*****
F* INDICATORS USED:
F* 50 - No record found on SETGT greater than search argument
F* 51 - No matching record on REDPE operation
F* 60 - Missing time entries from transaction file
F* 99 - First cycle processing
F* LR - Last record
F*****
```

Figure 186 (Part 1 of 6). PRG05 program

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FEMPMST IP E DISK
F*
F* The weekly transaction file contains the entry UC in positions
F* 71 through 72. This entry allows the program to control the
F* opening and closing of this file (see first cycle processing and
F* last record processing for details).
F*
FTRWEEKL IF E K DISK UC
F*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* Compile time array containing requestor messages.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E MESS 1 2 50
E*

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* Data structure CTLDS processes the control file data area CTLFIL.
I* Processing of this data area is controlled by the program using
I* the data area operation codes (see first cycle processing and
I* last record processing for details).
I*
IDpname....NODsExt-file++.....OccrLen+.....*
ICTLDS DS
I.....Ext-field+.....PFromTo++DField+.....*
I 1 6 CTCDE
I 7 120WKEND
I 13 180CMTDT
I 19 19 CALLE

```

Figure 186 (Part 2 of 6). PRG05 program


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* FIRST CYCLE PROCESSING: The following code is processed on
C* the first RPG/400 program cycle only. Indicator 99 is tested
C* IFEQ to '0', the transaction weekly file is opened, and the
C* control-file data area is retrieved with the reserved word *LOCK
C* to give this program exclusive use of it until the job ends.
C* Indicator 99 is then set on (equal to 1) to prevent this
C* routine from being processed on any other cycles.
C*****
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C          *IN99      IFEQ '0'
C                      OPEN TRWEEKL
C          *NAMVAR    DEFN CTLFIL    CTLDS
C          *LOCK      IN   *NAMVAR
C                      MOVE '1'      *IN99
C                      END
C*****
C* MAINLINE PROCESSING: The employee master is processed using the
C* RPG/400 program cycle. For each employee record read, at least
C* one entry should be in the transaction weekly file. If no
C* record is found in the transaction weekly, a message is sent to
C* the employee.
C*****

```

Figure 186 (Part 3 of 6). PRG05 program

Weekly Time File Update

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C* Using the employee number EMPNO, the transaction file is posi-
C* tioned at the next record with an employee number greater than the
C* employee record being processed. Then using the operation code
C* REDPE, the next prior sequential record is read. If the employee
C* number of the record read does not match the employee number from
C* the employee master, indicator 51 is set on (equal to 1).
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          EMPNO      SETGTRCWEK          50
C          EMPNO      REDPERCWEK          51
C* The following code is processed if indicator 51 is set on by the
C* REDPE operation. The employee is sent a message, stating that
C* their time entries are missing. To send the message, the program
C* passes control to a separate program that passes the employee
C* user ID to the called program. The PLIST operation contains
C* the parameter list name PLIST1 as factor 1 and is followed by the
C* PARM operation specifying the user ID field EUSRI.
C*
C* This operation could have been coded by simply placing the PARM
C* entries immediately after the CALL operation, but is coded this
C* way to illustrate the PLIST operation code. The program passes
C* control to program PROC5, which is a control language program for
C* sending the employee message. When this program receives control
C* back, indicator 60 is set on (equal to 1) to indicate that all
C* time entries have not been made and that program PROC5 is removed
C* from the list of activated programs by using the FREE operation.
C          *IN51      IFEQ '1'
C          PLIST1     PLIST
C                   PARM          EUSRI
C                   CALL 'PROC5'  PLIST1
C                   MOVE '1'      *IN60
C                   FREE 'PROC5'
C                   END
```

Figure 186 (Part 4 of 6). PRG05 program

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The last record indicator LR is set on by the RPG/400 program
C* cycle when the last employee master record has been read.
C* When LR is on, error indicator 60 is checked, and if on
C* (equal to 1), the person who asked of this program is sent message 1
C* from the message array MESS, using the DSPLY operation code.
C* The person then enters a response which is received into the
C* result field MRESP. The all time entries made flag field is
C* updated to N, and the data area is updated using the OUT operation.
C* If *IN60 is off, the ELSE operation is processed and the person
C* is sent message 2 from the message array MESS by the program using
C* the DSPLY operation code. The person must press Enter to
C* continue the job. The all time entries made flag field is
C* updated to Y and the data area is updated using the OUT operation.
C* If the MRESP field contains a Y, indicating the job continues,
C* external indicator U8 is set on (equal to 1). The UNLCK operation,
C* with *NAMVAR specified in factor 2, unlocks all data areas in the
C* program. The CLOSE operation is then processed to close the weekly
C* transaction file TRWEEKL.
C*
C* Note: If factor 1 on the OUT operation were blank, the data area
C*       would be unlocked as part of the function of the operation;
C*       however, the RPG reserved word *LOCK is coded to illustrate
C*       the UNLCK operation code.
C*

```

Figure 186 (Part 5 of 6). PRG05 program

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CLON01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
CLR          *IN60      IFEQ '1'
CLR          MOVE MESS,1    EMESS  50
CLR          EMESS      DSPLY '*EXT'  MRESP  1
CLR          MOVE 'N'      CALLE
CLR          *LOCK      OUT  *NAMVAR
CLR          ELSE
CLR          MOVE MESS,2    EMESS  50
CLR          EMESS      DSPLY '*EXT'  MRESP
CLR          MOVE 'Y'      CALLE
CLR          MOVE 'Y'      MRESP
CLR          *LOCK      OUT  *NAMVAR
CLR          END
C*
CLR          MRESP      IFEQ 'Y'
CLR          MOVE '1'      *INU8
CLR          END
CLR          UNLCK*NAMVAR
CLR          CLOSETRWEEL

```

0*

0* The compile time array MESS is entered below. The array is
0* preceded by ** to denote the beginning of the array.

0*

**** MESS - requestor messages**

Time entries missing. "Y"-continue "C"-cancel

No time entries missing. Press enter to continue.

Figure 186 (Part 6 of 6). PRG05 program

Weekly Employee Transaction Report Layout - PRG09

The weekly employee transaction report lists all time entry transactions by actual date worked within employee number. On a change of employee number, the total employee project, non-project and weekly hours are printed. A final report shows total project, non-project and weekly hours as well as an employee count for the week.

Figure 187 shows the weekly Employee Transaction Entry Report. The alphanumeric fields in the report are represented by a string of As, numeric fields are represented by a string of 9s, and dates are represented by MM/DD/YY. Program PRG09 is an SAA compatible program and the report specifications are program-described. Refer to the output specifications of program PRG09 for a detailed description of the report.

```

0000000000111111112222222222333333333344444444445555555555666666666677777777778888888888999999999900000000011111
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234

```

```

001
002
003 PRG09 EMPLOYEE TRANSACTION ENTRY PAGE 9999
004 FOR THE WEEK ENDING AAAAAAAAA 99, 1999 MM/DD/YY
005 EMPLOYEE NUMBER 999999 NAME AAAAAAAAAAAAAAAAAAAAAAAAAAAAA DEPARTMENT AAAAA
006
007 PROJECT REASON DESCRIPTION ACTUAL DATE HOURS
008 CODE CODE WORKED WORKED
009 AAAAAAAAA AAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA MM/DD/YY 9999.9-
010 AAAAAAAAA AAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA MM/DD/YY 9999.9-
011 AAAAAAAAA AAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA MM/DD/YY 9999.9-
012 AAAAAAAAA AAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA MM/DD/YY 9999.9-
013
014 EMPLOYEE TOTALS: PROJECT HOURS 9999.9-
015 NON PROJECT HOURS 9999.9-
016 WEEKLY TOTAL HOURS 9999.9-
017
018 REPORT TOTALS: PROJECT HOURS 99999999.9-
019 NON PROJECT HOURS 99999999.9-
020 WEEKLY TOTAL HOURS 99999999.9-
021 EMPLOYEE COUNT 99999
022
023

```

Figure 187. Weekly Employee Transaction Entry Report Layout - PRG09

Master File Update and Weekly Transaction Report - PRG09

Program PRG09 processes the weekly time entry transaction file TRWEEK to update the employee master, project master, and reason code master files, and to produce the weekly employee transaction summary report. The program is SAA compatible using program-described files.

Figure 188 shows the RPG/400 program PRG09 with embedded comments to explain the logic flow and use of various RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG09 - Time Reporting Master File Update
F* DESCRIPTION - This program updates the master files with the
F*                weekly transaction entries and produces the
F*                employee weekly transaction detail report.
F*                This program is SAA compatible.
F*****
F* INDICATORS USED:
F* 40 - Entry found on table look up
F* 50 - Invalid or missing employee record
F* 51 - Invalid or missing project code or reason code record
F* 69 - Exception output - heading lines
F* 70 - Exception output - employee heading line
F* 71 - Exception output - update project master
F* 72 - Exception output - update reason code master
F* 73 - Exception output - detail print line
F* 74 - Exception output - employee total line
F* 75 - Exception output - report total lines
F* 76 - Exception output - update employee master
F* L1 - Control level on employee number
F*****
```

Figure 188 (Part 1 of 16). Sample RPG/400 Program - PRG09

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* SUBROUTINES USED:
F* L1CLR - Control level detail time clear of work fields
F* UPDSR - Project and reason code master update and detail print
F* TOTL1 - Control-level total-time employee master update and
F*         total time print
F*****
F* This program uses program-described files. Files
F* used are: TRWEEK - weekly transaction entry file
F*           EMPMST - employee master file
F*           PRJMST - project master file
F*           RSNMST - reason code master file
F*           QSYSVRT - printer file
F*****
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTRWEEK  IP  F      53          DISK
FEMPMST  UF  F     103  4PI      2 DISK
FPRJMST  UF  F     120  8AI      2 DISK
FRSNMST  UF  F      73  8AI      2 DISK
FQSYSVRT 0   F     132    OF      PRINTER

```

Figure 188 (Part 2 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* The following extension specification describes the compile-time
E* table TABMTH. This table contains an entry for each month of
E* the year with the alternating entry TABNAM containing the month's
E* descriptive name. The table is accessed to provide the month
E* name in the heading line date field.
E*
E...FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E
                  TABMTH 1 12 2 0 TABNAM 9

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The weekly transaction entry file containing all time entries
I* is processed by employee number with control-level indicator L1
I* defined to control processing on a change of employee number.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
ITRWEK NS 01
I.....PFromTo++DField+L1M1FrP1MnZr...*
I                  P 2 50EMPNO L1
I                  6 13 EUSRI
I                  20 250CWKDT
I                  14 190ACDAT
I                  32 39 PRCDE
I                  40 47 RSCDE
I                  P 48 501EHWRK
I                  P 51 530TFRRN
I*

```

Figure 188 (Part 3 of 16). Sample RPG/400 Program - PRG09


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* Employee master is accessed randomly when a control break occurs.
I* The current month project hours (EPHRC) and the current month
I* reason code hours (ENPRC) are updated. Record identifying
I* indicator 02 is set on if the employee record read contains an A
I* in position 1, indicating an active record. If position 1 is not
I* an A, record identifying indicator 03 is set on. The RPG/400
I* program cycle sets these indicators on and off.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IEMPST NS 02 1 CA
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          6 35 ENAME
I          37 41 EDEPT
I          P 82 841EPHRC
I          P 93 951ENHRC
I      NS 03
I*
I* Project master is accessed randomly for each transaction read
I* if the project in the transaction is not blank. The current
I* month project hours (PRHRC) is updated. Record identifying
I* indicator 04 is set on if the project record read contains an A
I* in position 1, indicating an active record. If position 1 is not
I* an A, record identifying indicator 05 is set on. The RPG/400
I* program cycle sets these indicators on and off.
I*
IPRJMST NS 04 1 CA
I          10 59 PRDSC
I          P 107 1101PRHRC
I      NS 05

```

Figure 188 (Part 4 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I* Reason code master is accessed randomly for each transaction
I* read if the reason code in the transaction is not blank. The
I* current month reason code hours (RSHRC) is updated. Record
I* identifying indicator 06 is set on if the project record read
I* contains an A in position 1, indicating an active record. If
I* position 1 is not an A, record identifying indicator 07 is set on.
I* The RPG/400 program cycle sets these indicators on and off.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IRSNMST NS 06 1 CA
I.....PFromTo++DField+L1M1FrP1MnZr...*
I 10 59 RSDSC
I P 60 631RSHRC
I NS 07
I* The following named constants define edit words for the weekly
I* employee transaction entry report.
I*
I.....Constant+++++++C.....Field+.....*
I ' 0. -' C EDTHR1
I ' 0. -' C EDTHR2
I* The control-file data area contains the week ending date that
I* is used in the report headings and for accessing the month
I* descriptive name from the table TABMTH.
I*
IDsname....NODsExt-file++.....OccrLen+.....*
ICTLFIL UDS
I.....Ext-field+.....PFromTo++DField+.....*
I 1 6 CTCDE
I 7 120WKEND
I 7 80WKMTH
I 9 100WKDAY
I 11 120WKYR
I 13 180CMTDT
I 19 19 CALLE

```

Figure 188 (Part 5 of 16). Sample RPG/400 Program - PRG09

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C* First cycle processing. The following code is processed on the
C* first cycle only. Indicator 99 (*IN99) is off (equal to 0) on
C* the first cycle and the code following the IFEQ operation is
C* processed. The data area data structure CTLFIL is implicitly
C* retrieved by the RPG/400 program. Using the month field WKMTH
C* from the data area, the LOKUP operation is performed to retrieve
C* the month descriptive name from table file TABMTH. The alternating
C* entry TABNAM is moved to report heading field MNAME if the look up
C* is successful (40 is on). If not, the literal UNKNOWN is moved to
C* MNAME. Work fields used for report totals are initialized to zero
C* by the Z-ADD operation and indicator 99 is set on to prevent this
C* routine from being processed in subsequent cycles.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN99      IFEQ '0'
C          WKMTH      LOKUPTABMTH   TABNAM  9      40
C  40          MOVELTABNAM   MNAME   9
C N40          MOVEL 'UNKNOWN' MNAME
C          Z-ADD0      PRTOT   91
C          Z-ADD0      RSTOT   91
C          Z-ADD0      WKTOT   91
C          Z-ADD0      EMCNT   50
C          EMPNO      CHAINEMPST                50
C          *IN50      IFEQ '0'
C          *IN02      ANDEQ '0'
C          MOVE '1'      *IN50
C          END
C          MOVE '1'      *IN99
C          END

```

C* The RPG/400 program cycle controls the reading of the
C* transaction file and the setting on of last record indicator LR.
C* This is controlled by defining the TRWEEK file as P (primary)
C* in position 16 of the file specification.

Figure 188 (Part 6 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* MAINLINE: The mainline consists of three EXSR operations and
C* last record (LR) processing. The first routine is processed at
C* control-level detail time. Control level detail time happens on
C* the initial RPG/400 program cycle and on the first record of each
C* control group. In other words, when the employee number changes,
C* the L1CLR routine is processed before processing is done on the
C* new employee group. The second routine is processed on each
C* RPG/400 detail cycle to accumulate employee totals and update
C* the project and reason code master files. The third routine is
C* processed at control-level total time. Control level total time
C* happens when the last record of the control group has been read
C* and on last record.
C*
C* The final three lines of code in the mainline is processed on
C* the last record only. Indicator 69 is set on to skip to new
C* page and print headings, and indicator 75 is set on to print
C* the report totals. The EXCPT operation is used to process
C* exception output. The indicators are set off after the EXCPT
C* operation to prevent the same output from being done by
C* subsequent EXCPT operations.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C  L1          EXSR L1CLR
C  01          EXSR UPDSR
CL1          EXSR TOTL1
CLR          SETON          6975
CLR          EXCPT
CLR          SETOF          6975
C*
```

Figure 188 (Part 7 of 16). Sample RPG/400 Program - PRG09

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* L1CLR subroutine is processed at control level detail time. The
C* employee total work fields are cleared by the program using the
C* Z-ADD operation. Using the employee number and the CHAIN
C* operation, the program retrieves the employee master record. If
C* the record is not found, resulting indicator 50 is set on. If the
C* record is found (*IN50 equals 0), but the record identifying
C* indicator 02 is not on (*IN02 equals 0), indicator 50 is set on.
C* Indicator 50 controls the printing of the employee name, employee
C* department, and the updating of the employee master at total time.
C* Indicator 69 is set on to skip to a new page and print the
C* heading lines. Indicator 70 is set on to print the employee
C* heading line. The EXCPT operation is run to process exception
C* output, and indicators 69 and 70 are set off.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          L1CLR          BEGSR
C          Z-ADD0          PRHRS    51
C          Z-ADD0          RSHRS    51
C          Z-ADD0          WKHRS    51
C          EMPNO          CHAINEMPST          50
C          *IN50          IFEQ '0'
C          *IN02          ANDEQ '0'
C          MOVE '1'          *IN50
C          END
C          SETON          6970
C          EXCPT
C          SETOF          6970
C          ENDSR
C*

```

Figure 188 (Part 8 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The UPDSR subroutine is processed for each detail record. The
C* work field DESC is cleared first by moving blanks to it. This
C* prevents a description from a previous record being printed when
C* both project code and reason code records cannot be found in the
C* master files. If the project code is not equal to blanks, the
C* project master is accessed using PRCDE and the CHAIN operation.
C* If the record is not found, resulting indicator 51 is set on.
C* If the record is found (*IN50 equals 0) and the record identifying
C* indicator 04 is on (*IN04 equals 1), the transaction hours are
C* added to the current month project hours PRHRC and to the employee
C* project hours work field PRHRS and the project description is
C* moved to work field DESC. Indicator 71 is set on to update the
C* project master and indicator 73 is set on to print the detail line.
C* The EXCPT operation is run to process exception output and
C* indicators 71 and 73 are set off.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          UPDSR      BEGSR
C          MOVE *BLANKS  DESC
C          PRCDE      IFNE *BLANKS
C          PRCDE      CHAINPRJMST          51
C          *IN51      IFEQ '0'
C          *IN04      ANDEQ'1'
C          EHWRK      ADD PRHRC      PRHRC
C          EHWRK      ADD PRHRS      PRHRS
C          MOVE PRDSC  DESC      50
C          SETON
C          EXCPT
C          SETOF      7173
C          END
C          END
C*

```

Figure 188 (Part 9 of 16). Sample RPG/400 Program - PRG09

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* If the reason code is not equal to blanks, the reason code master
C* is accessed using RSCDE and the CHAIN operation. If the record
C* is not found, resulting indicator 51 is set on. If the record
C* is found (*IN50 equals 0) and the record identifying indicator 06
C* is on (*IN06 equals 1), the transaction hours are added to the
C* current month reason code hours RSHRC and to the employee reason
C* code hours work field RSHRS and the reason code description is
C* moved to work field DESC. Indicator 72 is set on to update the
C* reason code master and indicator 73 is set on to print the detail
C* line. The EXCPT operation is run to process exception output
C* and indicators 72 and 73 are set off.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          RSCDE      IFNE *BLANKS
C          RSCDE      CHAINRSMST          52
C          *IN52      IFEQ '0'
C          *IN06      ANDEQ'1'
C          EHWRK      ADD  RSHRC          RSHRC
C          EHWRK      ADD  RSHRS          RSHRS
C          MOVE RSDSC          DESC
C          SETON          7273
C          EXCPT
C          SETOF          7273
C          END
C          END
C          ENDSR
C*

```

Figure 188 (Part 10 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The TOTL1 subroutine is processed on control level total time.
C* The employee work field totals are added to the report work field
C* totals. Indicator 74 is set on and the EXCPT operation is run
C* to print the employee totals and indicator 74 is set off. If
C* indicator 50 is off (*IN50 equals 0, employee record found), the
C* employee weekly project hours total PRHRS is added to the employee
C* master record field EPHRC, and the weekly reason code hours total
C* RSHRS is added to ENHRC. Indicator 76 is set on and the EXCPT
C* operation is run to update the employee master and indicator 76
C* is then set off.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          TOTL1      BEGSR
C          PRHRS      ADD  PRTOT      PRTOT
C          RSHRS      ADD  RSTOT      RSTOT
C          PRHRS      ADD  RSHRS      WKHRS
C          WKHRS      ADD  WKTOT      WKTOT
C          EMCNT      ADD  1          EMCNT
C                               SETON          74
C                               EXCPT
C                               SETOF          74
C          *IN50      IFEQ '0'
C          PRHRS      ADD  EPHRC      EPHRC
C          RSHRS      ADD  ENHRC      ENHRC
C                               SETON          76
C                               EXCPT
C                               SETOF          76
C                               END
C                               ENDSR

```

Figure 188 (Part 11 of 16). Sample RPG/400 Program - PRG09


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0*
0* This program uses exception output for all its output operations.
0* The following code describes the printer file QSYSPRT contents,
0* spacing and skipping. The first two exception groups are printed
0* when indicator 69 is on. The first exception causes a skip to
0* line 03 of a new page, and the second exception spaces one line
0* before printing and one line after printing. RPG reserved words
0* PAGE is used to handle page numbering and UDATE to print the
0* system date.
0*
0Name++++DFBASbSaN01N02N03Excnam.....*
QSYSPRT E 03 69
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0
0          6 'PRG09'
0          61 'EMPLOYEE TRANSACTION'
0          67 'ENTRY'
0          105 'PAGE'
0          PAGE Z 110
0          E 11      69
0          55 'FOR THE WEEK ENDING'
0          MNAME      65
0          WKDAY      68
0          72 ', 19'
0          WKYR      74
0          UDATE Y 110
0*

```

Figure 188 (Part 12 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0*
0* The following three exception lines are controlled by indicator
0* 70 and print additional heading information. The first exception
0* line prints the employee information. If the employee record is
0* not found (indicator 50 is on), the employee name is replaced by
0* the error message. The next two exception lines print headings
0* for the detail lines.
0*
0Name++++DFBASbSaN01N02N03Excnam.....*
0      E 11      70
0.....N01N02N03Field+YBEnd+PConstant/editword+++++...*
0                                20 'EMPLOYEE NUMBER'
0                                EMPNO Z  28
0                                39 'NAME'
0                                N50      ENAME  71
0                                50      66 'EMPLOYEE NUMBER INVALID'
0                                88 'DEPARTMENT'
0                                N50      EDEPT  95
0      E 1      70
0                                22 'PROJECT          REASON'
0                                40 'DESCRIPTION'
0                                108 'ACTUAL DATE          HOURS'
0      E 1      70
0                                21 'CODE          CODE'
0                                108 'WORKED          WORKED'
0*

```

Figure 188 (Part 13 of 16). Sample RPG/400 Program - PRG09

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0*
0* The following exception line is controlled by indicator 73 and
0* prints each transaction detail.
0*
0Name++++DFBASbSaN01N02N03Excnam.....*
0      E 1      73
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0          PRCDE      9
0          RSCDE     23
0          DESC      79
0          ACDAT Y   94
0          EHWRK    108 EDTHR1
0*
0* The following three exception lines are controlled by indicator
0* 74 and print on a change of employee number or control break.
0*
0      E 2      74
0          78 'EMPLOYEE TOTALS:'
0          93 'PROJECT HOURS'
0          PRHRS    108 EDTHR1
0      E 1      74
0          91 'NON PROJECT HOURS'
0          RSHRS    108 EDTHR1
0      E 1      74
0          98 'WEEKLY TOTAL HOURS'
0          WKHRS    108 EDTHR1
0*

```

Figure 188 (Part 14 of 16). Sample RPG/400 Program - PRG09

Weekly Time File Update

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0*
0* The following four exception lines are controlled by indicator
0* 75 and print at end of file or last record.
0*
0Name+++DFBASbSaN01N02N03Excnam.....*
0      E 2      75
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0              71 'REPORT TOTALS:'
0              87 'PROJECT HOURS'
0              PRTOT 108 EDTHR2
0      E 1      75
0              91 'NON PROJECT HOURS'
0              RSTOT 108 EDTHR2
0      E 1      75
0              92 'WEEKLY TOTAL HOURS'
0              WKTOT 108 EDTHR2
0      E 1      75
0              88 'EMPLOYEE COUNT'
0              EMCNT Z 108
0*
```

Figure 188 (Part 15 of 16). Sample RPG/400 Program - PRG09

.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..

0*
 0* The following exception line is controlled by indicator 71 and
 0* updates the project master file record.

0*
 0Name++++DFBASbSaN01N02N03Excnam.....*
OPRJMST E 71

0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0 PRHRC 110P

0* The following exception line is controlled by indicator 72 and
 0* updates the reason code master file record.

ORSNMST E 72
0 RSHRC 63P

0* The following exception line is controlled by indicator 76 and
 0* updates the employee master file record.

OEMPST E 76
0 EPHRC 84P
0 ENHRC 95P

* ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7

0* The following table contains descriptions for each month
 0* of the year. The month number is used as the look up to
 0* retrieve the month description - the alternating table
 0* element. The table begins in column 1 of the output
 0* specification.
 0*

**** TABMTH - Month Description Table**

- 01JANUARY**
- 02FEBRUARY**
- 03MARCH**
- 04APRIL**
- 05MAY**
- 06JUNE**
- 07JULY**
- 08AUGUST**
- 09SEPTEMBER**
- 10OCTOBER**
- 11NOVEMBER**
- 12DECEMBER**

Figure 188 (Part 16 of 16). Sample RPG/400 Program - PRG09

Monthly Processing

All the master files are processed after the last weekly update for the month to produce monthly reports, add current month values to the year-to-date values, and prepare transaction files for new month processing.

Technical design for each step in the monthly process contains all or part of the following:

- Display format layout
- Display file data descriptions
- Program code and narratives
- Printer spacing chart.

Monthly Time File Update and Reporting

Figure 189 shows the Time Reporting System Main Menu. The first step in the monthly update is to change the month end date in the control file using option 3. After the control file has been updated, you call the monthly update by entering option 5 (Monthly time file update & reporting). Option 5 calls PROC3, prompting if the run is for year end. You must enter a Y or an N. PROC3 then submits PROC4 to batch for processing. See Figure 190 on page 445.

- CALL PGM(PROC3)

```
TMENU                                Time Reporting System
                                      Main Menu

      1. Master file maintenance              (PRG01)
      2. Control file maintenance            (PRG02)
      3. Time file transaction entry          (PRG03)
      4. Weekly time file update             (PROC1)
      5. Monthly time file update & reporting (PROC3)

      8. Display messages                    (DSPMSG)
      9. Sign off                            (SIGNOFF)

Selection or command
====>
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support   F16=System main menu
```

Figure 189. Time Reporting Menu

The monthly time file update consists of two control level programs:

- CL program PROC3 is an interactive program that prompts you for a Y or an N response for year end processing. The program accepts an uppercase or a lowercase response. The program then submits PROC4 to batch for processing.
- CL program PROC4 is a batch job that produces the monthly employee, project, and reason code summary reports. It also prepares the master files

and transaction files for new month processing (and new year, if year end has been requested).

```

/* Monthly Time File Update and Reporting: */
/* This procedure is the first step in the monthly time */
/* reporting update. The program sends a message prompting */
/* if this run is for year end. If the run is for year end, */
/* the CHGJOB command sets on job switch 4. The update program */
/* The update program PROC4 is then submitted to batch. */
/* */
BEGIN:      PGM
           DCL          &REPLY *CHAR LEN(1)
           SNDUSRMSG   MSG('Update for year end Y or N') +
                MSGRPY(&REPLY)
           IF          COND(&REPLY *EQ Y) THEN(DO)
           SBMJOB      CMD(CALL PGM(PROC4)) JOB(PROC4) SWS(00010000)
           ENDDO
           ELSE
           IF          COND(&REPLY *NE Y) THEN(DO)
           SBMJOB      CMD(CALL PGM(PROC4)) JOB(PROC4) SWS(00000000)
           ENDDO
/* */
ENDIT:      ENDPGM

```

Figure 190. CL Program PROC3

Time Reporting Monthly Update

```
/* Monthly Time File Update and Reporting: */
/* This procedure is run monthly to produce the monthly */
/* employee, project and reason code reports and to prepare */
/* the master files and transaction files for new month */
/* processing. */
/* */
/* Program PRG06 reads the monthly transaction file to produce */
/* the employee time entry report. */
/* */
BEGIN:      PGM
            RTVJOBA
            CALL      PGM(PRG06)
/* */
/* Program PRG07 reads the monthly transaction file to produce */
/* the project time entry report. */
/* */
PRG07:      CALL      PGM(PRG07)
/* */
/* Program PRG08 reads the monthly transaction file to produce */
/* the reason code time entry report. */
/* */
PRG08:      CALL      PGM(PRG08)
/* */
/* This step adds the current month hours to the year-to-date */
/* hours and clears the month-to-date field.  If this is a year */
/* end run, the year-to-date is rolled to the prior year-to-date */
/* and the year-to-date is cleared.  The step loops three times. */
/* Each time program PRG04 is called, the opened and updated */
/* file is controlled by the external indicator set on by the */
/* CHGJOB command. */
/* U1 - Employee master */
/* U2 - Project master */
/* U3 - Reason code master */
/* */
```

Figure 191 (Part 1 of 2). CL Program PROC4


```

PRG04:    CHGJOB    SWS(100X0000)
          CALL     PGM(PRG04)
          IF       COND(%SWITCH(100X0000)) THEN(DO)
                    CHGJOB SWS('010X0000')
                    GOTO   CMDLBL(PRG04)
                    ENDDO

          ELSE
          IF       COND(%SWITCH(010X0000)) THEN(DO)
                    CHGJOB SWS('001X0000')
                    GOTO   CMDLBL(PRG04)
                    ENDDO

/*                                               */
/* CLEAR step clears the monthly transaction file in */
/* preparation for new month activity.                */
/*                                               */
CLEAR:    CLRPFM    FILE(TRMNTH)
/*                                               */
ENDIT:    ENDPGM

```

Figure 191 (Part 2 of 2). CL Program PROC4

Time Reporting Employee Summary Report Layout - PRG06RP

The Time Reporting Employee Summary report lists all time entry transactions for the employee for the month. The report is organized by week ending date for an employee number. Subtotals are printed on a change of week ending date and an employee summary is printed on a change of employee number. The employee summary includes percentage calculations to show what portion of the employee's time is spent on project related and non-project related tasks.

Figure 192 shows the Time Reporting Employee Summary report. The alphanumeric fields defined in the report are represented by a string of As, numeric fields are represented by a string of 9s, and dates are represented by MM/DD/YY. See Figure 193 on page 449 for the data description specifications for report PRG06RP.

	0	0	0	0	0	0	0	0	0	1	1
	000000000111111111222222222233333333334444444444555555555566666666667777777777888888888899999999990000000001111111111										
001											
002											
003	PRG06RP			TIME REPORTING EMPLOYEE SUMMARY					PAGE 9999		
004				FOR THE PERIOD ENDED MM/DD/YY					MM/DD/YY		
005											
006	EMPLOYEE NUMBER	999999		EMPLOYEE NAME	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA			DEPARTMENT	AAAA		
007											
008	PROJECT	REASON		DESCRIPTION				ACTUAL DATE	HOURS		
009	CODE	CODE						WORKED	WORKED		
010	AAAAAAAA	AAAAAAAA		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA				MM/DD/YY	9999.9-		
011	AAAAAAAA	AAAAAAAA		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA				MM/DD/YY	9999.9-		
012											
013								WEEKLY TOTAL HOURS	9999.9-		
014											
015	AAAAAAAA	AAAAAAAA		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA				MM/DD/YY	9999.9-		
016	AAAAAAAA	AAAAAAAA		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA				MM/DD/YY	9999.9-		
017											
018								WEEKLY TOTAL HOURS	9999.9-		
019											
020											
021				CURRENT	% OF	YEAR TO	% OF				
022				MONTH	TOTAL	DATE	TOTAL				
023	EMPLOYEE SUMMARY:	PROJECT HOURS		9999.9-	9999.9-	999999.9-	9999.9-				
024		NON PROJECT HOURS		9999.9-	9999.9-	999999.9-	9999.9-				
025		TOTAL HOURS		9999.9-	9999.9-	999999.9-	9999.9-				

Figure 192. Time Reporting Employee Summary Report Layout - PRG06RP

Employee Summary Report Data Descriptions - PRG06RP

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* This print file describes the format for the monthly time
A* reporting employee summary report. In this printer file are
A* four record formats, identified by R in position 17 followed by
A* the format name in positions 19 through 20. The following
A* keywords are used:
A* EDTCDE(a) - Edits output capable numeric fields.
A* PAGNBR - Specifies a four digit, zoned decimal field to
A* contain the page number.
A* REF(REFMST) - Lines containing an R in position 29 use the
A* attributes from a previously defined field in
A* this reference file.
A* SKIPB(n) - Specifies that the printer device is to skip to
A* a specific line before it prints the next line.
A* SPACEA(n) - Specifies that the printer device is to space (n)
A* lines after it prints one or more lines.
A* SPACEB(n) - Specifies that the printer device is to space (n)
A* lines before it prints the next line or lines.
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A REF(REFMST)
A*
A* The first format, TITLE1, contains the definition for the
A* heading lines of the report. The format is written on the
A* first cycle, on a change of employee number, or when overflow
A* occurs while printing details for an employee.
A*

```

Figure 193 (Part 1 of 4). Employee Summary Report Data Descriptions - PRG06RP

Monthly Processing

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R TITLE1                                SKIPB(3)
A          2'PRG06RP'
A          37'TIME REPORTING EMPLOYEE SUMMARY'
A          90'PAGE'
A          95PAGNBR
A          SPACEA(1)
A          38'FOR THE PERIOD ENDED'
A          CMTDT      R          59EDTCDE(Y)
A          RDATE      6S 0      91EDTCDE(Y)
A          SPACEA(2)
A          2'EMPLOYEE NUMBER'
A          EMPNO      R          19EDTCDE(Z)
A          31'EMPLOYEE NAME'
A N60          ENAME      R          46
A 60          46'INVALID EMPLOYEE NUMBER'
A          82'DEPARTMENT'
A N60          EDEPT      R          94
A          SPACEA(2)
A          2'PROJECT      REASON'
A          26'DESCRIPTION'
A          78'WEEK ENDING      HOURS'
A          SPACEA(1)
A          3'CODE          CODE'
A          81'DATE          WORKED'
A          SPACEA(1)

```

Figure 193 (Part 2 of 4). Employee Summary Report Data Descriptions - PRG06RP

.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...

A*

A* The second format, DETAIL, contains the definition for the detail
A* print lines. The format is written for each detail record in
A* the monthly transaction file.

A*

AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*

```

A          R DETAIL
A          PRCDE      R          2
A          RSCDE      R          14
A N61N62    RDESC          50      26
A 61                26'INVALID PROJECT CODE'
A 62                26'INVALID REASON CODE'
A          CWKDTX      R          79REFFLD(CWKDT)
A                                EDTCDE(Y)
A          EHWRK      R          93EDTCDE(L)
A                                SPACEA(1)

```

A*

A* The third format, TOTL1, contains the definition for total time
A* level break L1. The format is written on a change of week
A* ending date or a change of employee number.

A*

```

A          R TOTL1                SPACEB(1)
A                                71'WEEKLY TOTAL HOURS'
A          WKTOT          5S 1    93EDTCDE(L)
A                                SPACEA(2)

```

Figure 193 (Part 3 of 4). Employee Summary Report Data Descriptions - PRG06RP

Monthly Processing

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*
A* The fourth format, TOTL2, contains the definition for total
A* time level break L2. The format is written on a change of
A* employee number.
A*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A      R TOTL2
A      44'CURRENT          % OF'
A      67'YEAR TO          % OF'
A      SPACEA(1)
A      45'MONTH          TOTAL'
A      69'DATE            TOTAL'
A      SPACEA(1)
A      2'EMPLOYEE SUMMARY'
A      23'PROJECT HOURS'
A      PRMTH              5S 1  44EDTCDE(L)
A      PCMTH              5S 1  55EDTCDE(L)
A      PRYER              7S 1  66EDTCDE(L)
A      PCYER              5S 1  79EDTCDE(L)
A      SPACEA(1)
A      23'NON PROJECT HOURS'
A      NPMTH              5S 1  44EDTCDE(L)
A      NCMTH              5S 1  55EDTCDE(L)
A      NPYER              7S 1  66EDTCDE(L)
A      NCYER              5S 1  79EDTCDE(L)
A      SPACEA(1)
A      23'TOTAL HOURS'
A      TOMTH              5S 1  44EDTCDE(L)
A      TCMTH              5S 1  55EDTCDE(L)
A      TOYER              7S 1  66EDTCDE(L)
A      TCYER              5S 1  79EDTCDE(L)
A      SPACEA(1)

```

Figure 193 (Part 4 of 4). Employee Summary Report Data Descriptions - PRG06RP

Employee Summary Report RPG/400 Program - PRG06

Figure 194 shows RPG/400 program PRG06 with embedded comments to explain the logic flow and use of various RPG functions and operation codes.

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG06 - Time Reporting Employee Summary Report
F* DESCRIPTION - This program produces the time reporting employee
F*                summary report. All time entries for the month
F*                are printed by week ending date with subtotals by
F*                week and an employee summary showing month and
F*                year-to-date totals.
F*****
F* This program uses externally described files. Files used are:
F* TRMNTHL - Logical view of TRMNTH, monthly transaction file
F*                by employee number and week ending date.
F* EMPMST  - Employee master file
F* PRJMST  - Project master file
F* RSNMST  - Reason code master file
F* PRG06RP - Employee summary report file
F*****
F* INDICATORS USED:
F* 60 - Employee master record not found
F* 61 - Project master record not found
F* 62 - Reason code master record not found
F* 99 - First cycle processing
F* L1 - Control level on week ending date
F* L2 - Control level on employee number
F*****
F* SUBROUTINES USED:
F* DTLSR  - Detail calculations routine
F* LICHK  - Line count check routine
F* L2CLR  - Clear work fields at detail time L2
F* SUBRL1 - Total time calculations - change of week ending date
F* SUBRL2 - Total time calculations - change of employee number

```

Figure 194 (Part 1 of 10). Sample RPG/400 Program - PRG06

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTRMNTHL IP E K DISK
FEMPMST IF E K DISK
FPRJMST IF E K DISK
FRSNMST IF E K DISK
FPRG06RP 0 E PRINTER
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E* The following arrays are used to store the weekly project and
E* reason code hours for the employee. Each array contains up to
E* five weekly totals.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E ARR 5 5 1
E ARR 5 5 1
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I* The following code renames the monthly transaction file input
I* field names. These fields appear in other data definitions
I* and are overlaid when those files are read, these renames
I* prevent the overlay.
I*
IRcdname+....In.....*
IRCMNTH
I.....Ext-field+.....Field+L1M1..PlMnZr...*
I EMPNO EMPNOXL2
I CWKDT CWKDTXL1
I CMTDT CMTDTX
I*
I* Externally described control file data area
I*
IDcname....NODsExt-file++.....OccrLen+.....*
ICTLFIL EUDS
I*
```

Figure 194 (Part 2 of 10). Sample RPG/400 Program - PRG06


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* FIRST CYCLE PROCESSING: Indicator 99 is set off (equal to 0)
C* on the first RPG/400 program cycle and the routine is processed.
C* The TIME operation retrieves the time of day and the system date
C* and places them in the result field TDATE. The time of day
C* occupies the first six positions and the system date the last
C* six positions of TDATE. The MOVE operation moves the last six
C* positions to the result field RDATE to provide the run date for
C* the report. The RPG reserved word UDATE could have been specified
C* on the output specifications to accomplish the same result.
C* Indicator 99 is then set on (equal to 1) to prevent this routine
C* from being processed on subsequent cycles.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          *IN99      IFEQ '0'
C          TIME              TDATE  120
C          MOVE TDATE      RDATE   60
C          MOVE '1'        *IN99
C          END

```

Figure 194 (Part 3 of 10). Sample RPG/400 Program - PRG06

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* MAINLINE: The mainline routine consists of four EXSR subroutines.
C* The first subroutine is processed at detail time when the
C* control level indicator L2 is on. This occurs on the first
C* RPG/400 program cycle and on the RPG/400 program cycle following
C* total time calculations. The L2CLR subroutine clears work fields
C* and writes report headings. The second subroutine, DTLSR, is
C* processed on each RPG/400 detail time cycle. The routine writes
C* detail report lines and accumulates data for total time printing.
C* The third and fourth subroutines are processed at total time.
C* The SUBRL1 subroutine is processed on a change of week ending
C* date and also on a change of employee number (RPG/400 logic sets
C* on all lower level control indicators when a control break occurs,
C* that is, when L2 is set on, so is L1). The SUBRL2 subroutine is
C* processed on a change of employee number.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C  L2                EXSR L2CLR
C                    EXSR DTLSR
CL1                EXSR SUBRL1
CL2                EXSR SUBRL2
C*****
C* DTLSR SUBROUTINE: This routine performs detail time operations.
C* Error indicators *IN61 and *IN62 are set off (equal to 0) as
C* part of housekeeping. If the project code PRCDE is not equal
C* to blanks, the hours worked are added to the current element of
C* the project array. The array is incremented each time the week
C* ending date changes, and is reset to 1 (the first element) when
C* the employee number changes. The project master file is
C* accessed using the CHAIN operation. If the record is not found,
C* indicator 61 is set on. If the record is found, the project
C* description is moved to the work field RDESC.
C*
```

Figure 194 (Part 4 of 10). Sample RPG/400 Program - PRG06

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          DTLR      BEGSR
C          MOVE '0'      *IN61
C          MOVE '0'      *IN62
C          PRCDE      IFNE *BLANKS
C          EHWRK      ADD  ARR,P      ARR,P
C          PRCDE      CHAINPRJMST          61
C          *IN61      IFEQ '0'
C          MOVE PRDSC      RDESC
C          END
C          ELSE
C*
C* The preceding ELSE statement denotes the end of the project code
C* operations. If the project code is equal to blanks, a reason
C* code must exist. The hours worked are added to the current
C* element of the non-project hours array and the reason code master
C* file is accessed using the CHAIN operation. If the record is
C* not found, indicator 62 is set on. If the record is found,
C* the reason code description is moved to the work field RDESC.
C*
C          EHWRK      ADD  ARR,N      ARR,N
C          RSCDE      CHAINRSNMST          62
C          *IN62      IFEQ '0'
C          MOVE RSDSC      RDESC
C          END
C          END

```

Figure 194 (Part 5 of 10). Sample RPG/400 Program - PRG06

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The preceding END statement denotes the end of the original IF
C* in this subroutine. The detail record has now been processed
C* and the program is ready to write the detail report line. The
C* WRITE statement writes the record format DETAIL in the externally
C* described printer file PRG06RP. The format contains one line and
C* the line counter is incremented by one. Each time an output
C* operation is performed to the printer file, subroutine LICK is
C* processed. This routine determines if page overflow processing
C* should be performed.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          WRITEDETAIL
C          ADD 1          LICNT
C          EXSR LICK
C          ENDSR
C*
C*****
C* SUBRL1 SUBROUTINE: This routine performs total time operations.
C* The project hours and non-project hours are added to provide the
C* total weeks hours using the current element to each array. The
C* line counter value is checked. If it is greater than or equal
C* to 59, it is set to 60 and the LICK overflow routine is
C* processed. The program performs these operations to ensure that
C* enough print lines are available on the page to print the weekly
C* total line. The print format TOTL1 is then written. The
C* project and non-project array indexes are incremented by one
C* for the next week's hours, and the line counter is incremented
C* by two for the print lines written in format TOTL1.
C*
```

Figure 194 (Part 6 of 10). Sample RPG/400 Program - PRG06

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          SUBRL1    BEGSR
C          ARRP,P    ADD  ARR,N    WKTOT
C          LICNT     IFGE 59
C          Z-ADD60    LICNT
C          EXSR LICHK
C          END
C          WRITETOTL1
C          ADD 1      P
C          ADD 1      N
C          ADD 2      LICNT
C          EXSR LICHK
C          ENDSR
C*
C*****
C* SUBRL2 SUBROUTINE: This routine performs total time operations.
C* The line counter value is checked. If it is greater than or
C* equal to 55, then it is set to 60 and the LICHK overflow routine
C* is processed. The program performs these operations to ensure
C* that enough print lines are available on the page to print the
C* employee total lines. The project and non-project arrays are
C* summed using the XFOOT operation. This operation adds all
C* elements of the array together and places the sum in the result
C* field. The series of arithmetic operations that follow prepare
C* the employee total line for printing. The total project hours
C* are added to the total non-project hours, the result is equal to
C* total month hours. The percentage of project hours of the total
C* is calculated by dividing total project hours by total hours and
C* multiplying the result (WRK1) by 100. The percentage of non-
C* project hours of the total is calculated by dividing total non-
C* project hours by the total hours and multiplying the result (WRK1)
C* by 100. The percent total field is set to 100 using the Z-ADD
C* operation.
C*

```

Figure 194 (Part 7 of 10). Sample RPG/400 Program - PRG06

Monthly Processing

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          SUBRL2    BEGSR
C          LICNT     IFGE 55
C          Z-ADD60    LICNT
C          EXSR LICHK
C          END
C          XFOOTARRP  PRMTH
C          XFOOTARRN  NPMTH
C          PRMTH     ADD  NPMTH  TOMTH
C          PRMTH     DIV  TOMTH  WRK1   53H
C          WRK1      MULT 100    PCMTH
C          NPMTH     DIV  TOMTH  WRK1    H
C          WRK1      MULT 100    NCMTH
C          Z-ADD100  TCMTH
C*
C* The following calculations add the current month to year-to-date
C* totals and performs the same expressions as for current month.
C*
C          PRMTH     ADD  EPHRY    PRYER
C          NPMTH     ADD  EPNRY    NPYER
C          PRYER     ADD  NPYER    TOYER
C          PRYER     DIV  TOYER    WRK1    H
C          WRK1      MULT 100    PCYER
C          NPYER     DIV  TOYER    WRK1    H
C          WRK1      MULT 100    NCYER
C          Z-ADD100  TCYER
C*

```

Figure 194 (Part 8 of 10). Sample RPG/400 Program - PRG06

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The employee total line is now ready for printing. The TOTL2
C* format is written. Because the program will perform detail
C* time L2 operations on the next cycle to prepare for the next
C* employee, the line counter field is not incremented and
C* overflow is not checked after writing the format
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          WRITETOTL2
C          ENDSR
C*
C*****
C* LICHK SUBROUTINE: This routine controls page overflow.
C* If the line count is greater than or equal to 60, the heading
C* format TITLE1 is written and the line count is set to 9.
C*
C          LICHK      BEGSR
C          LICNT      IFGE 60
C                   WRITETITLE1
C                   Z-ADD9          LICNT
C                   END
C                   ENDSR

```

Figure 194 (Part 9 of 10). Sample RPG/400 Program - PRG06

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* L2CLR SUBROUTINE: This routine prepares work fields and prints
C* heading lines before processing the first employee detail record.
C* The project hours array ARRP and the non-project hours array
C* ARRN are initialized to 0. The array elements are then set
C* to 1 for the first occurrence. The employee master file is
C* accessed using the employee number from the transaction record.
C* If the employee record is not found, indicator 60 is set on.
C* The report headings are printed by writing print format TITLE1
C* and the line counter is set to 9.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          L2CLR      BEGSR
C          Z-ADD0      ARRP
C          Z-ADD0      ARRN
C          Z-ADD1      P          10
C          Z-ADD1      N          10
C          EMPNOX     CHAINEMPST          60
C          WRITETITLE1
C          Z-ADD9      LICNT    30
C          ENDSR
```

Figure 194 (Part 10 of 10). Sample RPG/400 Program - PRG06

Time Reporting Project Summary Report Layout - PRG07RP

The Time Reporting Project Summary report lists all time entry transactions charged against a project code. The report is organized by employee number within week ending date within project code. Subtotals are printed on a change of week ending date and a project summary is printed on a change of project code.

Figure 195 shows the Time Reporting Project Summary Report. The alphanumeric fields defined in the report are represented by a string of As, numeric fields are represented by a string of 9s, and dates are represented by MM/DD/YY. See Figure 196 on page 464 for the data descriptions specifications for report PRG07RP.

	000000000111111111222222222233333333334444444444555555555566666666667777777777888888888899999999990000000001111111111				
	123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234				
001					
002					
003	PRG07RP		TIME REPORTING PROJECT SUMMARY		PAGE 9999
004			FOR THE PERIOD ENDED MM/DD/YY		MM/DD/YY
005					
006	PROJECT CODE	AAAAAAA	DESCRIPTION	AA	
007					
008	RESPONSIBILITY		START	ESTIMATED	COMPLETION
009			DATE	END DATE	DATE
010	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	MM/DD/YY	MM/DD/YY
011					ESTIMATED
012	EMPLOYEE	EMPLOYEE NAME		WEEK ENDING	HOURS
013	NUMBER			DATE	WORKED
014	999999	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	9999.9-
015	999999	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	9999.9-
016	999999	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	9999.9-
017					
018			WEEKLY TOTAL		999999.9-
019					
020	999999	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	9999.9-
021	999999	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA		MM/DD/YY	9999.9-
022					
023			WEEKLY TOTAL		999999.9-
024					
025	PROJECT SUMMARY:	CURRENT	CURRENT	PRIOR YEAR	TOTAL PROJECT
026		MONTH	YEAR TO DATE	TOTAL	HOURS
027		999999.9-	99999999.9-	99999999.9-	99999999.9-
028					% VARIANCE TO
					ESTIMATED HOURS
					9999.9-

Figure 195. Time Reporting Project Summary Report Layout - PRG07RP

Project Summary Report Data Descriptions - PRG07RP

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* This print file describes the format for the monthly time
A* reporting project summary report. In this printer file are
A* four record formats, identified by an R in position 17 followed
A* by the format name in positions 19 through 20. The following
A* keywords are used:
A* DATE - Specifies the system date
A* EDTCDE(a) - Edits output capable numeric fields
A* PAGNBR - Specifies a four digit, zoned decimal field to
A* contain the page number.
A* REF(REFMST) - Any lines containing the an R in position 29
A* uses the attributes from a previously defined
A* field in this reference file.
A* REFFLD - References a field to a previously defined field.
A* SKIPB(n) - Specifies that the printer device is to skip to
A* a specific line before it prints the next line.
A* SPACEA(n) - Specifies that the printer device is to space n
A* lines after it prints one or more lines.
A* SPACEB(n) - Specifies that the printer device is to space n
A* lines before it prints the next line or lines.
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A REF(REFMST)
A*
A* The first format, TITLE1, contains the definition for the
A* heading lines of the report. The format is written on the
A* first cycle, on a change of project number, or when overflow
A* occurs while printing details for a project code.
A*
```

Figure 196 (Part 1 of 4). Project Summary Report Data Descriptions - PRG07RP

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R TITLE1                      SKIPB(3)
A                      2'PRG07RP'
A                      34'TIME REPORTING PROJECT SUMMARY'
A                      88'PAGE'
A                      93PAGNBR
A                      SPACEA(1)
A                      35'FOR THE PERIOD ENDED'
A          CMTDT      R                    56EDTCDE(Y)
A                      89DATE EDTCDE(Y)
A                      SPACEA(2)
A                      4'PROJECT CODE'
A          PRCDEX     R                    18REFFLD(PRCDE)
A                      35'DESCRIPTION'
A          PRDSC      R                    48
A                      SPACEA(2)
A                      4'RESPONSIBILITY'
A                      41'START          ESTIMATED'
A                      69'COMPLETION      ESTIMATED'
A                      SPACEA(1)
A                      42'DATE           END DATE'
A                      72'DATE           TOTAL HOURS'
A                      SPACEA(1)
A N60          PRRSP   R                    4
A N60          PRSTR   R                   40EDTCDE(Y)
A N60          PREND   R                   55EDTCDE(Y)
A N60          PRCMP   R                   70EDTCDE(Y)
A N60          PREST   R                   86EDTCDE(L)
A                      SPACEA(2)
A                      4'EMPLOYEE        EMPLOYEE NAME'
A                      54'WEEK ENDING    HOURS'
A                      SPACEA(1)
A                      5'NUMBER'
A                      57'DATE           WORKED'
A                      SPACEA(1)

```

Figure 196 (Part 2 of 4). Project Summary Report Data Descriptions - PRG07RP

Monthly Processing

.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..

A*

A* The second format, DETAIL, contains the definition for the detail

A* print lines. The format is written for each detail record in

A* the monthly transaction file.

A*

AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*

A R DETAIL

A EMPNO R 5EDTCDE(Z)

A N61 ENAME R 18

A 61 18'INVALID EMPLOYEE NUMBER'

A CWKDTX R 55REFFLD(CWKDT)

A EDTCDE(Y)

A EHWRK R 71EDTCDE(L)

A SPACEA(1)

A*

A* The third format, TOTL1, contains the definition for total time

A* level break L1. The format is written on a change of week

A* ending date or a change of project code.

A*

A R TOTL1 SPACEB(1)

A 54'WEEKLY TOTAL'

A WKTOT 7S 1 69EDTCDE(L)

A SPACEA(2)

Figure 196 (Part 3 of 4). Project Summary Report Data Descriptions - PRG07RP

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
A*
A* The fourth format, TOTL2, contains the definition for total
A* time level break L2. The format is written on a change of
A* project code.
A*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R TOTL2                      SPACEB(1)
A          2'PROJECT SUMMARY:'
A          20'CURRENT          CURRENT'
A          49'PRIOR YEAR      TOTAL PROJECT'
A          82'% VARIANCE TO'
A          SPACEA(1)
A          21'MONTH          YEAR TO DATE'
A          52'TOTAL          HOURS'
A          81'ESTIMATED HOURS'
A          SPACEA(1)
A          PRMTH              7S 1      19EDTCDE(L)
A          PRYER              9S 1      32EDTCDE(L)
A          PRHRP              R          49EDTCDE(L)
A          PRTOT              9S 1      65EDTCDE(L)
A          PRVAR              5S 1      86EDTCDE(L)

```

Figure 196 (Part 4 of 4). Project Summary Report Data Descriptions - PRG07RP

Project Summary Report RPG/400 Program - PRG07

Figure 197 shows RPG/400 program PRG07 with embedded comments to explain the logic flow and use of various RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG07 - Time Reporting Project Summary Report
F* DESCRIPTION - This program produces the time reporting monthly
F*                project summary report. All time entries for the
F*                month are printed by employee number within week
F*                ending date within project code. Subtotals is
F*                printed by week and a project summary is printed
F*                on a change of project code.
F*****
F* This program uses externally described files. Files
F* used are: TRMNTHL - logical view of TRMNTH, monthly transaction
F*                file by project code, employee number and
F*                week ending date.
F*                EMPMST - employee master file
F*                PRJMST - project master file
F*                PRG07RP - project summary report file
F*****
F* INDICATORS USED:
F* 60 - Project master record not found
F* 61 - Employee master record not found
F* L1 - Control level on week ending date
F* L2 - Control level on project code
F*****
F* SUBROUTINES USED:
F* DTLSR - Detail calculations routine
F* LICHK - Line count check routine
F* L2CLR - Clear work fields at detail time L2
F* SUBRL1 - Total time calculations - change of week ending date
F* SUBRL2 - Total time calculations - change of project code
F*****
```

Figure 197 (Part 1 of 7). Sample RPG/400 Program - PRG07

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTRMNTN IP E K DISK
FEMPMST IF E K DISK
FPRJMST IF E K DISK
FPRG07RP 0 E PRINTER
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* The following array is used to store the weekly project code
E* hours. The array contains up to five weekly totals.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E ARR 5 5 1
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The following code renames the monthly transaction file input
I* field names. These fields appear in other data definitions
I* and are overlaid when those files are read, these code renames
I* prevent the overlay.
IRcdname+....In.....*
IRCMNTH
I.....Ext-field+.....Field+L1M1..PlMnZr...*
I PRCDE PRCDEXL2
I CWKDT CWKDTXL1
I CMTDT CMTDTX
I* Externally described control file data area
I*
IDcname....NODsExt-file++.....OccrLen+.....*
ICTLFIL EUDS
I*
```

Figure 197 (Part 2 of 7). Sample RPG/400 Program - PRG07

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* MAINLINE: The mainline routine consists of four EXSR subroutines.
C* The first subroutine is processed at detail time when the control
C* level indicator L2 is on. This occurs on the first RPG/400
C* program cycle and on the RPG/400 program cycle following total
C* time calculations. The L2CLR subroutine clears work fields and
C* writes report headings. The second subroutine, DTLSR, is
C* processed on each RPG/400 detail time cycle. The routine writes
C* detail report lines and accumulates data for total time printing.
C* The third and fourth subroutines are processed at total time.
C* The SUBRL1 subroutine is processed on a change of week ending
C* date and also on a change of project code (RPG logic sets on
C* all lower level control indicators when a control break occurs,
C* that is, when L2 is set on, so is L1). The SUBRL2 subroutine
C* is processed on a change of project code.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++
C L2 EXSR L2CLR
C EXSR DTLSR
CL1 EXSR SUBRL1
CL2 EXSR SUBRL2
C*
C*****
C* DTLSR SUBROUTINE: This routine performs detail time operations.
C* The hours worked EHWRK from the transaction record are added
C* to the current element of the project hours array. The employee
C* master file is accessed using the CHAIN operation and the
C* employee number from the transaction record. The detail record
C* is then written to the printer file PRG07RP by using the record
C* format DETAIL. The line counter is incremented by one and the
C* overflow routine LICHK is processed to determine if a skip to
C* new page and heading line output is required.
C*
```

Figure 197 (Part 3 of 7). Sample RPG/400 Program - PRG07


```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          DTLR      BEGSR
C          EHWRK     ADD  ARR,P   ARR,P
C          EMPNO     CHAINEMPST           61
C          WRITEDETAIL
C          ADD 1      LICNT
C          EXSR LICHK
C          ENDSR
C*
C*****
C* SUBRL1 SUBROUTINE: This routine performs total time operations.
C* The current week's total hours from the project hours array is
C* moved to the print field WKTOT using the Z-ADD operation and the
C* current occurrence of the project array (array index value in P).
C* The weekly total line is written to printer file PRG07RP using
C* print format TOTL1. The project hours array index P is
C* incremented by one for accumulating the next week's hours, and
C* the line counter is incremented by two. The LICHK overflow
C* routine is processed to determine if a skip to new page and
C* heading line output is required.
C*
C          SUBRL1     BEGSR
C          Z-ADDARR,P   WKTOT
C          WRITETOTL1
C          ADD 1      P
C          ADD 2      LICNT
C          EXSR LICHK
C          ENDSR

```

Figure 197 (Part 4 of 7). Sample RPG/400 Program - PRG07

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* SUBRL2 SUBROUTINE: This routine performs total time operations.
C* The line counter value is checked. If the value is greater than
C* or equal to 55, it is set to 60 and the LICHK overflow routine
C* is processed. The program performs these operations to ensure
C* that enough print lines are available on the page to print the
C* project total lines. The project array is summed using the
C* XFOOT operation. This operation adds all elements of the array
C* together and places the sum in the result field. The series of
C* arithmetic operations that follow prepare the project summary
C* lines for printing. The total project hours for the month are
C* added to the total year-to-date project hours from the project
C* file to determine current year-to-date hours. The current year-
C* to-date hours are added to the prior year total to determine the
C* total project hours. The total project hours is subtracted from
C* the estimated total hours, and the sign of the result is changed
C* using the Z-SUB operation to give the variance hours. The
C* variance hours are divided by the estimated total hours and then
C* multiplied by 100 to give the variance percent. This percent
C* indicates what percent the actual hours are greater than
C* (positive %) or less than (negative %) the estimated hours.
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C          SUBRL2    BEGSR
C          LICNT     IFGE 55
C                   Z-ADD60          LICNT
C                   EXSR LICHK
C                   END
C                   XFOOTARRP        PRMTH
C          PRMTH     ADD  EPHRY        PRYER
C          PRYER     ADD  PRHRP        PRTOT    91
C          PREST     SUB  PRTOT        PRDIF    91
C                   Z-SUBPRDIF        PRDIF
C          PRDIF     DIV  PRTOT        WRK1     53H
C          WRK1      MULT 100          PRVAR    51
```

Figure 197 (Part 5 of 7). Sample RPG/400 Program - PRG07

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The employee total line is now ready for printing. The TOTL2
C* format is written. Because the program will perform detail
C* time L2 operations on the next cycle to prepare for the next
C* project code, the line counter field is not incremented and
C* overflow is not checked after writing the format.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          WRITETOTL2
C          ENDSR
C*
C*****
C* LICHK SUBROUTINE: This routine controls page overflow. If the
C* line count is greater than or equal to 60, the heading format
C* TITLE1 is written and the line count is set to 9.
C*
C          LICHK      BEGSR
C          LICNT      IFGE 60
C                   WRITETITLE1
C                   Z-ADD9          LICNT
C                   END
C                   ENDSR

```

Figure 197 (Part 6 of 7). Sample RPG/400 Program - PRG07

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* L2CLR SUBROUTINE: This routine prepares work fields and prints
C* heading lines before processing the first project detail record.
C* The project hours array ARRP is initialized to 0 and the array
C* index set to 1. The project master file is accessed using the
C* project code from the transaction record. If the project record
C* is not found, indicator 60 is set on. The report headings are
C* printed by writing print format TITLE1 and the line counter is
C* set to 9.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          L2CLR      BEGSR
C          Z-ADD0      ARRP
C          Z-ADD1      P          10
C          PRCDEX     CHAINPRJMST          60
C          *IN60      IFEQ '1'
C          MOVE *BLANKS PRDSC
C          MOVE 'INVALID' PRDSC
C          END
C          WRITETITLE1
C          Z-ADD9      LICNT    30
C          ENDSR
```

Figure 197 (Part 7 of 7). Sample RPG/400 Program - PRG07

Reason Code Summary Report Data Descriptions - PRG08RP

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*****
A* This print file describes the format for the monthly time
A* reporting reason code summary report. In this printer file are
A* four record formats, identified by an R in position 17 followed
A* by the format name in positions 19 through 20. The following
A* keywords are used:
A* DATE - Specifies the system date
A* EDTCDE(a) - Edits output capable numeric fields
A* PAGNBR - Specifies a four digit, zoned decimal field to
A* contain the page number.
A* REF(REFMST) - Lines containing an R in position 29 uses the
A* attributes from a previously defined field in
A* this reference file.
A* REFFLD - References a field to a previously defined field.
A* SKIPB(n) - Specifies that the printer device is to skip to
A* a specific line before it prints the next line.
A* SPACEA(n) - Specifies that the printer device is to space n
A* lines after it prints one or more lines.
A* SPACEB(n) - Specifies that the printer device is to space n
A* lines before it prints the next line or lines.
A*****
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A REF(REFMST)
A*
A* The first format, TITLE1, contains the definition for the reports
A* heading lines. The format is written on the first cycle,
A* on a change of reason code or when overflow occurs while
A* printing details for a reason code.
A*
```

Figure 199 (Part 1 of 3). Reason Code Summary Report Data Descriptions - PRG08RP

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****
A          R TITLE1                               SKIPB(3)
A          2'PRG08RP'
A          29'TIME REPORTING REASON CODE SUMMARY'
A          83'PAGE'
A          88PAGNBR
A          SPACEA(1)
A          31'FOR THE PERIOD ENDED'
A          CMTDT      R          52EDTCDE(Y)
A          84DATE EDTCDE(Y)
A          SPACEA(2)
A          2'REASON CODE'
A          RSCDEX     R          15REFFLD(RSCDE)
A          29'DESCRIPTION'
A          RSDSC      R          42
A          SPACEA(2)
A          2'EMPLOYEE      EMPLOYEE NAME'
A          52'WEEK ENDING  HOURS'
A          SPACEA(1)
A          3'NUMBER'
A          55'DATE        WORKED'
A          SPACEA(1)

```

A*
A* The second format, DETAIL, contains the definition for the detail
A* print lines. The format is written for each detail record in
A* the monthly transaction file.
A*

Figure 199 (Part 2 of 3). Reason Code Summary Report Data Descriptions - PRG08RP

Monthly Processing

.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..
 AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++*****

```

A          R DETAIL
A          EMPNO      R          3EDTCDE(Z)
A N61      EMPNAM      30          16
A 61
A          CWKDTX      R          53REFFLD(CWKDT)
A
A          EHWK      R          69EDTCDE(L)
A          SPACEA(1)
  
```

A*
 A* The third format, TOTL1, contains the definition for total time
 A* level break L1. The format is written on a change of week
 A* ending date or a change of reason code.

```

A          R TOTL1          SPACEB(1)
A          52'WEEKLY TOTAL'
A          WKTOT          7S 1  67EDTCDE(L)
A          SPACEA(2)
  
```

A*
 A* The fourth format, TOTL2, contains the definition for total
 A* time level break L2. The format is written on a change of
 A* reason code.

```

A          R TOTL2          SPACEB(1)
A          2'REASON CODE SUMMARY:'
A          29'CURRENT          CURRENT'
A          60'PRIOR YEAR'
A          SPACEA(1)
A          30'MONTH          YEAR TO DATE'
A          63'TOTAL'
A          SPACEA(1)
A          RSMTH          7S 1  28EDTCDE(L)
A          RSYER          9S 1  41EDTCDE(L)
A          RSHRP      R          60EDTCDE(L)
  
```

Figure 199 (Part 3 of 3). Reason Code Summary Report Data Descriptions - PRG08RP

Reason Code Summary Report RPG/400 Program - PRG08

Figure 200 shows the RPG/400 program PRG08 with embedded comments to explain the logic flow and use of various RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG08 - Time Reporting Reason Code Summary Report
F* DESCRIPTION - This program produces the time reporting monthly
F*                reason code summary report. All time entries for
F*                the month are printed by employee number within
F*                week ending date within reason code. Subtotals
F*                are printed by week and a reason code summary is
F*                printed on a change of reason code.
F*****
F* This program uses externally described files. Files
F* used are: TRMNTHN - logical view of TRMNTH, monthly transaction
F*                file by reason code, employee number, and
F*                week ending date
F*                EMPMST - employee master file
F*                RSNMST - reason code master file
F*                PRG08RP - reason code summary report file
F*****
F* INDICATORS USED:
F* 60 - Reason code master record not found
F* 61 - Employee master record not found
F* 90 - String found in SCAN operation
F* L1 - Control level on week ending date
F* L2 - Control level on reason code
F*****
F* SUBROUTINES USED:
F* DTLSR - Detail calculations routine
F* *INZSR - Initialization subroutine
F* LICHK - Line count check routine
F* L2CLR - Clear work fields at detail time L2
F* SUBRL1 - Total time calculations - change of week ending date
F* SUBRL2 - Total time calculations - change of reason code
F*****
```

Figure 200 (Part 1 of 8). Sample RPG/400 Program - PRG08

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FTRMTHR IP E K DISK
FMPMST IF E K DISK
FRSNMST IF E K DISK
FPRG08RP 0 E PRINTER
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* The following array is used to store the weekly reason code
E* hours. The array contains up to five weekly totals.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E ARR 5 5 1
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* The following code renames the monthly transaction file input
I* field names. These fields appear in other data definitions
I* and are overlaid when those files are read, these code renames
I* prevent the overlay.
IRcdname+....In.....*
IRCMNTH
I.....Ext-field+.....Field+L1M1..P1MnZr...*
I RSCDE RSCDEXL2
I CWKDT CWKDTXL1
I CMTDT CMTDTX
I* Externally described control file data area
I*
IDcname....NODsExt-file++.....OccrLen+.....*
ICTLFIL EUDS
I*
```

Figure 200 (Part 2 of 8). Sample RPG/400 Program - PRG08

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* MAINLINE: The mainline routine consists of four EXSR subroutines.
C* The *INZSR initialization subroutine is processed first during the
C* initialization step of the program cycle. The *INZSR subroutine
C* initializes fields used in calculations.
C* The L2CLR subroutine is processed at detail time when the control
C* level indicator L2 is on. This occurs on the first RPG/400
C* program cycle and on the RPG/400 program cycle following total
C* time calculations. The L2CLR subroutine clears work fields and
C* writes report headings. The second subroutine, DTLSR, is
C* processed on each RPG/400 detail time cycle. This routine writes
C* detail report lines and accumulates data for total time printing.
C* The third and fourth subroutines are processed at total time.
C* The SUBRL1 subroutine is processed on a change of week ending
C* date and also on a change of reason code (RPG/400 logic sets on
C* all lower level control indicators when a control break occurs,
C* that is, when L2 is set on, so is L1). The SUBRL2 subroutine is
C* processed on a change of reason code.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C L2 EXSR L2CLR
C EXSR DTLSR
CL1 EXSR SUBRL1
CL2 EXSR SUBRL2
C*

```

Figure 200 (Part 3 of 8). Sample RPG/400 Program - PRG08

Monthly Processing

```
C*****
C*  DTLSR SUBROUTINE: This routine performs detail time operations.
C*  The hours worked EHWRK from the transaction record are added
C*  to the current element of the reason code hours array. The
C*  employee master file is accessed using the CHAIN operation and
C*  the employee number from the transaction record. If an EMPMST
C*  record is found, indicator 61 is off and the SCAN, SUBST and CAT
C*  operations format the employee name for the report. The employee
C*  name from the input record is in the format of first name,
C*  followed by a blank, followed by last name. The employee name is
C*  printed on the report in the reverse format: last name, followed by
C*  a blank, followed by the first name. The SCAN operation determines
C*  the position of the blank in ENAME. If the SCAN is successful: the
C*  lengths of the first and last name are calculated; the SUBST
C*  operations extract the first and last name from the ENAME field;
C*  the CAT operation concatenates the names with one blank between
C*  them. If the SCAN is not successful, the employee name is printed
C*  on the report as it appears on the input record. The detail
C*  record is then written to the printer file PRG08RP using the
C*  record format DETAIL. The line counter is incremented by one
C*  and the overflow routine LICHK is processed to determine if a
C*  skip to new page and heading line output is required.
C*
```

Figure 200 (Part 4 of 8). Sample RPG/400 Program - PRG08

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          DTLR      BEGSR
C          EHWRK     ADD  ARR,N      ARR,N
C          EMPNO     CHAINEMPST          61
C          *IN61     IFEQ '0'
C          BLK1      SCAN ENAME      BLKPOS  20      90
C          *IN90     IFEQ '1'
C          1         ADD  BLKPOS      SPOS   20
C          LENENM    SUB  BLKPOS      LENLNM  20
C          LENLNM    SUBSTENAME:SOSLNAME 30
C          BLKPOS    SUB  1           EPOS   20
C          EPOS      SUBSTENAME      FNAME  30
C          LNAME     CAT  FNAME:1     EMPNAM
C                  CLEARFNAME
C                  CLEARLNAME
C                  ELSE
C                  MOVE ENAME        EMPNAM
C                  END
C                  END
C                  WRITEDetail
C                  ADD  1            LICNT
C                  EXSR LICHK
C                  ENDSR
C*****

```

Figure 200 (Part 5 of 8). Sample RPG/400 Program - PRG08

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* *INZSR SUBROUTINE: This routine initializes fields using the
C* MOVE and Z-ADD operations. The BLK1 and LENENM fields are used
C* to format the employee name in the DTLSR subroutine. The N
C* field is the index for array ARRN.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          *INZSR      BEGSR
C          MOVE *BLANK   BLK1    1
C          Z-ADD30      LENENM  20
C          Z-ADD1       N        10
C          ENDSR
C*
C*****
C* SUBRL1 SUBROUTINE: This routine performs total time operations.
C* The current weeks total hours from the reason code hours array
C* is moved to the print field WKTOT using the Z-ADD operation and
C* the current occurrence of the reason code array (array index
C* value in N). The weekly total line is written to printer file
C* PRG08RP using print format TOTL1. The reason code hours array
C* index N is incremented by one for accumulating the next week's
C* hours and the line counter is incremented by two. The LICHK
C* overflow routine is processed to determine if a skip to new page
C* and heading line output is required.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          SUBRL1      BEGSR
C          Z-ADDARRN,N  WKTOT
C          WRITETOTL1
C          ADD 1        N
C          ADD 2        LICNT
C          EXSR LICHK
C          ENDSR
```

Figure 200 (Part 6 of 8). Sample RPG/400 Program - PRG08

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* SUBRL2 SUBROUTINE: This routine performs total time operations.
C* The line counter value is checked. If it is greater than or
C* equal to 55, then it is set to 60 and the LICHK overflow
C* subroutine is processed. The program performs these operations
C* to ensure that enough print lines are available on the page to
C* print the reason code total lines. The reason code array is
C* summed using the XFOOT operation. This operation adds all
C* elements of the array together and places the sum in the result
C* field. The current month total hours are added to the year-to-
C* date hours from the reason code file to determine the current
C* year-to-date hours.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          SUBRL2    BEGSR
C          LICNT     IFGE 55
C                   Z-ADD60          LICNT
C                   EXSR LICHK
C                   END
C                   XFOOTARRN        RSMTH
C          RSMTH     ADD  RSHRY      RSYER
C*
C* The reason code total line is now ready for printing. The TOTL2
C* format is written. Because the program will perform detail time
C* L2 operations on the next cycle to prepare for the next reason
C* code, the line counter field is not incremented and overflow is
C* not checked after writing the format.
C*
C                   WRITETOTL2
C                   ENDSR

```

Figure 200 (Part 7 of 8). Sample RPG/400 Program - PRG08

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* LICHK SUBROUTINE: This routine controls page overflow.
C* If the line count is greater than or equal to 60, the heading
C* format TITLE1 is written and the line count is set to 9.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          LICHK          BEGSR
C          LICNT          IFGE 60
C                      WRITETITLE1
C                      Z-ADD9          LICNT
C                      END
C                      ENDSR
C*
C*****
C* L2CLR SUBROUTINE: This routine prepares work fields and prints
C* heading lines before processing the first reason code detail
C* record. The reason code hours array ARRN is set to 0 using the
C* CLEAR operation and the array index is set to 1 using the RESET
C* operation. The array index, N, is initialized to 1 in the
C* initialization subroutine and is reset to that value. The
C* reason code master file is accessed using the reason code from
C* the transaction record. If the reason code record is not found,
C* indicator 60 is set on. The report headings are printed by
C* writing print format TITLE1 and the line counter is set to 9.
C*
C          L2CLR          BEGSR
C                      CLEARARRN
C                      RESETN
C          RSCDEX          CHAINRSNMST          60
C                      WRITETITLE1
C                      Z-ADD9          LICNT    30
C                      ENDSR
```

Figure 200 (Part 8 of 8). Sample RPG/400 Program - PRG08

Master File Monthly Update and Clear RPG/400 Program - PRG04

Figure 201 shows the RPG/400 program PRG04 with embedded comments to explain the logic flow and use of various RPG/400 functions and operation codes.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*****
F* PRG04 - Time Reporting Master File Update
F* DESCRIPTION - This program performs monthly and year end roll
F*                of time reporting hours. The files and type of
F*                update are controlled by external indicators.
F*****
F* This program uses externally described files. Files
F* used are: EMPMST - Employee master file
F*           PRJMST - Project master file
F*           RSNMST - Reason code master file
F*****
F* INDICATORS USED:
F* 50 - End of file
F* U1 - Employee master update
F* U2 - Project master update
F* U3 - Reason code master update
F* U4 - Year end processing
F*****
F* SUBROUTINES USED:
F* EMPSR - Update employee master
F* PRJSR - Update project master
F* RSNSR - Update reason code master
F*****
FfilenameIPEAF....RlenLK1AI0vKlocEDevice+.....KExit++Entry+A....U1.*
FEMPMST  UF  E           K           DISK           U1
FPRJMST  UF  E           K           DISK           U2
FRSNMST  UF  E           K           DISK           U3
```

Figure 201 (Part 1 of 5). Sample RPG/400 Program - PRG04

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* MAINLINE: The mainline routine determines which file is to be
C* processed and runs the appropriate subroutine. The master
C* files are all described as full procedural (F in position 16
C* of the file description specification), which allows the program
C* to perform the read/write operations on the files. All files
C* are controlled by a file condition indicator (position 71 to 72
C* of the file description specification) that controls which file
C* is processed.
C*****
C* The following code determines which subroutine is evoked.
C* If *INU1 is on (equal to 1), EMPSR is selected; if *INU2 is on,
C* PRJSR is selected; if *INU3 is on, RSNSR is selected.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C      *INU1      CASEQ'1'      EMPSR
C      *INU2      CASEQ'1'      PRJSR
C      *INU3      CASEQ'1'      RSNSR
C                      END
C*****
C* EMPSR SUBROUTINE: The following line of code performs a Do Until
C* condition. The program loops between the DOUEQ statement and
C* the END statement until end of file (*IN50 equals 1) has been
C* reached.
C*
C      EMPSR      BEGSR
C*
C      *IN50      DOUEQ'1'
C                      READ RCEMP                      50
```

Figure 201 (Part 2 of 5). Sample RPG/400 Program - PRG04

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C* The following lines of code add current month hours to the
C* year-to-date hours for the employee master file. Since factor
C* 1 is not specified in the statements, factor 2 is added to
C* the result fields and the result place in the result field.
C* If *INU4 is on, this session is run for year end, and the
C* current year hours are moved to the prior year hours.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++*
C          ADD  EPHRC      EPHRY
C          ADD  EPNRC      EPNRY
C  U4      MOVE EPHRY      EPHRP
C  U4      MOVE EPNRY      EPNRP
C* The following code clears the current month hours fields
C* by zeroing them and adding 0 to them. If *INU4 is on, this
C* session is being run for year end, and the current year
C* hours must be zeroed as well.
C          Z-ADD0      EPHRC
C          Z-ADD0      EPNRC
C  U4      Z-ADD0      EPHRY
C  U4      Z-ADD0      EPNRY
C* The following code updates the employee master file using
C* the RCEMP format.
C          UPDATRCEMP
C          END
C* The preceding END statement is associated with the DOUEQ
C* statement.
C*
C* Last record indicator *INLR is set on (equal to 1) and
C* the program ends.
C*
C          MOVE '1'      *INLR
C          ENDSR

```

Figure 201 (Part 3 of 5). Sample RPG/400 Program - PRG04

Monthly Processing

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*****
C* PRJSR SUBROUTINE: This subroutine performs the same functions
C* as the EMPSR subroutine only the project master is updated.
C* Refer to EMPSR for specific information.
C*
CLON01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C          PRJSR      BEGSR
C          *IN50      DOUEQ'1'
C                      READ RCPRJ                      50
C*
C* Add current month to year-to-date, and move current year to
C* to prior year if U4 is on.
C          ADD PRHRC      PRHRY
C U4          MOVE PRHRY      PRHRP
C*
C* Zero current month, and year-to-date if U4 is on.
C          Z-ADD0      PRHRC
C U4          Z-ADD0      PRHRY
C*
C* Update project master file.
C          UPDATRCPRJ
C          END
C*
C* Set on last record indicator.
C          MOVE '1'      *INLR
C          ENDSR
C*****
C* RSNSR SUBROUTINE: This subroutine performs the same functions
C* as the EMPSR subroutine only the reason code master is updated.
C* Refer to EMPSR for specific information.
C*
C          RSNSR      BEGSR
C          *IN50      DOUEQ'1'
C                      READ RCRSN                      50
```

Figure 201 (Part 4 of 5). Sample RPG/400 Program - PRG04

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* Add current month to year-to-date, and move current year to
C* to prior year if U4 is on.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C          ADD  RSHRC      RSHRY
C  U4      MOVE RSHRY      RSHRP
C*
C* Zero current month, and year-to-date if U4 is on.
C          Z-ADD0      RSHRC
C  U4      Z-ADD0      RSHRY
C*
C* Update reason code master file
C          UPDATRCRSN
C          END
C*
C* Set on last record indicator
C          MOVE '1'      *INLR
C          ENDSR
C*

```

Figure 201 (Part 5 of 5). Sample RPG/400 Program - PRG04

Year End Processing

All the master files are processed as part of the last monthly update for the year to prepare the files for the new year. Each master file contains both current year-to-date and prior year total hours. Program PRG04 performs both the monthly and year end roll of the time reporting hours. External switches are used to control which file is processed, and if the session is for a regular month end or for a combined month end and year end. Refer to the detailed discussion of program PRG04 for details.

Year End Processing

Appendix A. RPG Compiler and Auto Report Program Service Information

This appendix is provided for the RPG/400 compiler service personnel to use when investigating RPG/400 compiler problems and provides the following information:

- Compiler overview
- Compiler debugging options
- Intermediate representation of program (IRP) layout
- Automatic report program overview.

RPG/400 compiler programmers can also use this information to investigate RPG/400 compiler problems on their own before or instead of calling for service.

Compiler Overview

This section provides the following compiler information:

- How the compiler works
- Compiler phase descriptions
- Major compiler data area descriptions
- Compiler error message organization.

Figure 202 on page 494 summarizes how an RPG/400 source program is compiled into a (encapsulated) program object.

Intermediate text, which is output from step 1 in Figure 202 on page 494, is a representation of RPG/400 source statements that is created by compiler phases and exists only while they are running. This text can be dynamically listed with the ITDUMP parameter of the CL command CRTRPGPGM (Create RPG/400 Program) or can be listed at the completion of any compiler phase with the SNPDUMP parameter of the CL command CRTRPGPGM. Refer to “Compiler Debugging Options” on page 498 for explanations of these parameters and examples of intermediate text.

When compilation ends, intermediate text has been processed and converted to appropriate IRP (intermediate representation of a program). IRP, which is output from step 2 in Figure 202 on page 494, can be dynamically listed with the CODELIST parameter of the CL command CRTRPGPGM or can be listed at the end of compilation with an *LIST value for the GENOPT parameter on the CL command CRTRPGPGM. Refer to “Compiler Debugging Options” on page 498 for explanations of these parameters and examples of IRP statements.

A program template is output from step 3 in Figure 202 on page 494. A template is the final form of a program before it is converted to an operable program, which is called an encapsulated program. A template can be listed at the end of a compilation with an *DUMP value for the GENOPT parameter on the CL command CRTRPGPGM. Refer to “Compiler Debugging Options” on page 498 for explanation of this parameter and an example of a program template listing.

Compiler Phases

The compiler consists of the phases listed in Table 20 on page 495. These phases are shown in the order in which they are run.

If *NOGEN has been specified for the OPTION parameter on the CL command CRTRPGPGM, compilation ends following phase QRGCR.

During compilation, those phases that have a U in the third column in Table 20 on page 495 run unconditionally and those phases that have a C in the third column run only if they are required for the program being compiled.

The first compiler phase is named QRG1. All phases that follow QRG1 have names that begin with QRG and end with two identifying characters. These phases can be referred to by their identifying characters. For example, these characters can be used as values for debugging parameters in the CL command CRTRPGPGM. Refer to "Compiler Debugging Options" on page 498 for more information.

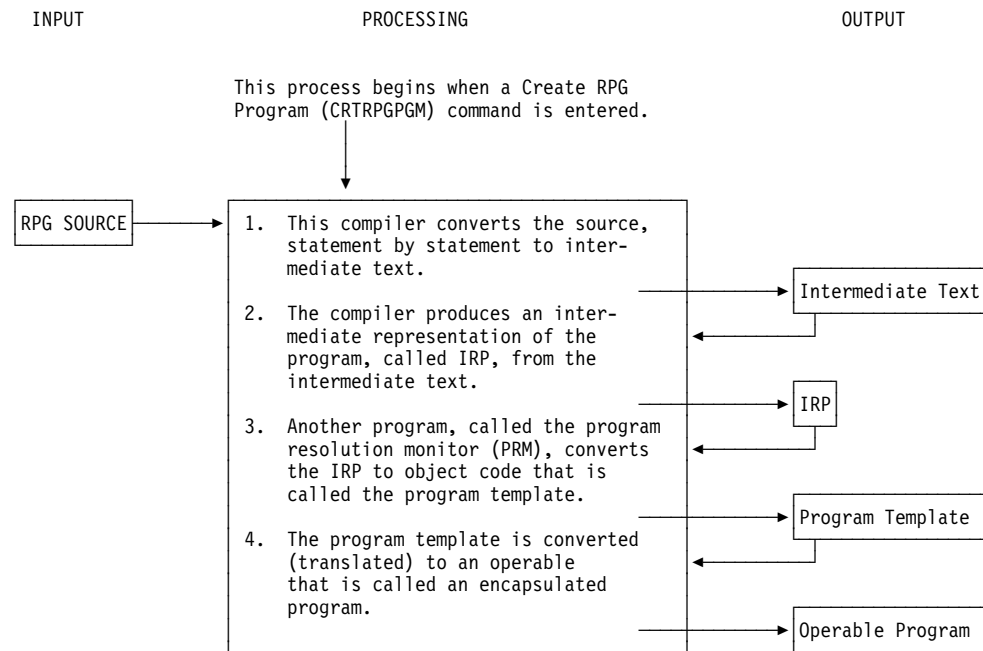


Figure 202. Overview of the RPG/400 Compiler

Table 20 (Page 1 of 2). Compiler Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRG1	Command interface that receives control when the CRTRPGPGM command is entered, assigns defaults to the command parameter list, and passes the command parameter list to QRGRT.	U
QRGRT	Root phase that controls the calling of all other compiler phases and contains all system interfaces such as reading and printing records.	U
QRGSF	Phase that diagnoses the file description specifications and builds a file table.	C
QRGSE	Phase that diagnoses the extension specifications and builds intermediate text.	C
QRGSI	Phase that diagnoses the input specifications and builds intermediate text.	C
QRGSC	Phase that diagnoses the calculation specifications and builds intermediate text.	C
QRGSO	Phase that diagnoses the output specifications and builds intermediate text.	C
QRGAE	Phase that generates declare statements for fields and creates edit masks.	U
QRGD1	Phase that diagnoses relational errors among the source specifications. These errors are illogical or incorrect combinations of entries.	U
QRGCR	Phase that produces a cross-reference listing, generates code for processing compile-time tables, and produces a list of compile-time messages on completion of a compilation.	U
QRGGB	Phase that generates user file control blocks (UFCBs).	U
QRGFB	Phase that is called by QRGGB to generate file information blocks (FIBs).	U
QRGPL	Phase that is called by QRGGB to generate PLISTs.	U
QRGGV	Phase that is called by QRGGB to generate file I/O drivers.	C
QRGGC	Mainline phase that controls processing of calculation operations.	C
QRGAC	Phase that is called by QRGGC to process arithmetic calculation operations.	C
QRGBC	Phase that is called by QRGGC to process branch calculation operations.	C
QRGCC	Phase that is called by QRGGC to process compare calculation operations.	C
QRGIC	Phase that is called by QRGGC to generate input/output linkages for calculation operations.	C
QRGMC	Phase that is called by QRGGC to generate code for MOVE calculation operations.	C
QRGRC	Phase that is called by QRGGC to process CLEAR and RESET calculation operations.	C
QRGTC	Phase that is called by QRGGC to process string calculation operations.	C
QRGYC	Phase that is called by QRGGC to process miscellaneous calculation operations.	C
QRGGI	Phase that generates code that extracts data from records and fills input fields.	C
QRGGO	Phase that generates code that builds output records.	C
QRGGS	Phase that generates data management for DISK and SEQ files.	C

Compiler Overview

Table 20 (Page 2 of 2). Compiler Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRGGW	Phase that is called by QRGGS to generate data management for WORKSTN files.	C
QRGGR	Phase that is called by QRGGS to generate data management for RAF files.	C
QRGC1	Phase that generates code for getting input and processing multiple files.	U
QRGEC	Phase that generates subroutines required for the program, generates code for processing run-time tables, and generates beginning and ending code for the program.	U

Major Compiler Data Areas

The major compiler data areas are a common area (VCOMMON), a field-name table (XFDTAB), a file-name table (XFLTAB), a record-name table (XRCTAB), and an indicator table (XINTAB).

Compiler Error Message Organization

Compiler error messages are organized according to the phases that issue them. For example, any compiler message beginning with 2 is issued by phase QRGSF. The following table lists compiler phases and the messages that they issue:

Table 21. Automatic Report Program Phases

Error Messages	Phase
0000 to 1999	QRGRT
2000 to 2999	QRGSF
3000 to 3999	QRGSE
4000 to 4999	QRGSI
5000 to 5999	QRGSC
6000 to 6999	QRGSO
7000 to 7999	QRGDI and QRGCR
8000 to 8999	QRGAE and QRGCR

Run-Time Subroutines

Table 22 lists the run-time subroutines that are used by the compiler.

<i>Table 22. Run-Time Subroutines</i>		
Subroutine Name	Subroutine Description	Called: Unconditionally (U) Conditionally (C)
QRGXINIT	Initializes the RPG/400 program.	U
QRGXDUMP	Provides a formatted dump of the RPG/400 program.	C
QRGXERR	Called when an error message is received.	C
QRGXPRT	Called by QRGXDUMP to print the dump.	C
QRGXSTAT	Called for the POST operation with a device specified in factor 1.	C
QRGXTIME	Called for the TIME operation code.	C
QRGXIOU	Called for the IN/OUT/UNLCK operation codes. Also used by the compiler to set the RETURNCODE data area and to retrieve the RPG/400 control-specification data area during compilation.	U
QRGXMSG	Sends RPG/400 run-time messages to the requester and provides the system dump when requested.	C
QRGXCLRF	Clears the file before a table dump at program end.	C
QRGXDSP	Called for the DSPLY operation code.	C
QRGXSIGE	Signals exception for run-time terminal error.	C
QRGXGDDM	Called for the CALL GDDM operation.	C
QRGINVX	Unlocks data areas when a program ends because of errors.	C
SUBR23R3	Message retrieving	C
SUBR40R3	Moving double-byte data and deleting control characters.	C
SUBR41R3	Moving double-byte data and adding control characters.	C

Compiler Debugging Options

This section explains each of the debugging parameters. For examples of debugging information that can be requested by these parameters, refer to “Examples of Using Compiler Debugging Options” on page 500.

***SOURCE Value for the OPTION Parameter**

A value of *SOURCE for the OPTION parameter requests a listing of the RPG/400 source program. The default is *SOURCE.

***XREF Value for the OPTION Parameter**

A value of *XREF for the OPTION parameter requests a cross-reference listing and a key field information table (when appropriate). Refer to Chapter 3, “Compiling an RPG/400 Program” for a description of this listing. The default is *XREF.

***DUMP Value for the OPTION Parameter**

A value of *DUMP for the OPTION parameter causes the contents of major data areas such as VCOMMON, file-name table, field-name table, and IT (intermediate text) to be printed. This printing occurs only if compilation ends abnormally. Therefore, *DUMP is usually specified when an unsuccessful compilation is retried. The default is *NODUMP.

***LIST Value for the GENOPT Parameter**

A value of *LIST for the GENOPT parameter causes IRP, its associated hexadecimal code, and any error messages to be listed. The default is *NOLIST.

***ATR Value for the GENOPT Parameter**

A value of *ATR for the GENOPT parameter causes the attributes for the IRP source to be listed. The listing includes the field descriptions and the statement numbers on which the fields are defined. The default is *NOATR.

***XREF Value for the GENOPT Parameter**

A value of *XREF for the GENOPT parameter causes a cross-reference listing of all objects defined in the IRP to be printed when compilation ends.

***DUMP Value for the GENOPT Parameter**

A value of *DUMP for the GENOPT parameter causes the program template to be listed. The default is *NODUMP.

***PATCH Value for the GENOPT Parameter**

A value of *PATCH for the GENOPT parameter reserves space in the compiled program for a program patch area. The program patch area can be used for your debugging purposes. The size of the patch area is based on the size of the generated program. The default is *NOPATCH.

*OPTIMIZE Value for the GENOPT Parameter

A value of *OPTIMIZE for the GENOPT parameter causes the compiler to generate a program that runs more efficiently and requires less storage. However, specifying *OPTIMIZE can substantially increase the time required to create a program. Existing programs can be optimized with the CL command CHGPGM.

ITDUMP Parameter

For the CRTRPGPGM command, the ITDUMP parameter causes dynamic listing of intermediate text produced by a specified phase. Dynamic listing means that the intermediate text is printed during compilation while the intermediate text is being built and stored. For the CRTRPTPGM command, the ITDUMP parameter causes a flow of the major routines run in one or more specified phases to be printed.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the ITDUMP parameter. The list must be enclosed in parentheses. For example, the following ITDUMP parameter causes dynamic listing of intermediate text produced by QRGSE, QRGSO, and QRGSC: ITDUMP(SESOSC).

SNPDUMP Parameter

The SNPDUMP parameter produces a listing of major data areas and intermediate text following the running of one or more specified phases.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the SNPDUMP parameter. The list must be enclosed in parentheses. For example, the following SNPDUMP parameter causes the listing of intermediate text produced by QRGSI, QRGSC, and QRGSO and also causes the contents of major data areas to be listed: SNPDUMP(SISCSO).

CODELIST Parameter

The CODELIST parameter causes dynamic listing of IRP produced by a specified phase. Dynamic listing means that the IRP is printed during compilation while the specified phase processes.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the CODELIST parameter. The list must be enclosed in parentheses. For example, the following CODELIST parameter causes dynamic listing of IRP produced by QRGGC, QRGGO, and QRGEC: CODELIST(GCGOEC).

PHSTRC Parameter

The PHSTRC parameter specifies whether or not a phase trace occurs during compilation. A phase trace consists of the names of compiler phases being printed on the compiler listing in the order that the phases process. The numbers of the RXT messages (such as compiler headings) are also listed as they are retrieved.

The values that can be coded for the PHSTRC parameter are *YES and *NO. *NO is the default value.

Examples of Using Compiler Debugging Options

Figure 203 on page 501 shows examples of debugging information that can be requested by compiler debugging options on the CRTRPGPGM command. The compiler listing in Figure 203 on page 501 was printed for a CRTRPGPGM command that specified the following debugging parameters:

```
GENOPT(*LIST *DUMP) ITDUMP(SC) SNPDUMP(GO)
CODELIST(GO) PHSTRC(*YES)
```

The PHSTRC(*YES) parameter causes the name of a phase to be printed when the phase processes. For example, **A** in Figure 203 shows that phase QRGSF processed the file description specification, phase QRGSE processed the extension specification, phase QRGSI processed the input specifications, and phase QRGSC processed the calculation specifications.

The ITDUMP(SC) parameter causes printing of intermediate text that phase QRGSC builds and stores. (See **B** in Figure 203.)

The CODELIST(GO) parameter causes printing of IRP produced by phase QRGGO when that phase ends. (See **C** in Figure 203.)

The SNPDUMP(GO) parameter causes printing of the contents of major data areas when phase QRGGO ends and causes printing of intermediate text produced by QRGGO. (See **D** in Figure 203.)

The *LIST value for the GENOPT parameter causes printing of IRP and machine instructions when compilation ends. (See **E** in Figure 203.) The headings in this IRP listing indicate the following information:

SEQ: A sequential numbering of the IRP statements. Error messages such as IRP syntax errors issued by the program resolution monitor use this number to refer to the IRP statements in error.

INST: A sequential numbering of the machine instructions generated from the IRP statements. Not all IRP statements cause machine instructions to be generated. The instruction number can be used as a breakpoint for OS/400 debugging functions. Refer to Chapter 4, "Error Messages, Testing, and Debugging" or the *CL Programmer's Guide* for further information about breakpoints.

GENERATED CODE: Machine instructions that have been generated from IRP statements.

GENERATED OUTPUT: IRP statements.

BREAK: Breakpoints in the IRP that can be used for stopping points in OS/400 debugging functions. Refer to Chapter 4, "Error Messages, Testing, and Debugging" or the *CL Programmer's Guide* for further information about breakpoints. If the breakpoint is a number, it indicates an RPG/400 source statement from which the IRP statement was generated.

The *DUMP value for the GENOPT parameter causes printing of the program template when compilation ends. (See **F** in Figure 203.)

Examples of Using Compiler Debugging Options

A value of *DUMP can be coded for the OPTION parameter to cause the contents of major compiler data areas to be printed if the compiler ends abnormally. Figure 204 on page 515 shows an example of the information printed. For this example, the command is:

```
CRTRPGGM OPTION(*DUMP)
```

```
RXT0001
RXT0002
RXT0003
RXT0004
RXT0028
5763RG1 V3R0M5 940125          IBM SAA RPG/400          QGPL/DATAE          01/25/94 14:40:34          Page    1

RXT0005
  Compiler . . . . . : IBM SAA RPG/400
RXT0020
  Command Options:
RXT0023
RXT0070
  Program . . . . . : QGPL/DATAE
RXT0071
  Source file . . . . . : *LIBL/QRPGSRC
RXT0072
  Source member . . . . . : *PGM
RXT0073
  Source listing options . . . . . : *SOURCE *XREF *GEN *NODUMP *NOSECLVL
RXT0024
RXT0074
  Generation options . . . . . : *LIST *NOXREF *NOATR *DUMP *NOOPTIMIZE
RXT0093
  Source listing indentation . . . . . : *NONE
RXT0029
RXT0096
  Type conversion options . . . . . : *NONE
RXT0026
RXT0086
  SAA flagging . . . . . : *NOFLAG
RXT0075
  Generation severity level . . . . . : 9
RXT0076
  Print file . . . . . : *LIBL/QSYSPRT
RXT0077
  Replace program . . . . . : *YES
RXT0027
RXT0087
  Target release . . . . . : *CURRENT
RXT0025
RXT0078
  User profile . . . . . : *USER
RXT0079
  Authority . . . . . : *CHANGE
RXT0080
  Text . . . . . : *SRCMBRTXT
RXT0081
  Phase trace . . . . . : *YES
RXT0082
  Intermediate text dump . . . . . : SC
RXT0083
  Snap dump . . . . . : GO
RXT0084
  Codelist . . . . . : GO
RXT0085
  Ignore decimal data error . . . . . : *NO
RXT0097
  Allow null values . . . . . : *NO
RXT0010
  Actual Program Source:
RXT0011
  Member . . . . . : DATAE
RXT0012
  File . . . . . : QRPGSRC
RXT0013
  Library . . . . . : QGPL
RXT0014
  Last Change . . . . . : 01/25/94 14:40:26
OPTIONS PASSED IN BY CL
```

Figure 203 (Part 1 of 14). Examples of Compiler Debugging Information

Examples of Using Compiler Debugging Options

```

RXT0055
      A d d i t i o n a l   D i a g n o s t i c   M e s s a g e s
                                                    PHASE - D1           00000
RXT7999
RXT6999
                                                    PHASE - AE           00000
                                                    PHASE - CR           00000
5763RG1 V3R0M5 940125      IBM SAA RPG/400      QGPL/DATAE      01/25/94 14:40:34      Page      5

RXT9111
TABLE OF END POSITION OFFSETS FOR FIELDS DESCRIBED USING POSITION NOTATION.
RXT9112
      1200      4
RXT9100
5763RG1 V3R0M5 940125      IBM SAA RPG/400      QGPL/DATAE      01/25/94 14:40:34      Page      6

RXT9104
      C r o s s   R e f e r e n c e

RXT9126
File and Record References:
RXT9108
      FILE/RCD   DEV/RCD   REFERENCES (D=DEFINED)
      01 QSYSVRT   PRINTER   200D 300 1100 1201
RXT9127
Field References:
RXT9107
      FIELD     ATTR     REFERENCES (M=MODIFIED D=DEFINED)
* 7031 EXDS      DS(3)    400D
      F1        Z(3,0)   500D 1000M
      PAGE     P(4,0)   1200
      1        LITERAL 1000
      100     LITERAL  700

RXT9128
Indicator References:
RXT9109
      INDICATOR REFERENCES (M=MODIFIED D=DEFINED)
      LR        600M
      OA        200D 1201
RXT9122
* * * * *   E N D   O F   C R O S S   R E F E R E N C E   * * * * *

RXT9132
5763RG1 V3R0M5 940125      IBM SAA RPG/400      QGPL/DATAE      01/25/94 14:40:34      Page      7

RXT9123
      M e s s a g e   S u m m a r y

RXT9131
* QRG6103 Severity: 00   Number: 1
      Message . . . . : No Overflow Indicator is specified but an
indicator is assigned to a file and automatic skip to 6 is
generated.

RXT9131
* QRG7031 Severity: 00   Number: 1
      Message . . . . : The Name or indicator is not referenced.

RXT9124
* * * * *   E N D   O F   M E S S A G E   S U M M A R Y   * * * * *

```

Figure 203 (Part 3 of 14). Examples of Compiler Debugging Information

Examples of Using Compiler Debugging Options

	PHASE - C1	00003
	PHASE - GC	00000
	PHASE - G0	00006

C		
;/* PHASE - QRGGO	DATE - 09/25/92 *//*SVP*/	GO
/*****	GEN TIME - 0 *****//*ZCOMENT*/	GO
;ENTRY .OFL	INT/*ZLABX*/	GO
;DCL INSPTR .OFLWRN/*OVERFLOW RETURN POINTER*//*ZDCLRTN*/		GO
;BRK '1201	'/*BRK POINT*//*ZBRKPT*/	GO
;CMPBLA(B) *IN1A,*OFF/ EQ(.00R0001)/*COND IND TST*//*ZCONID*/		GO
;SPACE 3		GO
;/* QSYSPT	FILE OUTPUT *//*ZFLCOM*/	GO
;SETSPP .FIBPTR,.F01FIB/*SET FIB PTR*//*ZFIBPT*/		GO
;CPYBLA .CURROP, C'WRITE'/* SET OP *//*ZCURROP*/		GO
;CPYBLA .PRTCTL,X'0000000000060000'/*SET OPERATION*//*ZPRC*/		GO
;CPYBWP .BUFPTR,.U01BUFO/*LOCATE BUFFER*//*ZBUFADD*/		GO
;CPYBLA .BUFFER(1:0132),.BLANKS	/*ZCLRBUF*/	GO
;CPYBLA .F01XSET,'00000WRITE*OFL 1201	'/*FILL FEEDBACK*/	GO
;CPYBLA .F01EIND,*OFF/*NO ERROR IND*//*ZFIBPT0*/		GO
;CMPBLA(B) .F01OPEN,*OFF/EQ(.DMEXL2)/*FILE OPEN?*//*ZFIBPT1*/		GO
;CALLI .XRVF01,*,.DRIVRTN/*ZPUT*/		GO
;.00R0001:/*LABEL*//*ZLAB*/		GO
;CPYBLA .OFL2A(2:2),'00'/*SET INTERNAL IN D OFF*//*Z0F20FF*/		GO
;B .OFLWRN/*RETURN*//*Z0FLRTN*/		GO

Figure 203 (Part 4 of 14). Examples of Compiler Debugging Information

Examples of Using Compiler Debugging Options

```

                                                    GO
5763RG1 V3R0M5 940125          IBM SAA RPG/400          QGPL/DATAE          01/25/94 14:40:34  Page 8
;ENTRY .EXCPT INT/*ZLABX*/                                                    GO
;DCL INSPTR .LINERTN/*EXCEPTION RETURN POINTER**/*ZDCLLRT*/                    GO
;BRK '1100  /*BRK POINT**/*ZBRKPT*/                                           GO
;SPACE 3                                                                    GO
; /* QSYSVRT FILE OUTPUT                                                    **/*ZFLCOM*/                    GO
;SETSP .FIBPTR,.F01FIB/*SET FIB PTR**/*ZFIBPT*/                                GO
;CPYBLA .CURROP,C'WRITE'/* SET OP **/*ZCURROP*/                                GO
;CMPBLA(B) *IN1A,*ON/NEQ(.OEC0001)/*OVERFLOW ?*/                              GO
;CALLI .FETCH01,*,.OFETCHR/*FETCH OVERFLOW*/                                  GO
;.OEC0001:/*RET/BR TGT**/*ZPRTOF*/                                           GO
;CPYBLA .PRTCTL,X'0000000100000000'/*SET OPERATION**/*ZPRC*/                  GO
;CPYBWP .BUFPTR,.U01BUFO/*LOCATE BUFFER**/*ZBUFADD*/                          GO
;CPYBLA .BUFFER(1:0132),.BLANKS /*ZCLRBUF*/                                    GO
;BRK '1200  /*BRK POINT**/*ZBRKPT*/                                           GO
;ADDN(S) PAGE ,1 /*INCREMENT**/*ZINC*/                                         GO
;CVTNC .BUFFER(0001:0004), PAGE ,X'02000400000000'/*NUM TO CHAR**/*ZNUM*/    GO
;OR(S) .BUFFER(0004:1),X'F0'/*FORCE POS*/                                       GO
;TSTRPLC .BUFFER(0001:0004),' '/*ZERO SUPPRESS**/*ZZEDT*/                    GO
;CPYBLA .F01XSET,'0000WRITEF*EXCP 1100 /*FILL FEEDBACK*/                      GO
;CPYBLA .F01EIND,*OFF/*NO ERROR IND**/*ZFIBPT0*/                              GO
;CMPBLA(B) .F01OPEN,*OFF/EQ(.DMEXL2)/*FILE OPEN?*//*ZFIBPT1*/                GO
;CALLI .XRVFW01,*,.DRIVRTN/*ZPUT*/                                             GO
;B .LINERTN/*BRANCH**/*ZUNBR*/                                                 GO
;DCL INSPTR .OFETCHR/*OVERFLOW FETCH RETURN**/*ZFETDCL1*/                    GO
;DCL DD .SAVOFOA CHAR(3)/*OVFL IND SAVE**/*ZFETDCL2*/                          GO
;ENTRY .FETCH01 INT/*OVRFLW FETCH SUBR**/*ZFETCH1*/                            GO
;CALLI .OFL,*,.OFLWRTN/*GOTO OVERFLOW ROUTINE**/*ZFETCH3*/                    GO
;B .OFETCHR/*RETURN**/*ZFETCH5*/                                              GO

```

Figure 203 (Part 5 of 14). Examples of Compiler Debugging Information

Examples of Using Compiler Debugging Options

```

RXT0034
PRM has been called.
5763SS1 01/25/94 14:40:34
SEQ INST GENERATED CODE *... .. 1 ... .. 2 ... .. 3 ... .. 4 .. . 5 ... .. 6 ... .. 7 ... .. 8 BREAK
00001 ;
00002 BRK ' .ENTRY ' /*Z1STBRK*/ ; .ENTRY
/* PHASE - QRGSF DATE - 06/26/87 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00003 /* PHASE - QRGSE DATE - 02/12/82 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00004 /* PHASE - QRGSI DATE - 05/15/86 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00005 /* PHASE - QRGSC DATE - 05/04/87 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00006 /* PHASE - QRGSO DATE - 06/27/86 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00007 /* PHASE - QRGD1 DATE - 11/26/87 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00008 /* PHASE - QRGAE DATE - 11/26/87 */ ; .ENTRY
/*SVP*/ ; .ENTRY
00009 ; .ENTRY
00010 0001 000004 0252 0009 0257 SETIEXIT .RPGXIEX,.RPXIEXP /*SET UP INVOCATION EXIT P ; .ENTRY
GM*/ ; .ENTRY
00011 0002 00000A 1011 0234 B .START /*BEGIN OF PROGRAM */ ; .ENTRY
00012 .STOP: ; .ENTRY
00013 0003 00000E 30B2 0007 000D CPYBLA .INVOC SW,*OFF /*ALLOW CALL TO THIS PGM*/ ; .ENTRY
00014 0004 000014 02A1 0000 RTX * /* RETURN */ ; .ENTRY
/*ZSTAR*/ ; .ENTRY
/*START OF THE PROGRAM*/ ; .ENTRY
00015 /* STATIC AREA FOR INDIC/FIELDS */ ; .ENTRY
00016 /* START OF STRUCTURE POINTED BY .DMPRTL*/ ; .ENTRY
00017 ; .ENTRY
00018 DCL DD .RPGM CHAR(1) BDRY(16) INIT ; .ENTRY
00019 DCL DD .FIRSTSW CHAR(1) INIT('0') /* PROGRAM CALLED BEFORE * ; .ENTRY
/ ; .ENTRY
00020 DCL DD .EOJSW CHAR(1) INIT('0') /*PROGRAM WENT TO EOJ*/ ; .ENTRY
00021 DCL DD .DUMPSW CHAR(1) INIT('0') /* DUMP REQUESTED*/ ; .ENTRY
00022 DCL DD .ERRTERM CHAR(1) INIT('0') /*PROGRAM ENDED */ ; .ENTRY
00023 DCL DD .INVOC SW CHAR(1) INIT('0') ; .ENTRY
00024 DCL DD .INVOCER CHAR(4) INIT('8888') ; .ENTRY
00025 DCL SYSPTR .RPGXIEX INIT('QRGXINVX', TYPE(PGM,1)) ; .ENTRY
/*INVOCATION EXIT PROGRAM* ; .ENTRY
/ ; .ENTRY
00026 DCL DD .INVXLVL BIN(2) INIT(1) /*INTERFACE LEVEL FOR QRGX ; .ENTRY
INVX*/ ; .ENTRY
/*END OF STATIC AREA STRUC ; .ENTRY
TURE*/ ; .ENTRY
00027 DCL DD .BLANKS CHAR(140) INIT((140)' ') ; .ENTRY
00028 DCL CON *ON CHAR(1) INIT('1') /* SET/CHECK INDICATORS ON ; .ENTRY

```

Figure 203 (Part 13 of 14). Examples of Compiler Debugging Information

Examples of Using Compiler Debugging Options

```

5763SS1 01/25/94 14:40:34          GENERATED OUTPUT          */          ; .ENTRY
MSGID                               MI INSTRUCTION STREAM SEMANTICS DIAGNOSTICS 01/25/94 14:40:34 Pag 56
5763SS1 01/25/94 14:40:34          GENERATED OUTPUT          01/25/94 14:40:34 Pag 57
OFFSET                               I TEMPLATE DISPLA
00000000    00003272    00000000    0201C4C1    E3C1C540    40404040    40404040    40404040    40404040    F
00000020    40404040    40404040    C0000000    00000000    00000000    00010000    00000000    00000000    00000000
00000040    00000000    00000000    000502FD    DD000400    00000000    00000000    00000000    00000000    00000000
00000060    208000FC    00000000    00000000    019602B1    00000100    00000C14    0000171C    00000008
00000080    0000015E    00003114    00000000    00000A4B    000026C8    00000000    00000196    00000000
000000A0    00000000    00000000    00000000    00000000    00000000    00000000    00000000    00000000
000000C0    00000000    00000000    00000000    00000000    00000000    00000000    00000000    00000000
000000E0    00000000    00000000    00000000    00000000    00000000    00000000    00000000    00000000
00000100    0000AD2    02520009    02571011    023430B2    0007000D    02A10000    21320050    020D10B2
00000120    021F000C    1042020B    20021022    022F0059    0283004C    022D0000    1C464000    021623CA
00000140    026A1C46    C0000216    20000059    10B2021F    000D1011    005C30B2    021F000D    10B2002E
00000160    002D10B2    002D005A    10B20231    20D710B2    0232002D    1CC24000    0005000C    02691CC2
00000180    4000005B    000C0269    10110061    30B20030    005E10B2    00120014    11930013    00141CC2
000001A0    40000016    000D005F    10BE0015    000C1011    006430BE    0015000D    1CC24000    0060000C
000001C0    006410B2    0060000C    10110068    30B20231    20D710B2    0232002D    10B2003A    00AB10B2
000001E0    003B00E9    1CC24000    005B000C    006210B2    005B000C    10110269    30B20232    00631011
00000200    026930B2    00300065    30B20030    006730B2    00300069    1CC24000    0016000C    02660293
00000220    00740000    006A30B2    0030006C    10B20016    000C1042    006D2001    3C461000    006D006F
00000240    00700293    007A0000    007B3143    006D2001    1011006E    3143001B    00711011    007230B2
00000260    00300073    1011005D    3CC24000    0013000D    00780082    00840154    10B20022    007510B2
00000280    01C80076    01320087    01A010B2    608F2001    2084000B    10B20190    007710B2    016C000D
000002A0    1CC24000    0174000D    01FF0293    01ED0000    009030B2    60112002    20020079    1011006A
000002C0    20820084    015410B2    0022007C    1CC2C000    0013000C    007D0293    00830000    008130B2
000002E0    01C8007E    01320087    01A010B2    608F2001    2084000B    11430019    200110A3    608F2001
00000300    20040019    007F1197    608F2004    200120F0    10A2608F    20012004    204010B2    01900080
00000320    10B2016C    000D1CC2    40000174    000D01FF    029301ED    00000090    3011007B    22930074
00000340    0000006A    30110081    21320085    00AC0132    0086011B    1042008E    409D2001    013201CD
00000360    4089008E    013201CF    00AE0132    01CE00AF    104201D4    01C91042    01D601CA    104201D3
00000380    01CB1042    01D501CC    10B20220    000C0283    01CD01D0    000010B2    0220000D    10B20141
000003A0    01420083    0088011F    014410B2    01100135    1CC24000    00C4000C    01D10132    01D900B2
000003C0    10B201DB    00B901E3    1C461000    00B800B9    01E310B2    81DA01D9    01E23042    00B801D8
000003E0    104201D8    01CC0293    01E00000    01C73011    01D13C46    400001D8    200001E7    1C462000
00000400    01D800B8    01E41C46    100001D8    00B801E1    1C461000    00B800B9    01E31011    01E43C46
00000420    900001D8    00B901E3    1C461000    00B800B9    01E310B2    81DA01D9    01E23042    00B801D8
00000440    101101E7    3CC24000    00DE01E5    01E310B2    81DA01D9    01E61011    01E33C46    400001D7
00000460    200001C7    1C461000    00B800B9    01EA1143    00B801D7    1C469000    00B800B9    01C710B2
00000480    81DA01D9    01E81011    01C73011    01C73143    00B801D7    1C469000    00B800BA    01C71147
000004A0    00B800BA    101101C7    3CC24000    0174000D    01FF0293    01DD0000    01D13011    009030B2
000004C0    0220000D    10B200E1    022110B2    00E301F1    1CC24000    01EF000C    01FB1CC2    400000DD
000004E0    01F201FD    1CC24000    00DD01F3    01FD0132    008500AC    1CE2C000    011B0000    01F410B2
00000500    00CB000D    10B200C1    000D10B2    00C2000C    10B200DB    000C10B2    00D2000C    10B200DA
00000520    000D1011    01F730B2    01410142    00B30088    011F0105    10B20110    01351011    01F730B2
00000540    00DC01F6    30B20225    000D10B2    0220000D    10B2003A    00AB10B2    003B00E9    10B20231
00000560    20C610B2    023300DC    01320213    00841CC2    400000C3    000C0090    10114202    00B33CC2
00000580    200000DC    01F90268    1CC21000    00DC01FA    02681011    009030B2    00DC01FC    10B20231
000005A0    20C610B2    023300DC    01320213    008410B2    003A00AB    10B2003B    00E91011    01F830B2
000005C0    00DC01FE    101101F7    30B200DC    02001011    01F730B2    01410142    00830088    011F0105
000005E0    10B20110    01351011    01F73CC2    C0000007    000C0235    1042022B    200210B2    02400008
00000600    1011026B    30B20007    000C10B2    00300236    10B2002F    02371042    022B2000    1CC24000
00000620    0003000C    023C0547    020E0000    20010132    00270000    1042020B    20020283    0230022C
00000640    00001C46    40000216    23CA0282    03EF0298    029C2001    10B20028    003F0082    02520245
00000660    1042020B    20010283    02550254    000010B2    0232002D    1C46C000    02162000    026A10B2
00000680    01C5002C    10B201C6    00361042    00232001    10220090    023A2132    00844204    002310B2
000006A0    00E90239    10B200C3    000D1011    025E3D47    C0000023    20010238    10B2021E    000B10B2
000006C0    002F023B    10B201EF    000D1042    022B2000    23EF0298    029C2001    1CC24000    0003000C
000006E0    005D10B2    0003000C    10110072    3042020B    20020283    023D0243    00001C46    40000216

```

Program DATAE is placed in library QGPL. 00 highest Error-Severity-Code.
RXT0048

***** END OF COMPILATION *****

Figure 203 (Part 14 of 14). Examples of Compiler Debugging Information

IRP Layout

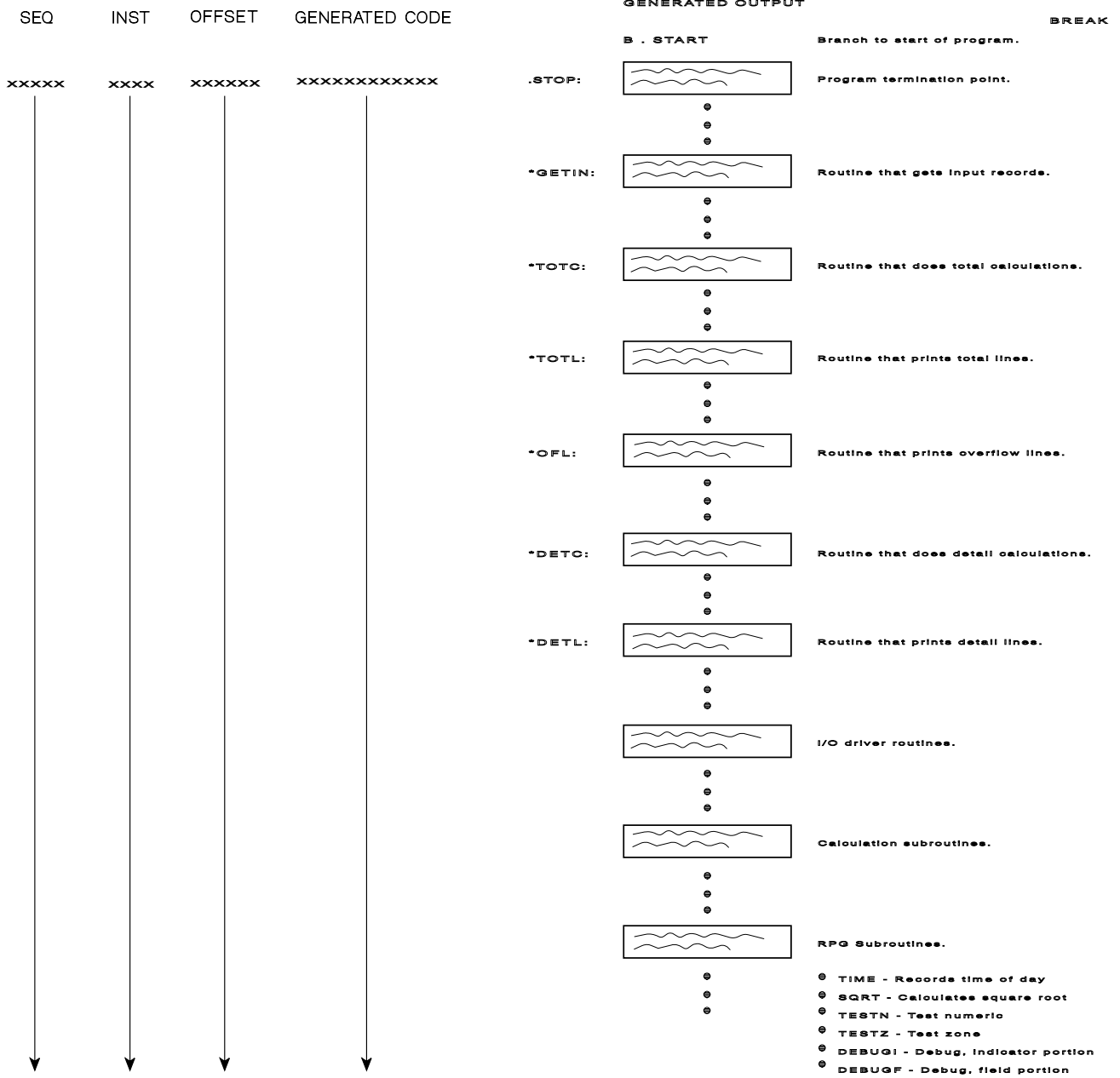


Figure 205 (Part 1 of 2). IRP Layout

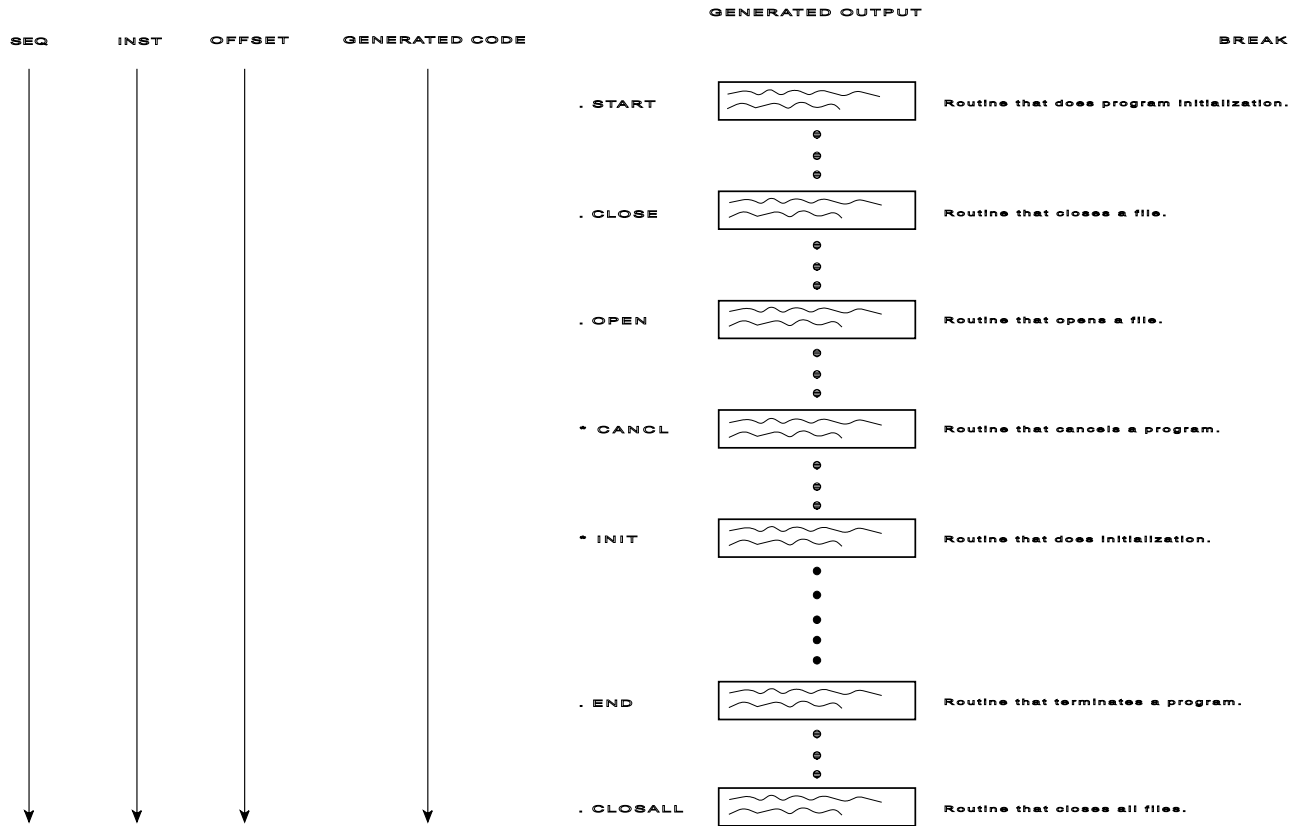


Figure 205 (Part 2 of 2). IRP Layout

Auto Report Program

The automatic report program consists of the phases listed in Table 23. Those phases that have a U in the third column of Table 23 process unconditionally, and those phases that have a C in the third column process only if they are required for the program being compiled. Figure 206 on page 519 shows the order in which the phases run.

The automatic report program major data areas are a common area (XREGN), a field name table (FLDTBL), and two buffers (BUFFER1 and BUFFER2).

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRPT0000	Command interface phase	U
QRPT0001	Root controlling phase	U
QRPT0002	I/O control phase and /COPY function	U
QRPT0003	Diagnostic phase for *AUTO	C
QRPT0004	Diagnostic phase for *AUTO	C
QRPT0005	*AUTO generation phase	C
QRPT0008	Wrap-up phase	U

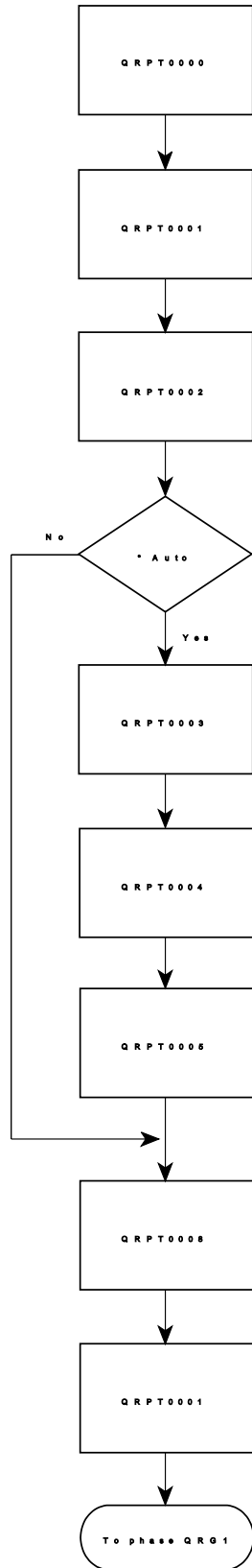


Figure 206. Order of Processing of Automatic Report Program Phases

Auto Report Program

Appendix B. RPG/400 and AS/400 RPG II System/36-Compatible Functions

This appendix contains information about additional RPG/400 functions that are not available with the AS/400 RPG II System/36-Compatible compiler.

Language Enhancements

The enhancements to the RPG/400 language over the AS/400 RPG II System/36-Compatible compiler are:

- *Externally described data:* Fields of a file are described to the OS/400 system through data-description specifications (DDS). The advantage of externally described data is that the fields of the file need be described only once to the OS/400 system and need not be described for each program that uses the file. If the file description changes, you can change its description in one place and then recompile the programs that use it.

The following file processing operation codes are available to the RPG/400 programming language:

- DELET (Delete Record)
- SETGT (Set Greater Than)
- UPDAT (Change Existing Record)
- WRITE (Create New Records)
- REDPE (Retrieve Prior Equal Record).

The following control operation codes are available to the RPG/400 programming language:

- CLOSE (Close Files)
 - FEOD (Force End of Data)
 - OPEN (Open File for Processing).
- *Work station support:* Allows the specification of the RPG/400 device name, WORKSTN, which is used for input and output from the display work station. The operation codes that support direct control over specific work station formats include:

- EXFMT (Execute Format)
- WRITE (Create New Records).

In addition, the RPG/400 programming language supports the subfile capability in WORKSTN support with the following operation codes:

- CHAIN (Random Retrieval from a Subfile)
- READC (Read Next Changed Record)
- UPDAT (Change Existing Record)
- WRITE (Create New Records).

See Chapter 8, Using WORKSTN Files for further information on the WORKSTN device.

- *ANDxx/ORxx operations:* Allow you to specify a more complex decision condition than a simple A to B comparison with the IFxx, DOUxx, and DOWxx operations.

Language Enhancements

- *CABxx (compare and branch) operation:* Allows you to do a compare and branch in one operation and eliminates the need to set and test resulting indicators.
- *CLEAR operation code:* Allows you to set elements in a data structure or a variable to zero, blank, or 0 (for indicators), depending on the field type.
- *Commitment control:* Allows you to group file operations by using the COMMIT and ROLBK operation codes. See Chapter 6, “Commitment Control” for information.
- *The compiler /COPY function:* Allows the merging of members from more than one source file during a compile. No sorting or modification of records can be done.
- *Data area operations:* The IN (Retrieve a Data Area) and OUT (Write a Data Area) operations allow you to access a data area and optionally allow you to lock or unlock a data area. The UNLCK operation (Unlock a Data Area) unlocks one or all locked data areas in a program.
- *Data structures:* The RPG/400 programming language supports two additional data structures:
 - Program status data structure, which provides program exception and error information to the program
 - Externally described data structures.
- *Data structure initialization:* Allows you to initialize an entire data structure, characters to blank, numerics to zero.
- *Date, time and timestamp SAA data types* are supported.
- *DSPLY (Display Function) operation:* Allows access to a display device without the use of a display device file and allows access to the message handler.
- *DSPPGMREF:* The referenced object information provided via the CL command DSPPGMREF now includes called programs in addition to files and data areas (not data structures).
- *ENDyy operation:* Provides for improved program readability by allowing you to indicate the type of structure (CASxx, D0, DOUxx, DOWxx, IFxx, or SELEC) the END operation is closing.

Note: When the result field is too small to hold the value of the limit (factor 2) plus the increment (END factor 2), a numeric data overflow may cause your RPG/400 program to loop forever. A System/36 RPG II program, however, will loop the required number of times specified by the start, limit, and increment values.
- *Exception/error handling:* Allows you to control the program logic by using the program exception/error subroutine *PSSR if program exception/errors occur while the program is running. See “Exception/Error Handling” on page 70 for detailed information on program exception and error handling.
- *Figurative constants:* Allow you to use additional RPG/400 reserved words that can be specified without specifying length and decimal positions because the implied length and decimal positions of a figurative constant are the same as that of the associated field. They are:
 - *ALL'a..'
 - *ALLX'x1x2..'
 - *HIVAL

- *LOVAL
 - *ON
 - *OFF
- *Floating minus edit codes:* Four new edit codes (N, 0, P, and Q) are provided for editing negative numbers with a floating minus (–) sign. The minus sign, if specified, is printed to the left of the most significant digit or floating currency symbol.
 - *Graphic SAA data types* are supported.
 - *Hexadecimal literals* are supported.
 - *Indicators referred to as data (*IN, *INxx):* Allows you an alternative method of referring to and manipulating indicators. The indicator array of indicators 01 through 99 (*IN) and the indicator field (*INxx) reduce coding and provide a simplified approach to many program processing requirements.
 - *Indentation bars:* Allow you to specify that D0 and SELEC statements and IF-ELSE clauses be indented on program listings for enhanced program readability.
 - *Initialization subroutine:* Allows you to specify a particular subroutine to be run at program initialization time.
 - *ITER and LEAVE operations:* ITER allows you to end the current iteration of a D0-group and start the next iteration. The LEAVE operation allows you to transfer control from within a D0-group to the statement following the corresponding ENDy operation.
 - *KLIST (Define a Composite Key)/KFLD (Define Parts of a Key) operations:* Allow you to indicate the name by which a composite key can be specified and the fields that comprise the composite key.
 - *Multiple occurrence data structures:* A data structure can appear n times in a program. See “Multiple Occurrence Data Structure” on page 223 for further information on multiple occurrence data structures.
 - *Named constants:* Allows you to specify a name to a constant. This name represents a specific value which cannot be changed when the program is running. You can specify named constants in Factor 1 and Factor 2 in the calculation specifications and in Constant or Edit Word fields in the output specifications. See “Named Constants” on page 237 for more information on the use of named constants.
 - Null-capable field support.
 - *Numeric variables:* The RPG/400 programming language supports numeric variables up to and including 30 digits. The maximum number of decimal digits allowed remains 9.
 - *Operation Extender (position 53):* N allows you to specify reading records without locking them. This is supported on five operations: READ, READE, READP, REDPE and CHAIN. P allows you to pad the result field after performing a CAT, SUBST,XLATE, MOVE, MOVEL or MOVEA operation.
 - *Overflow indicators:* You can use indicators 01 through 99 as overflow indicators on both program-described and externally described PRINTER files (in addition to 0A through 0G and 0V for program described files). See Chapter 5, “General File Considerations” for further information on overflow indicators.

Language Enhancements

- *Printer control (PRTCTL) option:* You can (1) dynamically specify space and skip operations instead of using values on the output specifications, and (2) access the current line value within the program. This option is allowed only with program-described files. See Chapter 5, “General File Considerations” for further information on the PRTCTL options.
- *Program Initialization Parameter.* Allows you to pass parameters in a pre-started program.
- *Program Initialization Parameters Data Area.* Allows you to predefine and store Program Initialization Parameters.
- *REDPE operation code:* Allows you to retrieve the prior sequential record from a full procedural file if the key of the record matches the search argument in Factor 1 (positions 18 to 27). You must also specify a file name or record name in Factor 2. You can also specify in the Result field a data structure into which the record can be read.
- *RESET operation code:* Allows you to set elements in a data structure, or a field, back to their values at the end of program initialization. When RESET is specified for a structure or a variable, a snapshot of that variable or structure is taken at the end of the *INIT cycle. The value is then used to reset the structure or variable.
- *Resulting indicators with MOVE and MOVE L operations:* You can specify resulting indicators on MOVE and MOVE L statements. They eliminate the need for additional operations to check for blank, zero, or plus/minus conditions.
- *Retry on timeout:* The RPG1218 error message has been updated to allow a retry to be requested when a timeout occurs on a record lock request.
- The *SELEC* operation allows you to specify the conditions to select which group of operations will be processed.
 - SELEC operation begins the SELEC group.
 - WHxx operation of a SELEC group allows you to determine where control passes after the SELEC operation is processed.
 - OTHER operation allows you to specify the sequence of operations to be processed if no WHxx condition is satisfied.
- *SEQ files:* Allow you to perform sequential input/output to any sequentially organized file, such as database, diskette, tape, savefile, or printer file. The actual device used is specified by an AS/400 Control Program Facility override command.
- *SPECIAL file with PLIST operation:* Allows you to specify an input/output device that is not directly supported by the RPG/400 programming language. You can add additional parameters to the RPG/400-created parameter list with the use of the PLIST and PARM operation codes. See “Special File” on page 101 for information on the SPECIAL device.
- *String operations CAT, CHECK, CHEKR, SCAN, SUBST, and XLATE:* The CAT operation allows you to concatenate two character strings. The result field can be a field name, array element, data structure, or table name. The SCAN operation allows you to scan a character string for a specified substring starting at a specific location for a specific length. The SUBST operation allows you to extract a substring from a specified source string starting at a specific location. The XLATE operation allows you to translate characters in factor 2 according to the FROM and TO strings in factor 1. The CHECK operation allows you to verify that

each character in factor 2 is among the valid characters in factor 1. The CHEKR operation provides similar function to CHECK but in the reverse direction (right to left).

- *SUBR23R3*: The message-retrieving subroutine has been enhanced to allow the system maximum of 3000 characters of second level text to be retrieved and will support message I containing 0-9 or A-F for the message identifier.
- *Subfield initialization*: Allows you to initialize a data structure subfield to a specific value.
- *TESTN (Test Numeric) operation*: Allows you to validity check a character field to ensure that it contains zoned decimal digits and blanks.
- *User-defined edit codes (5 through 9)*: Allow for unique customer- or nation-oriented editing. The user-defined edit codes are defined to the AS/400 system.
- *TIME* operation: Allows for 14 digits.
- *UNLCK* operation: Allows the last locked record to be unlocked for an update disk file. Records can still be unlocked by processing output operations defined by output specifications with no field names included.
- *Variable length fields* are supported.
- *4-digit year* is supported.

Note: For more information on RPG/400 enhancements, see the *RPG/400 Reference*.

Appendix C. Data Communication

The AS/400 system RPG/400 operations allow data communication through the WORKSTN file using ICF. There are no RPG/400 operations or specifications unique to data communication. The kinds of data communication supported through the WORKSTN file using ICF include APPC, Asynchronous, BSCCL, Finance, Intrasystem communications, Retail, and SNUF. The WORKSTN file used for data communication must be defined as a full procedural file (F in position 16 of the file-description specifications). Here is a list of some operation codes and corresponding data communication functions supported by the WORKSTN file:

Operation Codes	Data Communication Functions
OPEN (Open File for Processing)	Open (Input and Output), error recovery
CLOSE (Close Files)	Close (Permanent), error recovery
EXFMT (Execute Format)	Write/Read (Wait)
READ (Read a Record)	Read (Wait)
WRITE (Create New Records)	Write (Wait)
ACQ (Acquire)	Acquire a device, error recovery
REL (Release)	Release a device, error recovery

For more information on remote communication, see the *ICF Programmer's Guide*. For information on RPG/400 operations, see the *RPG/400 Reference*.

Exception and Error Handling with ICF Files

When a program has a run-time error, you can cancel the program. (See "Exception/Error Handling" on page 70.) If you do, all of the program's files are closed abnormally. For ICF files, the other end of the communications line is notified that there is a failure and the communication has ended abnormally. For a shared ICF file, the notification is sent when the last program closes the file.

Instead of canceling a program, you can continue processing (for example, in an error handling subroutine). It is your responsibility to recover from the error.

Communications Error Recovery

You may be able to recover from a device communications error, when using a multiple device file, by processing a REL (Release) operation followed by an ACQ (Acquire) operation for the device in error.

You may be able to recover from a file communications error by processing a CLOSE operation followed by an OPEN operation for the file in error. With shared files, the program must be closed for all the programs sharing the file, and then opened again.

For further information, see the *ICF Programmer's Guide*.

Appendix D. Distributed Data Management (DDM) Files

Distributed Data Management (DDM) allows you to access data files that reside on remote systems with a communications network that supports DDM. The RPG/400 compiler supports DDM files: you can retrieve, add, update or delete data records in a file that resides on another system.

For more information about accessing remote files, refer to the *DDM Guide*.

Distributed Data Management (DDM) Files

Appendix E. System/38 Environment Option of the RPG Compiler

This appendix describes how the System/38 environment option of the RPG compiler supports the same RPG syntax as the System/38 RPG III compiler, and the System/38 object naming conventions. The remainder of the appendix discusses differences between the System/38 RPG and the System/38 environment option of the RPG compiler, differences between the System/38 environment option of the RPG compiler, and the AS/400 system RPG/400 compiler, and the file types supported by each compiler.

Differences between System/38 RPG III and the System/38 Environment Option of the RPG Compiler

The System/38 environment option of the RPG compiler differs from the System/38 RPG III compiler in the following ways:

- The source-member name on the create command is used for the name of the spooled file that contains the compiler output.
- The format of the date used when the program is run is in the format described in the job value (set by the job description or by the CHGJOB command) rather than the system value.
- Numeric arrays are allowed on the MOVEA operation code.
- 30 digit numerics are supported.
- Card devices are not supported. If the RPG source specifies any of the card device syntax, an error of severity 30 occurs when you compile the program.
- Listing format differs.
- Message format is different for both compile and run time messages.

For those items that are the same as System/38 RPG III support, see the *System/38 RPG III Reference Manual and Programmer's Guide*, SC21-7725.

Differences between the System/38 Environment Option of the RPG Compiler and RPG/400 Compiler

Use RPG38 as the source type for the member containing the RPG source statements. The programs created will have the same values in the object attribute.

If you are using the CRTRPGPGM or CRTRPTPGM command directly, be sure to use the command in the QSYS38 library.

Most of the information in this User's Guide is applicable to the System/38 environment option of the RPG compiler support, with the following exceptions:

- The enhancements to the RPG/400 compiler over System/38 environment option of the RPG compiler are:
 - *CLEAR operation code*: Allows you to set elements in a data structure or a variable to zero, blank, or 0 (for indicators), depending on the field type.

System/38 Environment Option of the RPG Compiler

- *Date, time and timestamp SAA data types* are supported.
- *DSPPGMREF*: The referenced object information provided via the CL command DSPPGMREF now includes called programs in addition to files and data areas (not data structures).
- *ENDyy* operation: Provides for improved program readability by allowing you to indicate the type of structure (CASxx, D0, D0Uxx, D0Wxx, IFxx, or SELEC) the END operation is closing.
- *Figurative constants*: Allow you to use additional RPG/400 reserved words that can be specified without specifying length and decimal positions because the implied length and decimal positions of a figurative constant are the same as that of the associated field. They are:
 - *ALLX'x1x2..'
 - *0N
 - *0FF
- *Floating minus edit codes*: Four new edit codes (N, 0, P, and Q) are provided for editing negative numbers with a floating minus (–) sign. The minus sign, if specified, is printed to the left of the most significant digit or floating currency symbol.
- *Graphic SAA data types* are supported.
- *Hexadecimal literals* are supported.
- *Indentation bars*: Allow you to specify that D0 and SELEC statements and IF-ELSE clauses be indented on program listings for enhanced program readability.
- *ITER and LEAVE operations*: ITER allows you to end the current iteration of a D0-group and start the next iteration. The LEAVE operation allows you to transfer control from within a D0-group to the statement following the corresponding ENDyy operation.
- *Named constants*: Allows you to specify a name to a constant. This name represents a specific value that cannot be changed when the program is running. See “Named Constants” on page 237 for more information on the use of named constants.
- *Null-capable fields* are supported.
- *Operation Extender (position 53)*: 'N' allows you to specify reading records without locking them. This is supported on five operations: READ, READE, READP, REDPE, and CHAIN. 'P' allows you to pad the result field after performing a CAT, SUBST, XLATE, MOVE, MOVEL, or MOVEA operation.
- *Override to print file (OVRPRTF)*: When using the RPG/400 compiler, the OVRPRTF CL command causes a spool file with the same name as the file in the OVRPRTF command to be generated when the program is executed. The generated spool file name can then be changed by specifying a value for the SPLFNAME keyword in the OVRPRTF command. The SPLFNAME keyword is not available in the System/38 environment.
- *Program Initialization Parameters (PIP)*. Allows you to pass parameters in a pre-started program.
- *Program Initialization Parameter Data Area (PDA)*. Allows you to predefine and store Program Initialization Parameters.

- *REDPE operation code*: Allows you to retrieve the prior sequential record from a full procedural file if the key of the record matches the search argument in Factor 1 (positions 18 to 27). You must also specify a file name or record name in Factor 2. You can also specify in the Result field a data structure into which the record can be read.
- *RESET operation code*: Allows you to set elements in a data structure, or a field, back to their values at the end of program initialization. When RESET is specified for a structure or a variable, a snapshot of that variable or structure is taken at the end of the *INIT cycle. The value is then used to reset the structure or variable.
- *Retry on timeout*: The RPG1218 error message has been updated to allow a retry to be requested when a timeout occurs on a record lock request.
- The *SELEC* operation allows you to specify the conditions to select which group of operations will be processed.
 - SELEC operation begins the SELEC group.
 - WHxx operation of a SELEC group allows you to determine where control passes after the SELEC operation is processed.
 - OTHER operation allows you to specify the sequence of operations to be processed if no WHxx condition is satisfied.
- *String operations CAT, CHECK, CHEKR, SCAN, SUBST, and XLATE*: CAT allows you to concatenate two character strings. The result field can be a field name, array element, data structure, or table name. SCAN allows you to scan a character string for a specified substring starting at a specific location for a specific length. SUBST allows you to extract a substring from a specified source string starting at a specific location. XLATE allows you to translate characters in factor 2 according to the FROM and TO strings in factor 1. The CHECK operation allows you to verify that each character in factor 2 is among the valid characters in factor 1. The CHEKR operation provides similar function to the CHECK operation code, but in the reverse direction (right to left).
- *SUBR23R3*: The message-retrieving subroutine has been enhanced to allow the system maximum of 3000 characters of second level text to be retrieved and will support message I containing 0-9 or A-F for the message identifier.
- *TIME* operation: Allows for 14 digits.
- *UNLCK* operation: Allows the last locked record to be unlocked for an update disk file. Records can still be unlocked by processing output operations defined by output specifications with no field names included.
- *Variable-length fields* are supported.
- *4-digit year* is supported.
- File and program names must follow the System/38 naming convention (object.library) on the /COPY statement and in the FREE, CALL, and DSPLY operation codes.
- The System/38 environment option of the RPG compiler allows you to write to an existing relative record number while the RPG/400 compiler does not support this function and will give a run-time error.

System/38 Environment Option of the RPG Compiler

- The format of the information returned from a POST operation to a specific device is the same as the support on System/38.
- The create commands are the same as the System/38.

Table 24 shows the differences between the RPG/400 compiler and the System/38 environment option of the RPG compiler environments.

<i>Table 24 (Page 1 of 2). Differences between the RPG/400 Compiler and the System/38 Environment Option of the RPG Compiler</i>				
RPG/400 Compiler Parameter	System/38 Environment Option of the RPG Compiler Parameter	RPG/400 Compiler Options	System/38 Environment Option of the RPG Compiler Options	Comments
REPLACE	N/A			New parameter
		*YES	N/A	New option
		*NO	N/A	New option
TGTRLS	N/A			New parameter
		*CURRENT	N/A	New option
		*PRV	N/A	New option
		release-level	N/A	New option
AUT	PUBAUT			AUT replaces PUBAUT
		*LIBCRTAUT	N/A	New option
		*CHANGE	*NORMAL	*CHANGE replaces *NORMAL
		*USE	N/A	New option
		*ALL	N/A	New option
		*EXCLUDE	*NONE	*EXCLUDE replaces *NONE
		authorization list-name	N/A	New option
PGM	PGM			Existing parameter
		*CURLIB	N/A	New option/new default
SRCFILE	SRCFILE			Existing parameter
		*CURLIB	N/A	New option
PRTFILE	PRTFILE			Existing parameter
		*CURLIB	N/A	New option
OPTION	OPTION			Existing parameter
		*SECLVL	N/A	New option
		*NOSECLVL	N/A	New option
SAAFLAG	N/A			New parameter
		*NOFLAG	N/A	New option
		*FLAG	N/A	New option
INDENT	N/A			New parameter

Table 24 (Page 2 of 2). Differences between the RPG/400 Compiler and the System/38 Environment Option of the RPG Compiler

RPG/400 Compiler Parameter	System/38 Environment Option of the RPG Compiler Parameter	RPG/400 Compiler Options	System/38 Environment Option of the RPG Compiler Options	Comments
		*NONE	N/A	New option
		character-value	N/A	New option
CVTOPT	N/A			New parameter
		*NONE	N/A	New option
		*VARCHAR	N/A	New option
		*DATETIME	N/A	New option
		*GRAPHIC	N/A	New option
ALWNULL	N/A			New parameter
		*NO	N/A	New option
		*YES	N/A	New option

When you convert a System/38-compatible program to an AS/400 program, you can, and in some cases, must, use the support as described in this manual for the above list of items.

File Types Supported by Each Compiler

Similar to programs, files (object type *FILE) can be created with a System/38 or an OS/400 system attribute. System/38 files have a 38 added to their object attribute. OS/400 system files do not. See the *System/38 Environment Programmer's Guide/Reference* for information on object attributes for the various types of files.

Any type of RPG/400 program can use files created with either attribute. The System/38 environment option of the RPG compiler programs are not restricted to using System/38 files. RPG/400 programs are not restricted to using OS/400 files. For example, a System/38 environment option of the RPG compiler program can use an OS/400 system display file or database file. An AS/400 system RPG/400 program can use a System/38 display file or database file.

The ability to mix file types and program types also applies to the communications file types for an RPG/400 program, even though the System/38 file types are different from the OS/400 system file types. An AS/400 program can use a communications, BSC or mixed device file.

There are some items worth noting about such combinations. An AS/400 program that uses an BSC, communications, or mixed-device file uses the AS/400 system RPG/400 *STATUS values. Some of these values are set based on major/minor return codes. Also, the format of the information returned from a POST operation to a specific device is the AS/400 RPG/400 version. Some of the items, such as

File Types Supported by Each Compiler

remote-location name, are not returned because the file types do not support a remote-location name.

Note: ICF files are not supported with a System/38 environment option of the RPG/400 compiler program. Using an ICF file in the System/38 environment may cause unpredicted results.

Appendix F. Examples of Using Arrays

This appendix gives several examples of using arrays. For detailed information on how to code an array, how to specify the initial values of the array elements, and how to change the values of an array, refer to the *RPG/400 Reference*.

The following figures illustrate the ways of using arrays:

Figure	Array Examples
Figure 207 on page 538	Building an array using fields as indexes
Figure 208 on page 539	Building an array using fixed indexes
Figure 209 on page 540	Calculating totals without arrays
Figure 210 on page 542	Calculating totals with arrays
Figure 211 on page 543	Using arrays to format field output
Figure 212 on page 545	Printing one array element per line
Figure 213 on page 546	Printing more than one array element per line

Examples of Using Arrays

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* This example illustrates a method of building an array using
E* fields in input records as indexes. The array has 12 elements;
E* each element is 5 positions long. The array could be defined with
E* any number of elements (to a maximum of 99) without additional
E* input specifications.
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E
AR
12 5
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* To build an array using field indexes, assign different values to
I* fields X1 through X10 on each input record type 03 and to fields
I* X1 and X2 on each input record type 04. Succeeding type 03
I* records can then load 10 additional elements in array AR, up to
I* the maximum defined in the array; each type 04 record can load
I* two additional elements. Blanks and other fields can appear on
I* the input records because the array elements and their indexes
I* are identified by the From and To entries. To set up the array
I* in this manner requires a minimum of coding and no calculations.
I* However, extra work is required to set up the indexing scheme for
I* the input records.
I*
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrP1MnZr...*
IFILE1 AA 03 80 C1
I.....PFromTo++DField+L1M1FrP1MnZr...*
I
2 30X1
I
4 8 AR,X1
I
9 100X2
I
11 15 AR,X2
I
16 170X3
I
18 22 AR,X3
I
"
I
"
I
"

```

More Array Elements

Figure 207 (Part 1 of 2). Building an Array Using Input Fields as Indexes

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          54  55X10
I          56  60 AR,X10
I      BB  04  80 C2
I          2   30X1
I          4   8  AR,X1
I          9  100X2
I         11  15  AR,X2
I*
```

Figure 207 (Part 2 of 2). Building an Array Using Input Fields as Indexes

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* This example shows how eighteen 5-character elements of array
E* AR1 are loaded with only two specification lines.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E          AR1          30  5
E*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
I*
I* In these input specifications, the remaining elements of AR1
I* are loaded one after another until the array is full. Each
I* additional element is coded on a separate line. Each new record
I* requires a separate means of identification. For example, if
I* another 03 record followed the first, the fields on the second
I* record would overlay the fields read in from the first record.
I* This method works well for small arrays.
I*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IFILE1  AA  03 100 C1
I.....PFromTo++DField+L1M1FrP1MnZr...*
I          1  90 AR1
I      BB  04 100 C2
I          1   5 AR1,19
I          6  10 AR1,20
I          "
I          "
I          " More Array Elements
I          "
I          "
```

Figure 208. Building an Array Using Fixed Indexes

Examples of Using Arrays

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* The specifications in this example calculate three levels of
C* totals. As they are read from input records, the fields FIELDA,
C* FIELDB, FIELDC, and FIELDD are added to the first-level totals
C* L1A, L1B, L1C, and L1D. These first-level totals are added at
C* the time of an L1 control break to totals L2A, L2B, L2C, and L2D.
C* Similarly, at an L2 control break, the second-level totals are
C* added to third-level totals L3A, L3B, L3C, and L3D. In addition,
C* as control breaks occur, L1, L2, and L3 total output is processed;
C* and total fields are set to zeros after they are written to the
C* output device.
C*
CL0N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++*
C          FIELDA    ADD  L1A      L1A      62          ADD TO L1 TOTLS
C          FIELDB    ADD  L1B      L1B      62
C          FIELDC    ADD  L1C      L1C      62
C          FIELDD    ADD  L1D      L1D      62
CL1       L1A        ADD  L2A      L2A      62          ADD TO L2 TOTLS
CL1       L1B        ADD  L2B      L2B      62
CL1       L1C        ADD  L2C      L2C      62
CL1       L1D        ADD  L2D      L2D      62
CL2       L2A        ADD  L3A      L3A      62          ADD TO L3 TOTLS
CL2       L2B        ADD  L3B      L3B      62
CL2       L2C        ADD  L3C      L3C      62
CL2       L2D        ADD  L3D      L3D      62
C*

```

Figure 209 (Part 1 of 2). Calculating Totals without Arrays

```

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
0      T 20      L1
0.....N01N02N03Field+YBEnd+PConstant/editword+++++++...*
0              L1A  KB  15
0              L1B  KB  30
0              L1C  KB  45
0              L1D  KB  60
0      T 20      L2
0              L2A  KB  15
0              L2B  KB  30
0              L2C  KB  45
0              L2D  KB  60
0      T 20      L3
0              L3A  KB  15
0              L3B  KB  30
0              L3C  KB  45
0              L3D  KB  60

```

Figure 209 (Part 2 of 2). Calculating Totals without Arrays

Examples of Using Arrays

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E*
E* The three levels of totals shown in this example are calculated
E* with arrays.
E*
E...FromfileTofile++Name++N/rN/tbLenPDSArnamLenPDSComments+++++++*
E           SL1           4 6 2           L1 TOTALS
E           SL2           4 6 2           L2 TOTALS
E           SL3           4 6 2           L3 TOTALS
E*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C* Note the reduction in coding required to specify the functions.
C* For example, the L1 control break in the following calculation
C* specifications fills the same function as the 4 lines of L1
C* in the calculation specifications shown in the previous example.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C           FIELDA   ADD SL1,1   SL1,1           ADD FOR L1 TOTL
C           FIELDB   ADD SL1,2   SL1,2
C           FIELD C   ADD SL1,3   SL1,3
C           FIELD D   ADD SL1,4   SL1,4
CL1        SL1       ADD SL2       SL2           ADD FOR L2 TOTL
CL2        SL2       ADD SL3       SL3           ADD FOR L3 TOTL
C*
```

Figure 210 (Part 1 of 2). Calculating Totals with Arrays

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
O*
O* Similarly, the output specifications are reduced from 15 lines
O* to 6. The method using arrays results in only two positions
O* between array elements.
O*
OName++++DFBASbSaN01N02N03Excnam.....*
O           T 20       L1
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
O           SL1       KB 60
O           T 20       L1
O           SL2       KB 60
O           T 20       L1
O           SL3       KB 60
O*
```

Figure 210 (Part 2 of 2). Calculating Totals with Arrays

The following figure shows an example of using arrays to format field output.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++Name++N/rN/tbLenPDSArnamLenPDSComments+++++++*
E           ARA           4 5 0
E           ARB           5 10
E           ARC           6 4 2
E*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.....*
IIN      AA  01  80  C
I        OR  02  80  C1
I.....PFromTo++DField+L1M1FrP1MnZr...*
I           51  74  ARC
I           1  20  ARA      01
I           1  50  ARB      02
I*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excnam.....*
OOUT     D  1      01
O        OR      02
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
O           ARC      84 '0 . &CR'
O           01      ARA,1 Z  89
O           02      ARB,X1  100
O*
```

Figure 211 (Part 1 of 2). Using Arrays to Format Field Output

Examples of Using Arrays

This figure illustrates the use of three arrays to format field output. The arrays are defined as follows:

Array Name	Number of Elements	Element Length
ARA	4	5
ARB	5	10
ARC	6	4

Array ARA is contained in the input records with record identifying indicator 01, ARB in the records with record identifying indicator 02, and ARC in both types of records. Array ARC and the element of array ARA are to be included together in an output record as are arrays ARC and an element (identified by X1) of array ARB. Every element in array ARC is edited according to the edit word '0b.b&CR' (b = blank).

The contents of the arrays in the first two input records are as follows:

Record	Array	Array Contents
1	ARA	12345678901234567890
	ARC	01234567890123456789876N (note that N equals minus 5)
2	ARB	JOHNbDOEbJObSMITHb LEEbMARXbJIMbKNOTSb TIMbTYLERb
	ARC	(the same as record 1)

In the first output record, the location and contents of the arrays are as follows (b = blank):

Array	Location	Contents
ARA (first element)	85-89	12345
ARC	37-84	b1.23bb45.67bb 89.01bb23.45bb 67.89bb87.65bCR

For the second output record assume that the content of field X1 is 4; the locations and contents of the arrays are as follows:

Array	Location	Contents
ARB (fourth element)	91-100	JIMbKNOTSb
ARC	37-84	b1.23bb45.67bb 89.01bb23.45bb 67.89bb87.65bCR

Figure 211 (Part 2 of 2). Using Arrays to Format Field Output

The figure below shows a method of printing one array element per line on the printer output device.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E...FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++*
E
AR2 5 21 15 0
E*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
CLR DO 21 IN 30 DO 21 TIMES
CLR EXCPTTOTAL
CLR END
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OARFILE E 1 TOTAL
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
O AR2,IN 20
O*
```

Figure 212. Printing One Array Element per Line

Examples of Using Arrays

The following figure shows a method of printing more than one array element per line on the printer output device.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++Name++N/rN/tbLenPDSArnamLenPDSComments+++++++*
E                AR1      6 10 10
E                AR2      6 50 10
E*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C                DO  50      IN    20      DO THRU IN=50
C                MOVEAAR2,IN  AR1          MOVE TO ARRAY
C                EXCPTTOTAL          PRINT
C                END  10              ADD 10 TO IN
C*
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excnam.....*
OARFILE E 1                TOTAL
O.....N01N02N03Field+YBEnd+PConstant/editword+++++++*
O                AR1      B 100
O*
```

Figure 213. Printing More than One Element per Line

Appendix G. Glossary of Abbreviations

Abbreviation	Stands For	Definition
CL	Control Language	The set of all commands with which a user requests functions.
CPF	The system support licensed program for System/38.	It provides many functions that are fully integrated in the system such as work management, database, data management, job control, message handling, security, programming aids, and service. The equivalent function in the AS/400 system is the OS/400 system.
ICF	Intersystem Communications Function	A function of the OS/400 system that allows a program to interactively communicate with another program or system.
DBC	Double Byte Characters (DBC) Variables	Variables that can contain double-byte data (Japanese, Simplified Chinese, Traditional Chinese, and Korean ideograms).
DDS	Data Description Specifications	A description of the user's database for device files that is entered into the system using a fixed-form syntax. The description is then used to create files.
DDM	Distributed Data Management	A function of the operating system that allows an application program or user on a source system to access data files on remote systems connected by a communications network that also uses DDM.
GDDM	Graphical Data Display Manager	A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphics primitives. Contrast with <i>Presentation Graphics Routines</i> .
OS/400	N/A	The operating system for the AS/400 system. It provides many functions that are fully integrated in the system. These are work management, database, data management, job control, message handling, security, programming aids, and service.
PGR	Presentation Graphics Routines	A group of routines that allows business charts to be defined and displayed procedurally through function routines. Contrast with <i>Graphical Data Display Manager</i> .
SNA	Systems Network Architecture	The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of Systems Network Architecture networks. Note: The layered structure of SNA allows the ultimate origins and destinations of information (that is, the end users) to be independent of, and unaffected by, the specific SNA network services and facilities used for information exchange.
SQL	Structured Query Language	A language that can be used within programs written in other languages, or interactively to access database manager data and to control access to database manager resources.

Glossary of Abbreviations

Bibliography

Publications Guide, GC41-9678, which contains a brief description of each manual in the AS/400 library and information on how to order additional publications.

Data Management Guide, SC41-9658, which contains information about managing key aspects of the system.

Data Description Specifications Reference, SC41-9620, which describes data description specifications that are used for describing files.

Distributed Data Management Guide, SC41-9600, which contains information about remote communication for the RPG/400 programmer.

Database Guide, SC41-9659, which contains a detailed discussion of the AS/400 database structure. This manual also describes how to use data description specifications (DDS) keywords.

Communications: Intersystem Communications Function Programmer's Guide, SC41-9590, which provides information an application programmer needs to write applications that use AS/400 communications and the Intersystem Communications Function file.

Programming: GDDM Programming Guide SC41-0536, and *Programming: GDDM Programming Reference,*

SC41-0537, which provide guidance on the Graphical Data Display Manager (GDDM) for programmers who need to write graphics applications.

Programming: System/38 Environment Programmer's Guide and Reference, SC41-9755, which describes migrating from System/38 and converting to an AS/400 system.

Software Installation, SC41-3120, which describes how to install the RPG/400 licensed program on your system.

System Operation, SC41-3203, which describes how to operate the AS/400 System.

Systems Application Architecture Structured Query Language/400 Reference, SC41-9608*, which describes SQL on the AS/400 system.

Languages: RPG Reference Summary, SX09-1164, which contains a summary of operation codes, edit codes, indicators, and status codes.

RPG Debugging Template, GX21-9129, which provides a template of the RPG specifications.

RPG/400 Reference, SC09-1817, which provides a reference for the RPG/400 compiler.

Index

Special Characters

/END EXEC delimiter 4
/EXEC SQL delimiter 4
* (asterisk)
 generated specifications 271, 284
** (double asterisk)
 generated specifications 271, 284
*** (triple asterisk)
 generated specifications 271, 284
*ALL value 41
*ATR, GENOPT parameter 498
*CHANGE value 41
*CTLSPEC value 32
*CURLIB value 32, 33, 39
*CURRENT value 40
*DATETIME value 38
*DUMP value 35
*ENTRY PLIST 259
*EXCLUDE value 41
*FLAG value 39
*GEN value 35
*GRAPHIC value 38
*LIBCRTAUT value 41
*LIBL value 33, 39
*LSTDBG value 36
*NODUMP value 35
*NOFLAG value 39
*NOGEN value 35
*NOLSTDBG value 36
*NONE value 37, 38
*NOOPTIMIZE value 36, 499
*NOSECLVL value 35
*NOSOURCE value 34
*NOSRCDBG value 35
*OPTIMIZE value 36
*OPTIMIZE, GENOPT parameter 499
*OWNER value 40
*PATCH, GENOPT parameter 498
*PGM value 33
*PRV 40
*SECLVL value 35
*SOURCE 34, 498
*SRCDBG value 35
*SRCMBRTXT value 34
*USE value 41
*USER value 40
*VARCHAR value 38
& (ampersand)
 auto-report copy function 277

Numerics

01-99 indicators
 See field and field record relation indicators
 See indicators conditioning calculations and output
 See resulting indicators

A

A\$\$SUM subroutine (auto report) 285
abbreviations of terms 547
access path
 example of 120
 for externally described DISK file iii, 114
 for indexed file 119
ALL value 41
allow null values parameter 44
alternate collating sequence
ALWNULL parameter 44
ampersand (&)
 auto-report copy function 277
AND relationship
application design 15
area parameter for SPECIAL PLIST 102
array
 alternating
 binary format 217
 compile-time
 examples of using iii, 537—546
 fixed indexes, using 539
 formatting field output 543
 initialization 241
 packed format 215
 prerun-time arrays
 printing elements of 545, 546
 run-time
 See run-time array
arrival sequence access path 114
ascending sequence
asterisk (*)
 generated specifications 271, 284
 indication on auto report total lines 280
AUT parameter 41
authority parameter 41
authorization-list name value 41
*AUTO output specifications
 See *also* auto report function
 generated end positions 281
 generated RPG/400 specifications 284
 group printing 271
 report format 280
 spacing and skipping 280

- auto report function
 - See also* AUTO output
 - /COPY 276
 - /COPY statement
 - format 276
 - general description 275
 - modifying copied specifications iii, 276, 277
 - A\$\$SUM subroutine 285
 - AUTO 282
 - column headings and fields, placement 281
 - compilation of a source program 25
 - CRTRPTPGM command 294
 - definition 271
 - examples 299—320
 - format 280
 - generated specifications
 - calculation 285
 - output 285
 - order of specifications included with /COPY 275
 - page headings 281
 - phases 518
 - placement of /COPY statement in program 275
 - print lines, overflow 283
 - programming aids 291
 - report body 283
 - service information 493
 - sorting of specifications with /COPY 275
 - spacing and skipping 280
 - using 271

B

- bibliography 549
- binary field 217
 - data structure subfield specifications 221
 - definition 217
- binary format
 - definition 217
 - input field specification 217
 - output field specification 217
- Binary Synchronous Communication 527
- blank after
- blocking/unblocking records 82
- body of a report 283
- breakpoints iii, 54, 57
- BSC
 - See* Binary Synchronous Communication

C

- calculation specifications 22
 - general description 22
 - program-described WORKSTN file 155
- calculations generated by auto report iii, 285
- CALL (call a program) operation code
 - a subroutine 261

- CALL (call a program) operation code (*continued*)
 - DSPPGMREF 258
 - GDDM 260
 - message-retrieving subroutine (SUBR23R3) 261
 - moving double byte data and adding control characters (SUBR41R3) 264
 - moving double byte data and deleting control characters (SUBR40R3) 263
 - other programs 258
 - program communication 258
 - query names of called programs 258
- CHANGE value 41
- character
- CL programs
 - See* control language (CL) program
- clear command 147
- CLEAR operation code 240
- CODE/400
 - compiler options 35
- CODELIST parameter 42, 499
- coding form
 - See* calculation specifications
 - See* control specifications
 - See* data description specifications
 - See* extension specifications
 - See* file description specifications
 - See* input specifications
 - See* line counter specifications
 - See* output specifications
- collating sequence
 - See* alternate collating sequence
- column headings in auto report 281, 283
- combined file 156
- COMIT (commit) operation code
 - commitment control 108
 - system considerations 108
 - with multiple devices 108
- command attention (CA) keys 143
- command function (CF) keys 143
- comments
 - /COPY statement 275
- commitment control 107
- common area
 - VCOMMON 496
 - XREGN 518
- Common Programming Interface 263
- communication
 - accessing other programs and systems 143
 - between objects 255
 - data 527
 - error recovery 527
 - exception/error handling 527
 - with objects in system 255
- compile time array or table
 - See also* array
 - initialization 241

- compiler
 - data areas 496
 - debugging options 498, 500
 - differences between RPG/400 and the System/38 Environment Option of the RPG Compiler 531
 - differences between System/38 RPG III and System/38 Environment Option of the RPG Compiler 531
 - directives
 - error message organization 496
 - file types, supported 535
 - phase descriptions 493
 - phases 494, 495, 518
 - service information 493
 - System/38 environment option 531
- compiling
 - auto report source program 294
 - CRTRPGPGM 25
 - CRTRPTPGM 294
 - RPG source program 25
- composite key operation codes
 - See search argument
- conditional branching 6
- conditioning files
 - See external indicators
- conditioning output
 - overflow indicators 92
- consecutive processing 124
- constants
 - See named constants
- constants, figurative
 - See *ALL'X..',
 - See *BLANK/*BLANKS
 - See *HIVAL
 - See *LOVAL
 - See *ZERO/*ZEROS
- control break
- control field
- control language (CL) program
 - calling 255
 - commands used with RPG/400 2
 - commonly used commands 2
 - definition 547
- control level (L1-L9) indicators
- control specifications 21
 - general description 21
- control-record format, subfile 150
- controlled loop
 - do until operation 13
 - do while operation 11
 - general description 9
- CoOperative Development Environment/400
 - compiler options 35
- /COPY statement
 - See auto report function

- copy function, auto report
 - See auto report function
- CPI support 263
- Create Auto Report Program (CRTRPTPGM)
 - command 294, 295
- Create RPG/400 Program (CRTRPGPGM)
 - command 25, 498
- cross-reference listing
 - CRTRPTPGM 297
 - how to request for RPG source program 35
- CRTRPTPGM command 295
- CTLSPEC value 32
- CURLIB value 32, 33, 39
- CURRENT value 40
- CVTOPT parameter 38

D

- D-*AUTO
 - See also AUTO output
 - overflow of print lines 283
- data area data structure
 - general information 223
 - initialization 242
 - statement
 - externally described 220
 - program described 220
 - rules 222
 - specifications 222
 - subfields
 - rules 226
 - specifications 224
 - used to access data area 267
- data areas
 - compiler 496
 - general information 267
 - how to access 267
 - printing 498
 - restrictions 267
 - retrieval
 - implicit 223
 - RETURNCODE 26
 - unlocking
 - implicit 223
 - writing
 - implicit 223
- data communication 527
- data description specifications (DDS)
 - See the DDS Reference
- data field formats 215
- data structure statement
 - externally described 220
 - program-described 220
 - rules 223
 - rules for specifying 222
 - specifications 221, 222

- data structure subfields
 - format 221
 - overlapping 226
 - rules 226
 - specifications 224
- data structures 215
 - data area 223
 - examples of 226
 - examples of initialization 243
 - externally described 236
 - file information 224
 - format 221
 - general information 220
 - initialization 241
 - length of 222
 - multiple occurrences of 222, 223
 - program-status 224
 - special 223
 - special considerations when initializing 241
 - special types
 - data area 223
 - file information (INFDS) 223
 - program status 223
 - statement
 - See data structure statement
 - subfields
 - overlapping 226
- database data
 - date fields 250
 - DBCS-graphic fields 251
 - null values 252
 - time fields 250
 - timestamp fields 250
 - variable-length fields 247
- date data fields (SAA data type) 250
- DATETIME value 38
- DBCS-graphic data type 251
- DDM (distributed data management) files 529
- DDS
 - See the DDS Reference
- debugging
 - examples 500
 - general discussion 47
 - using DEBUG 60
 - using DUMP 60
- descending sequence
- designing your program
 - See *also* application design
 - overview 5
 - structured programming 6
- detail lines
 - See AUTO output
- detail printing, auto report
 - See AUTO output
- device files
 - device dependence 75
 - device files (*continued*)
 - device independence 75
 - DISK files 113
 - multiple-device 157
 - PRINTER files 90
 - SEQ files 99
 - SPECIAL files 101
 - WORKSTN files 143
 - device name, function of 75
 - DISK file
 - externally described
 - access path 114
 - as program-described 123
 - examples 114
 - general description 113
 - processing 118
 - record format specifications 113
 - file operation codes allowed
 - for keyed processing methods 141
 - for non-keyed processing methods 141
 - general description 113
 - processing charts
 - processing methods
 - consecutive processing 124
 - overview 123
 - random-by-key processing 133
 - relative-record-number processing 123
 - sequential-by-key processing 124
 - sequential-within-limits processing 132
 - program described
 - indexed file 119
 - processing 123
 - record-address file 122
 - sequential file 122
 - record-format specifications 113
 - displaying messages 47
 - distributed data management (DDM) 529, 549
 - DO operation code
 - flowchart and summary 9
 - double asterisk (**)
 - generated specifications 271, 284
 - double byte character set
 - moving 263, 264
 - DOUxx (do until) operation code
 - and structured programming 13
 - DOWxx (do while) operation code
 - and structured programming 11
 - DS
 - See data structures
 - DSPPGMREF 258, 522
 - *DUMP
 - GENOPT parameter 498
 - GENOPT parameter value 36
 - OPTION parameter value 36, 498
 - DUMP value 35

dump, formatted 60
duplicate field names on /COPY modifier
statement 279

E

edit codes
edit source (STRSEU) command 23
edit word
encapsulated program 493
ending a program 265, 266
ENDSR (end subroutine) operation code
environment
 System/38 3, 531
error handling 253
 See also exception/error handling
error messages
 organization 496
 using, displaying, and printing 47
error parameter for SPECIAL PLIST 101
exception/error handling 70
 communications files 527
 device communications 527
 RPG default 70
 with ICF files 527
EXCLUDE value 41
extension specifications 22
 general description 22
external (U1-U8) indicators
externally described data structure 236
externally described file
 access path 114
 adding to external description 84
 advantages 78
 as program-described 123
 as WORKSTN file 143, 144
 data format 215
 definition 78
 file description specifications for 84
 output specifications for 88
 overriding 87
 processing methods 118
 record format specifications 113
 renaming record format 85
 specifications 84

F

fetch overflow
 See also overflow (OA-OG, OV) indicators
 general description 94
 logic 95
field
 binary 217
 format in file 215
 in auto report 281

field (*continued*)
 packed 215
field name
 table
 FLDTBL 518
 XFDTAB 496
field-reference file, example of 115
file
 array
 See array
 database
 See DISK file
 deleting records from
 device dependence 75
 device independence 75
 DISK
 See DISK file
 externally described 78
 See also externally described file
 externally described disk 113
 field format 215
 general considerations 75
 indexed 119
 locking 81
 name
 externally described 78
 override 87
 program-described 90
 open options 83
 override 87
 PRINTER 90
 processing charts
 PRINTER file 98
 sequential file 100
 SPECIAL file 104
 program-described 78, 90
 redirection 76
 SEQ 99, 122
 sharing 83
 SPECIAL 101
 types
 supported by RPG/400 535
 supported by System/38 environment RPG com-
 piler 535
 valid keys 117
 WORKSTN 143
file condition
 See external indicators, UC
file considerations
 general 75
file description specifications 22
 CFILE continuation line option 143
 commitment control 108
 for externally described files 85
 general description 22
 modifying 277

- file description specifications (*continued*)
 - using the copy function with automatic report 276
- file information data structure
 - general information 224
 - statement
 - externally described 220
 - program-described 220
 - rules 222
 - specifications 221
 - subfields
 - rules 226
 - specifications 223
- file locking 81
- file name table, XFLTAB 495
- file operation codes
 - allowed with DISK file 141
 - allowed with PRINTER file 90, 98
 - allowed with sequential file 100
 - allowed with SPECIAL file 103, 104
 - allowed with WORKSTN file 147, 150
- file sharing 83
- FLAG value 39
- floating minus edit codes, specifying 523, 532
- floating-point fields 86
- flowchart
 - fetch-overflow logic 95
- format
 - external processing 220
 - internal processing 220
 - signs 219
- format name 154
- format of data
 - binary 217
 - data in fields 215
 - packed-decimal 215
 - zoned-decimal 216
- formatted dump, using 60
- formatting report (auto report) 280
- FREE (deactivate a program) operation code
 - program communication 261
- function keys
 - indicators 146
 - with WORKSTN file 146

G

- GDDM 260
- GEN value 35
- generated RPG/400 program, auto report
 - calculations 285
 - examples 299
 - group printing 271
 - output specifications 285
 - reformatting *AUTO page headings 281
 - source of specifications 275
 - subroutine (A\$\$SUM) 285

- generated specifications, auto report 271, 284
- generation of program
 - See compiling
- generation options parameter 36
- generation severity level parameter 33
- GENLVL parameter 33
- GENOPT parameter 36
 - *ATR 498
 - *DUMP 498
 - *LIST 498
 - *OPTIMIZE 499
 - *PATCH 498
 - *XREF 498
- Grant Object Authority command 41
- graphic data type 251
- GRAPHIC value 38
- Graphical Data Display Manager 260
- graphics 260
- graphics support 260
- group printing
 - examples 271
 - specifications 271
- GRTOBJAUT command 41

H

- H1-H9
 - See halt (H1-H9) indicators
- halt (H1-H9) indicators
 - used to end a program 266
- handling, exception/error
 - See exception/error handling
- header specifications
 - See control specifications
- help command key 147
- home command key 147

I

- ICF communications file 143
- ideographic (IGC) variables
 - Double Byte Characters (DBC) variables 265
 - subroutines SUBR40R3 263
 - subroutines SUBR41R3 264
- If-Then-Else structure 6
- IGC
 - See ideographic (IGC) variables
- IGNDECERR parameter 44
- ignore decimal data error parameter 44
- IGNORE option 86
- ignoring record format 86
- INDENT parameter 37
- indexed file
 - access path 119
 - general description 119
 - valid search arguments 119

- indicators
 - control level (L1-L9)
 - external (U1-U8)
 - function key (KA-KN, KP-KY)
 - with WORKSTN file 146
 - halt (H1-H9)
 - used to end a program 266
 - last record (LR)
 - used to end a program 266
 - overflow
 - examples 93, 95
 - fetch overflow logic 94
 - general description 90
 - presence or absence of 92
 - relation to program cycle 95
 - setting of 95
 - with PRINTER file 90
 - return (RT)
 - used to end a program 266
 - table, XINTAB 496
- INFDS
 - See file information data structure
- information
- initialization
 - of data structure subfields 242
 - of data structures 241
 - of data structures, examples 243
 - overview 240
 - special considerations for data structures 241
- initialization subroutine (*INZSR)
 - initialization overview 240
- input
 - designing 6
 - file 156
- input field specifications
 - binary format 217
 - modifying 277
- input record
 - unblocking 82
- input specifications 22
 - general description 22
 - program-described WORKSTN file 155
- intermediate text
 - description 493
 - listing 493
- intermediate text dump parameter 42
- interprogram communication 255
- Intersystem Communications Function (ICF) iii, 143
- IRP listing, layout 36, 515
- ITDUMP parameter 42, 499

K

- KA-KN KP-KY indicators
 - See function key indicators

- key
 - composite 118
 - for a record or a file 117
 - partial 118
- keyed processing
 - access path 114
 - examples 133
 - indexed file 119
 - record-address limits file 122
 - sequential 123
 - sequential-within-limits 132
- keyed-sequence access path 114
- keywords
 - DDS 113
 - for continuation line 113
 - CLEAR 147
 - HELP 147
 - HOME 147
 - PRINT 147
 - ROLLDOWN 147
 - ROLLUP 147
 - for display device file
 - CLEAR 147
 - HELP 147
 - HOME 147
 - PRINT 147
 - ROLLDOWN 147
 - ROLLUP 147

L

- L1-L9 (control level) indicators
 - See control level (L1-L9) indicators
- last record (LR) indicator
 - used to end a program 266
- level checking 80
- LIBCRTAUT value 41
- LIBL value 33, 39
- limits records 114
- line counter specifications 22
 - general description 22
- linkage to other programs
 - CALL/FREE/RETRN 255
 - SPECIAL device 101
- *asterisk.LIST
 - GENOPT parameter value 498
- listing, sample of source and cross-reference 50
- locking
 - file 81
 - read without locking 81
 - record locking wait time 81
 - retry on timeout 81
 - standalone 81
 - under commitment control 109
 - UNLCK 81

logical file
 See DISK file
LR (last record) indicator
 See last record (LR) indicator
LSTDBG value 36

M

merging of members 522
message retrieval 265
message-retrieving subroutine (SUBR23R3) 261
messages
 displaying 47, 49
 printing 47, 49
 SAA 49
 using 47
modifier statements (/COPY function)
 file description specifications 277
 input-field specifications 277
monitor message (MONMSG) command 26
moving double-byte data and adding control characters
 (SUBR41R3) 264
moving double-byte data and deleting control characters
 (SUBR40R3) 263
multiple devices attached to application program 108
multiple-device file 157

N

named constants 237
 rules 237
 specifying 237, 523
NODUMP value 35
NOFLAG value 39
NOGEN value 35
NOLSTDBG value 36
NONE value 37, 38
nonkeyed processing 140
NOOPTIMIZE value 36
NOSECLVL value 35
NOSOURCE value 34
NOSRCDBG value 35
NOXREF value 35
null value support 252
numeric fields
 auto report
 centering column headings 281
 format 215, 216

O

OA-OG, OV (overflow) indicators
 See overflow (OA-OG, OV) indicators
objects
 communication 255

online information
open data path 83
 sharing 83
operation codes 147, 150
 allowed with DISK file 141
 allowed with PRINTER file 90
 allowed with sequential file 99
 allowed with SPECIAL file 104
OPTIMIZE value 36
OPTION parameter 34, 498
 *DUMP 498
 *SOURCE 498
 *XREF 498
option parameter for SPECIAL PLIST 101
output
 designing 5
 generated 285
 specifications 88
output field specifications
 binary format 217
output file 156
output record
 blocking 82
output specifications
 general description 23
 program-described WORKSTN file 154
output spooling 77
overflow
 D/T-*AUTO print lines 283
 indicators 92
 page 91
overflow indicators
 conditioning output 92
 examples 93, 95
 fetch-overflow logic 94
 general description 92
 presence or absence of 92
 relation to program cycle 95
 setting of 95
 with PRINTER file 90
overlapping subfields 226
override, file 87, 140
overriding external description 87
OWNER value 40

P

packed decimal format
 array/table field 215
 definition 215
 description 215
 input field 215
 length in bytes 216
 length in digits 216
 output field 215

- page headings
 - AUTO 282
 - in auto report 281
- page number, in PRINTER file 91
- page overflow, in PRINTER file 91
- parameter list
 - See also* PARM (identify parameters) operation code
 - created by PARM 259
 - created by SPECIAL 101
 - identifying 259
 - rules for specifying
- PARM (identify parameters) operation code
 - program communication 259
 - rules for specifying 260
- partial key 118
 - processing methods
 - externally described DISK files 118
- PDA (PIP data areas)
 - description 269
 - initialization 242
- PGM parameter 32
- PGM value 33
- PGR (Presentation Graphics Routines) 260
- phase trace 499
- phase trace parameter 41
- phase-name value 42
- phases
 - auto report program 518
 - compiler 494
 - trace 499
- PHSTRC parameter 41, 499
- physical file
- PIP (Program Initialization Parameters) data area
 - and data areas (PDA) 269
 - initialization 242
- PLIST (identify a parameter list) operation code
 - program communication 259
 - rules for specifying 259
- PLIST keyword for SPECIAL file
 - description of parameters 101
- prerun-time array or table
 - description of parameters 101
 - initialization 241
- Presentation Graphics Routines 260
- prestart jobs 269
- preventing printing over perforation 94
- print command key 147
- print file parameter 39
- print lines in an auto report 283
- print messages 47
- printer control option
 - See* PRTCTL
- PRINTER file
 - access current line value 96
 - fetch-overflow logic 94
 - file operation codes allowed 90
- PRINTER file (*continued*)
 - maximum number of files allowed in program 90
 - modify forms control 96
 - overflow indicators 90
 - page overflow 91
 - processing chart 98
 - PRTCTL (printer control) 96
- processing methods
 - consecutive 124
 - for DISK file 118, 123
 - for externally described file 118
 - keyed
 - See* keyed processing
 - nonkeyed 140
 - relative-record-number 123
 - sequential only 124, 140
 - sequential-by-key 124
 - sequential-within-limits 132
- program
 - abnormal ending 266
 - data areas 267
 - deactivating 261
 - entering 23
 - normal ending 266
 - return without an end 267
 - samples 321
 - template 493
- program communication
 - CALL/FREE/RETRN 255
 - data 527
 - returning from a called program 265
 - SPECIAL 101
- program cycle
 - commitment control 109
 - fetch overflow logic 95
 - initialization 243
- program design
 - applications 15
 - RPG 5
- program ending 265
- program exception/error subroutine
 - example 74
- program identification
 - See* program name
- program initialization parameters
 - and data areas (PDA) 269
- Program Initialization Parameters data area
 - initialization 242
- program name
 - *PGM parameter 33
- program parameter 32
- program status data structure
 - general information 223
 - multiple occurrences of 223
- statement
 - externally described 220
 - program-described 220

- program status data structure (*continued*)
 - statement (*continued*)
 - rules 222
 - specifications 221
 - subfields
 - rules 226
 - specifications 224
- program-described file 78
 - as DISK file 119
 - as WORKSTN file 154, 156
 - data format 215
 - definition 78
 - valid search arguments 119
- programming aids 291
- prompt screens
 - CRTTRPGPGM command 29
- protecting records/files
 - See file locking by RPG
- PRTCTL (printer control)
 - example 98
 - general information 96
- PRTFILE parameter 39
- PRV value 40
- *PSSR
 - See program exception/error subroutine
- public authority for an object 41

Q

- QRPGSRC value 33
- QSYSPRT value 39
- query names of called programs 258

R

- random by key processing iii, 132
- RCLRSC command 84
- READ (read a record) operation code
 - with data communication 527
- READC (read next modified record) operation code
 - with WORKSTN subfile 152
- Reclaim Resources (RCLRSC) command 84
- RECNO
 - with relative-record-number processing 123
- record
 - limits 122
 - locking 81
 - name table, XRCTAB 496
 - releasing 82
 - valid keys 117
- record address file
 - relative-record number 122
 - sequential-within-limits 122
 - with limits records 122
 - with relative record numbers 122

- record address limits file
 - See record address file
- record address relative record number file
 - See record address file
- record format
 - for a subfile 150
 - ignoring 86
 - renaming 85
 - specifications for externally described file 113
- record locking 81
- record sharing
 - See file locking by RPG
- redirection, file
 - definition 76
 - general description 76
- relative record number record address file
 - See record address file
- relative-record-number 123
- relative-record-number processing 123
- release-level parameter 40
- releasing a locked record 82
- RENAME option 85
- renaming record-format names 85
- RESET operation code 240
- retrieval of data area
 - implicit 223
- RETRN (return to caller) operation code
 - program communication 265
- retry on a record lock timeout 82
- return (RT) indicator
 - used to end a program 266
- return status parameter 102
- RETURNCODE data area 26
- returning from a called program iii, 265
- Revoke Object Authority command 41
- rolldown command key 147
- rollup command key 147
- RPG II S/36-Compatible and RPG/400 521
- RPGOBJ 32
- RT (return) indicator
 - See return (RT) indicator
- run-time array
 - See array
- run-time subroutines 497
- RVKOBJAUT command 41

S

- SAA data types
 - date, time and timestamps 250
 - DBCS-graphic data type 251
 - null value support 252
 - overview 247
 - variable-length fields 247
- SAA flagging parameter 39

SAA messages, flagging 49
SAA support 263
sample programs 321
 checklist 321
 control file maintenance 383
 data descriptions 384
 data area control file 326, 336
 data area control file maintenance 326
 data entry 166
 data maintenance 174
 database design 324
 database field definition 332
 database reference master file 333
 employee master file 324, 337
 format name on output specifications 206
 inquiry 159
 inquiry by zip code and search on name 196
 master file maintenance 326, 346, 347, 382
 data descriptions 354
 employee master maintenance 349
 employee master selection 348
 project master maintenance 351
 project master selection 350
 reason code master maintenance 353
 reason code master selection 352
 select format 347
 master file maintenance RPG/400 program
 PRG01 364
 PRG02 386
 monthly processing 444
 employee summary report data descriptions 449
 master file monthly update and clear RPG/400
 program 487
 monthly time file update and reporting 444
 project summary report data descriptions 464
 project summary report RPG/400 program 468
 reason code summary report data
 descriptions 476
 reason code summary report RPG/400
 program 479
 time reporting employee summary report 453
 time reporting employee summary report
 layout 448
 time reporting project summary report layout 463
 time reporting reason code summary report
 layout 475
 monthly time-entry file reporting and update
 process 330
 monthly transaction entry file 341
 project master file 324, 338
 read operation with time-out 211
 reason-code master file 325, 339
 subfile processing 187
 time file transaction entry 394
 time reporting menu design 343
 time reporting transaction entry 395
 data descriptions 397
 sample programs (*continued*)
 time reporting transaction entry (*continued*)
 employee selection display 395
 time reporting transaction entry RPG/400 program
 PRG03 401
 time-file entry process 327
 transaction history files 325
 variable start line 208
 weekly time file update 418
 master file update 428
 time file entry edit 421
 weekly employee transaction report layout 427
 weekly transaction report 428
 weekly time-file update process 328
 weekly transaction entry file 340
 year end processing 491
save-while-active 50
screen design aid (SDA) 4
SDA
 See screen design aid (SDA)
search argument
 externally described file
 description 117
 referencing a partial key 118
 valid 117
 program-described file 119
SECLVL value 35
SEQ file
 example 99
 file operation codes allowed 100
 general description 99
 processing chart 100
 restrictions 99
 variable-length 99
sequence
 sequence, collating
 See alternate collating sequence
sequence checking
 on input specifications 90
sequential file 122
sequential operation 6
sequential-by-key processing iii, 124
sequential-only processing 124
sequential-within-limits processing iii, 132, 133
service information
 auto report function 493
 compiler 493
SEU
 See source entry utility (SEU)
severity-level-value 34
sharing an open data path for a file 83
signs
 external format 219
 internal format 219
snap dump parameter 42

- SNPDUMP parameter 42, 499
- source and cross-reference listing example 50
- source entry utility (SEU) iii, 3, 23
- source file parameter 32
- source listing indentation parameter 37
- source listing options parameter 34
- source member parameter 33
- source program
 - compiling
 - in System/38 environment 45
 - using CRTRPGPGM command 26, 27
 - entering into system 23
 - listing 498
 - running 49
- SOURCE value 34
- special command keys 147
- SPECIAL file
 - definition 101
 - deleting records from 104
 - device name 101
 - general description 101
 - parameter list 101
 - valid file operations 104
- specifications
 - /COPY statement 275
 - copied, modifying 276
 - data structure statement 221
 - data structure subfields 224
 - data structure subfields, rules for 226
 - entering and coding 21
 - externally described file 84
 - file description 85
 - file description, modifying 277
 - forms 21
 - generated 284
 - group printing 271
 - input-field, modifying 277
 - output 88
 - output, generated by auto report 285
 - record format 113
- spooling 77
- SRCDBG value 35
- SRCFILE parameter 32
- SRCMBR parameter 33
- SRCMBRTXT value 34
- starting commitment control 107
 - ending commitment control 107
- status parameter for SPECIAL PLIST 101
- STRSEU (edit source) command 23
- Structured Query Language (SQL) 4
- subfields
 - for data structure subfield specifications 224
 - for file information data structure 61
 - for PRTCTL 97
 - rules for 226
 - rules for initializing 242
- subfields (*continued*)
 - within a data structure 224
 - description 224
 - specifications for 224
- subfile
 - control-record format 150
 - descriptions 150
 - examples 152, 153
 - file operation codes allowed with 151
 - general description 150, 151
 - record format 150
 - uses of 152
- subfiles
 - SUBR23R3 (message retrieval) 261
 - SUBR40R3 (manipulating Double Byte Characters variables) 263
 - SUBR41R3 (manipulating Double Byte Characters variables) 264
- subroutines
 - A\$\$SUM (auto report) 285
 - calling special subroutines 261
 - run-time 497
 - SUBR23R3 (message retrieval) 261
 - SUBR40R3 (manipulating DBC variables) 263
 - SUBR41R3 (manipulating DBC variables) 264
- summary tables
 - file operation codes allowed with
 - DISK 140
 - PRINTER 90, 98
 - sequential 100
 - SPECIAL 103, 104
 - WORKSTN 147, 150
 - PRINTER file processing 98
 - sequential file processing 100
 - SPECIAL file processing 104
 - WORKSTN file processing 150
- syntax
 - CRTRPGPGM command 29
 - CRTRPTPGM command 295
- system functions
 - spooling 77
- System/36-Compatible RPG II and RPG/400 521
- System/38 environment
 - differences between RPG/400 and the System/38 Environment Option of the RPG Compiler 531
 - differences between System/38 RPG III and System/38 Environment Option of the RPG Compiler 531
 - environment on the AS/400 system 3
 - RPG/400 compiler 531
 - source program compiling 45

T

- T-*AUTO
 - See AUTO output

- table
 - See array
 - searching
 - See LOKUP operation
- tape file 122
- target release parameter 40
- techniques for efficient coding 6
- template, program 493
- test library, using iii, 51, 54
- testing
 - breakpoint 54
 - fields
 - See field indicators
 - test library 51
 - trace 58
- TEXT parameter 34
- TGTRLS parameter 40
- time data fields (SAA data type) 250
- timestamp data fields (SAA data type) 250
- trace, using iii, 58
- triple asterisk (***)
 - generated specifications 271, 284
- type conversion options parameter 38

U

- U1-U8
 - See external (U1-U8) indicators
- unblocking/blocking records 82
- usage of indicators
 - See indicators
- USE value 41
- user profile parameter 40
- USER value 40
- USRPRF parameter 40
- utilities
 - screen design aid (SDA) 4
 - source entry utility (SEU) 3

V

- valid file operations
 - SPECIAL file 104
 - WORKSTN file
- valid keys
 - for file 117
 - for records 117
- VARCHAR value 38
- variable-length fields 247
- variable-length records 99
- VCOMMON 496

W

- WORKSTN file
 - definition 143

- WORKSTN file (*continued*)
 - examples 158
 - externally described 143
 - processing 144
 - file operation codes allowed with 147
 - function key indicators with 146
 - multiple-device 157
 - processing 147
 - program-described 154
 - calculation specifications 155
 - combined file 156
 - considerations 156
 - input file 156
 - input specifications 155
 - output file 156
 - output specifications 154
 - with format name 154
 - without format name 156
 - sample data entry program 166
 - sample data maintenance program 174
 - sample format name program 206
 - sample inquiry and search program 196
 - sample inquiry program 159
 - sample read with time-out program 211
 - sample subfile processing program 187
 - sample variable start line program 208
 - subfiles
 - control-record format 150
 - examples 152
 - for display-device file 150
 - record format 150
 - uses of 152
 - using 143
- WRITE (create new records) operation code
 - with data communication 527

X

- XFDTAB 496
- XFLTAB 496
- XINTAB 496
- XRCTAB 496
- *XREF
 - GENOPT parameter 498
 - GENOPT parameter value 35
 - OPTION parameter value 35
- XREF value 35

Z

- zoned decimal format
 - definition 216
 - description 216



Program Number: 5763-RG1



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-1816-00

