Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xxiii.
Contents

Notices .......................................................... xxiii
About This Manual ............................................. xxv
Who Should Use This Manual ................................. xxv
What You Should Know ........................................ xxv
 Naming Conventions ........................................... xxvi
 Coding Conventions ........................................... xxvii
 How To Use the RPG II Coding Examples .................. xxvii
 Index Search .................................................... xxviii
 Help for Control Language Commands ....................... xxix

Chapter 1. An Introduction to RPG II and the AS/400 System ...... 1
 OS/400 and the Control Language ............................ 1
   The AS/400 Operating System ................................ 1
   The AS/400 Control Language ................................ 1
 Accessing RPG II from OS/400 ............................... 2
 Your Choice of Two Environments: AS/400 System or the System/36 Environment ........................................ 3
   The System/36 Environment .................................. 3
   Using the System/36 RPG II Procedures ..................... 4
 Files Used by RPG II .......................................... 5
 RPG Program Cycle ........................................... 6
 Designing Your RPG II Program ................................ 11
   Steps in Designing Your Program ............................ 13
   Designing the Output ....................................... 13
   Designing the Processing ................................... 13
   Designing the Input ........................................ 14
 Structured Programming in RPG II ............................ 14
   Sequential Operation ...................................... 14
   Conditional Branching ..................................... 14
 Repeating an Operation ....................................... 16
   Do While .................................................. 16
   Do Until .................................................. 18
   Do ......................................................... 20
 Summary of Structured Programming Operation Codes ............ 22
 The RPG Screen Design Aid Utility (RPGSDA) ................. 23

Chapter 2. Entering RPG II Specifications .......................... 25
 The RPG II Specifications ..................................... 25
   Control Specification ...................................... 26
   File Description Specifications ............................ 27
   Extension Specifications ................................... 27
   Line Counter Specifications ............................... 27
   Telecommunications Specifications ........................ 27
   Input Specifications ....................................... 28
   Calculation Specifications ................................ 28
   Output Specifications ...................................... 28
 Entering Your Program into the System ......................... 29
 Using SEU to Enter or Change an RPG II Source Program ...... 29
 The RPG II Syntax Checker ................................... 30
Using the RPGSEU Display .................................................. 30

Chapter 3. Compiling an RPG II Program .................................. 33
Using CRTS36RPG to Compile an RPG II Source Program in the AS/400 system .................................................... 33
Using the CRTS36RPG Command ............................................. 34
  Elements of the CRTS36RPG Command Line ....................... 35
  Entering Only Certain Parameter Values .......................... 35
  Entering Only the Parameter Values ............................... 35
Completing the First CRTS36RPG Display .............................. 36
Completing the Second CRTS36RPG Display ......................... 41
Using RPGC to Compile an RPG II Source Program in the System/36 Environment ...................................................... 45
Calling RPGC ................................................................. 45
Using the First RPGC Display .............................................. 45
Using the Second RPGC Display ........................................... 47
Using the SEU Split Display to Correct Errors ....................... 48
Printing an RPG Cross-Reference Listing (RPGX Procedure) in System/36 Environment ...................................................... 49
Cross-Reference Listing ..................................................... 50
  Sample Cross-Reference Listing .................................. 51
Interpreting a Compiler Listing ........................................... 52

Chapter 4. Testing and Debugging an RPG Program ..................... 61
Using, Displaying, and Printing Messages .................................. 61
  Using Messages .......................................................... 61
  Displaying and Printing Messages ................................ 63
Running an RPG Program ..................................................... 63
  Example of OCL Statements Used to Run a Program ............. 63
  RPG Halt Messages ..................................................... 63
Using Breakpoints ............................................................ 64
  Example of Using Breakpoints .................................. 64
  Considerations for Using Breakpoints ......................... 68
Using a Trace ................................................................. 68
  Example of Using a Trace ........................................ 69
  Considerations When Using a Trace .............................. 69
Using an RPG Formatted Dump ............................................ 70
Using the DEBUG Operation Code ...................................... 76
  Records Written by the DEBUG Operation ......................... 77
Debugging a Program That Uses a WORKSTN File .................... 77

Chapter 5. The Detailed RPG II Program Cycle .......................... 81

Chapter 6. Using a DISK File ................................................ 89
SEQUENTIAL FILES .......................................................... 89
Creating a Sequential File .................................................. 90
  Example of Creating a Sequential File .......................... 91
Reading a Sequential File .................................................. 92
  Reading Consecutively ............................................. 93
  Reading Randomly by Relative Record Number .................. 94
  Reading Randomly by Relative Record Number and/or Consecutively ........................................ 95
  Reading Randomly by Address Output (Addrout) File ......... 96
Updating a Sequential File .................................................. 98
Deleting Records from a Sequential File .............................. 99
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating Consecutively</td>
<td>99</td>
</tr>
<tr>
<td>Example of Updating and Deleting Records</td>
<td>100</td>
</tr>
<tr>
<td>Updating Randomly by Relative Record Number</td>
<td>103</td>
</tr>
<tr>
<td>Updating Randomly by Relative Record Number and/or Consecutively</td>
<td>103</td>
</tr>
<tr>
<td>Updating Randomly by Address Output (Addrout) File</td>
<td>104</td>
</tr>
<tr>
<td>Adding Records to a Sequential File</td>
<td>104</td>
</tr>
<tr>
<td>Adding Records at the End of a File</td>
<td>105</td>
</tr>
<tr>
<td>Example of Adding Records at the End of a File</td>
<td>105</td>
</tr>
<tr>
<td>Adding Records between Records in a File</td>
<td>107</td>
</tr>
<tr>
<td>Example of Adding Records between Records in a File</td>
<td>109</td>
</tr>
<tr>
<td>DIRECT FILES</td>
<td>110</td>
</tr>
<tr>
<td>Creating a Direct File That Does Not Allow Deletions</td>
<td>111</td>
</tr>
<tr>
<td>Example of Creating a Direct File That Does Not Allow Deletions</td>
<td>112</td>
</tr>
<tr>
<td>Creating a Direct File That Allows Deletions</td>
<td>113</td>
</tr>
<tr>
<td>Example of Creating a Direct File That Allows Deletions</td>
<td>115</td>
</tr>
<tr>
<td>Reading a Direct File</td>
<td>115</td>
</tr>
<tr>
<td>Reading Consecutively</td>
<td>116</td>
</tr>
<tr>
<td>Example of Reading Consecutively</td>
<td>117</td>
</tr>
<tr>
<td>Reading Randomly by Relative Record Number</td>
<td>119</td>
</tr>
<tr>
<td>Example of Reading Randomly by Relative Record Number</td>
<td>119</td>
</tr>
<tr>
<td>Reading Randomly by Relative Record Number and/or Consecutively</td>
<td>123</td>
</tr>
<tr>
<td>Reading Randomly by Address Output (Addrout) File</td>
<td>124</td>
</tr>
<tr>
<td>Updating a Direct File</td>
<td>127</td>
</tr>
<tr>
<td>Deleting Records from a Direct File</td>
<td>127</td>
</tr>
<tr>
<td>Updating Consecutively</td>
<td>128</td>
</tr>
<tr>
<td>Updating Randomly by Relative Record Number</td>
<td>129</td>
</tr>
<tr>
<td>Example of Updating Randomly by Relative Record Number</td>
<td>129</td>
</tr>
<tr>
<td>Updating Randomly by Relative Record Number and/or Consecutively</td>
<td>132</td>
</tr>
<tr>
<td>Updating Randomly by Address Output (Addrout) File</td>
<td>132</td>
</tr>
<tr>
<td>Adding Records to a Direct File</td>
<td>133</td>
</tr>
<tr>
<td>INDEXED FILES</td>
<td>135</td>
</tr>
<tr>
<td>Creating an Indexed File</td>
<td>135</td>
</tr>
<tr>
<td>Example of Creating an Indexed File</td>
<td>137</td>
</tr>
<tr>
<td>Creating an Alternative Index File for an Indexed File</td>
<td>137</td>
</tr>
<tr>
<td>Example of Creating an Alternative Index File</td>
<td>139</td>
</tr>
<tr>
<td>Example of Using an Alternative Index File with Only One Field as its Key</td>
<td>139</td>
</tr>
<tr>
<td>Using an Alternative Index File with Noncontiguous Fields as its Key</td>
<td>141</td>
</tr>
<tr>
<td>Reading an Indexed File</td>
<td>141</td>
</tr>
<tr>
<td>Reading Sequentially by Key Field</td>
<td>142</td>
</tr>
<tr>
<td>Reading Sequentially within Key-Field Limits</td>
<td>143</td>
</tr>
<tr>
<td>Using a Limits Record</td>
<td>145</td>
</tr>
<tr>
<td>Using the SETLL Operation</td>
<td>147</td>
</tr>
<tr>
<td>Reading Randomly by Key Field</td>
<td>150</td>
</tr>
<tr>
<td>Reading Randomly and/or Sequentially by Key Field</td>
<td>151</td>
</tr>
<tr>
<td>Example of Reading an Indexed File Randomly and Sequentially by Key Field</td>
<td>152</td>
</tr>
<tr>
<td>Reading Randomly by Address Output (Addrout) File</td>
<td>153</td>
</tr>
<tr>
<td>Updating an Indexed File</td>
<td>156</td>
</tr>
<tr>
<td>Deleting Records from an Indexed File</td>
<td>156</td>
</tr>
<tr>
<td>Updating Sequentially by Key Field</td>
<td>157</td>
</tr>
<tr>
<td>Updating Sequentially within Key-Field Limits</td>
<td>158</td>
</tr>
<tr>
<td>Updating Randomly by Key Field</td>
<td>158</td>
</tr>
<tr>
<td>Example of Updating an Indexed File Randomly by Key Field</td>
<td>158</td>
</tr>
<tr>
<td>Creating Display Formats for CONSOLE Files</td>
<td>287</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Using the RPGR Procedure</td>
<td>288</td>
</tr>
<tr>
<td>Creating Display Formats for CONSOLE Files with CRTS36RPGR</td>
<td>290</td>
</tr>
<tr>
<td>Using Displays</td>
<td>294</td>
</tr>
<tr>
<td>Display Formats</td>
<td>294</td>
</tr>
<tr>
<td>Prompt Format</td>
<td>297</td>
</tr>
<tr>
<td>Changing the Display Format</td>
<td>297</td>
</tr>
<tr>
<td>Erasing the CONSOLE File Buffer</td>
<td>298</td>
</tr>
<tr>
<td>Using a CONSOLE File with KEYBORD and CRT Files</td>
<td>298</td>
</tr>
<tr>
<td>Using a KEYBORD File</td>
<td>298</td>
</tr>
<tr>
<td>File Description Specifications</td>
<td>299</td>
</tr>
<tr>
<td>Calculation Specifications for a KEY Operation</td>
<td>299</td>
</tr>
<tr>
<td>Using a KEY Operation</td>
<td>302</td>
</tr>
<tr>
<td>Bypassing a KEY Operation</td>
<td>302</td>
</tr>
<tr>
<td>Using a Message Member</td>
<td>303</td>
</tr>
<tr>
<td>Calculation Specifications for a SET Operation</td>
<td>304</td>
</tr>
<tr>
<td>Allowing Function Keys To Be Pressed</td>
<td>308</td>
</tr>
<tr>
<td>Using the SET and KEY Operations Together</td>
<td>309</td>
</tr>
<tr>
<td>Using a CRT File</td>
<td>310</td>
</tr>
<tr>
<td>File Description Specifications</td>
<td>310</td>
</tr>
<tr>
<td>Output Specifications</td>
<td>311</td>
</tr>
<tr>
<td>File- and Record-Identification Entries</td>
<td>311</td>
</tr>
<tr>
<td>Field-Description Entries</td>
<td>312</td>
</tr>
<tr>
<td>Displaying Data</td>
<td>312</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 11. Using a BSCA File</th>
<th>313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining a BSCA File</td>
<td>314</td>
</tr>
<tr>
<td>File Description Specifications</td>
<td>314</td>
</tr>
<tr>
<td>Telecommunications Specifications</td>
<td>316</td>
</tr>
<tr>
<td>Programming Considerations</td>
<td>318</td>
</tr>
<tr>
<td>First RPG Program Cycle</td>
<td>318</td>
</tr>
<tr>
<td>Removing Strings of Embedded Blanks</td>
<td>318</td>
</tr>
<tr>
<td>Removing Trailing Blanks</td>
<td>319</td>
</tr>
<tr>
<td>ASCII-EBDIC Character Translation</td>
<td>319</td>
</tr>
<tr>
<td>Control Breaks</td>
<td>320</td>
</tr>
<tr>
<td>Reclaim Resources</td>
<td>320</td>
</tr>
<tr>
<td>RPG Diagnostics</td>
<td>320</td>
</tr>
<tr>
<td>Configuring Your System for BSC</td>
<td>320</td>
</tr>
<tr>
<td>Translating T-SPEC Entries To AS/400 Configuration</td>
<td>320</td>
</tr>
<tr>
<td>Descriptions of BSC Functions</td>
<td>323</td>
</tr>
<tr>
<td>Receive-Only Function</td>
<td>323</td>
</tr>
<tr>
<td>Send-Only Function</td>
<td>324</td>
</tr>
<tr>
<td>Send-and-Receive Function</td>
<td>324</td>
</tr>
<tr>
<td>Send a File, Then Receive a File</td>
<td>324</td>
</tr>
<tr>
<td>Receive a File, Then Send a File</td>
<td>324</td>
</tr>
<tr>
<td>Send Records Interspersed with Receive Records</td>
<td>324</td>
</tr>
<tr>
<td>Systems That Use BSC</td>
<td>325</td>
</tr>
<tr>
<td>Device-Dependent Considerations</td>
<td>325</td>
</tr>
<tr>
<td>IBM 3740 Data Entry System</td>
<td>325</td>
</tr>
<tr>
<td>Restrictions</td>
<td>326</td>
</tr>
<tr>
<td>Single-File Support</td>
<td>326</td>
</tr>
<tr>
<td>Multiple-File Support</td>
<td>326</td>
</tr>
<tr>
<td>Blocked Records</td>
<td>327</td>
</tr>
<tr>
<td>RPG Specifications</td>
<td>327</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Defining Arrays and Tables</td>
<td>400</td>
</tr>
<tr>
<td>Loading Arrays and Tables</td>
<td>402</td>
</tr>
<tr>
<td>Loading Compile-Time Arrays and Tables</td>
<td>402</td>
</tr>
<tr>
<td>Loading Prerun-Time Arrays and Tables</td>
<td>403</td>
</tr>
<tr>
<td>Loading Run-Time Arrays</td>
<td>404</td>
</tr>
<tr>
<td>Array Information in One Record</td>
<td>405</td>
</tr>
<tr>
<td>Array Information in More than One Record</td>
<td>406</td>
</tr>
<tr>
<td>Using an Array Name and Index</td>
<td>407</td>
</tr>
<tr>
<td>Searching Arrays and Tables</td>
<td>408</td>
</tr>
<tr>
<td>Searching an Array</td>
<td>409</td>
</tr>
<tr>
<td>Searching One Table</td>
<td>411</td>
</tr>
<tr>
<td>Searching Related Tables</td>
<td>411</td>
</tr>
<tr>
<td>Specifying Arrays</td>
<td>413</td>
</tr>
<tr>
<td>Changing the Contents of Arrays and Tables</td>
<td>414</td>
</tr>
<tr>
<td>Changing the Contents Temporarily</td>
<td>414</td>
</tr>
<tr>
<td>Changing the Contents Permanently</td>
<td>415</td>
</tr>
<tr>
<td>Adding Entries to Arrays and Tables</td>
<td>415</td>
</tr>
<tr>
<td>Writing Arrays and Tables</td>
<td>416</td>
</tr>
<tr>
<td>Editing Arrays</td>
<td>417</td>
</tr>
<tr>
<td>Examples of Using Arrays</td>
<td>418</td>
</tr>
<tr>
<td>Example of Using Tables</td>
<td>426</td>
</tr>
<tr>
<td>File Description Specifications</td>
<td>427</td>
</tr>
<tr>
<td>Extension Specifications</td>
<td>427</td>
</tr>
<tr>
<td>Input Specifications</td>
<td>428</td>
</tr>
<tr>
<td>Calculation Specifications</td>
<td>428</td>
</tr>
<tr>
<td>Chapter 15. Using Data Structures</td>
<td>429</td>
</tr>
<tr>
<td>Coding a Data Structure</td>
<td>429</td>
</tr>
<tr>
<td>Data Structure Statement</td>
<td>430</td>
</tr>
<tr>
<td>Subfields</td>
<td>430</td>
</tr>
<tr>
<td>Rules for Coding Data Structures</td>
<td>430</td>
</tr>
<tr>
<td>Examples of Data Structures</td>
<td>431</td>
</tr>
<tr>
<td>Example 1. Defining One Area of Storage in More than One Way</td>
<td>431</td>
</tr>
<tr>
<td>Example 2. Defining Subfields within a Field</td>
<td>433</td>
</tr>
<tr>
<td>Example 3. Reorganizing Fields in an Input Record</td>
<td>435</td>
</tr>
<tr>
<td>Special Data Structures</td>
<td>435</td>
</tr>
<tr>
<td>SAVDS Data Structure</td>
<td>435</td>
</tr>
<tr>
<td>Local Data Area for a Display Station</td>
<td>436</td>
</tr>
<tr>
<td>File Information Data Structure</td>
<td>436</td>
</tr>
<tr>
<td>Chapter 16. Editing Numeric Fields</td>
<td>439</td>
</tr>
<tr>
<td>Edit Codes</td>
<td>440</td>
</tr>
<tr>
<td>Examples of Using the Currency Symbol with an Edit Code</td>
<td>444</td>
</tr>
<tr>
<td>Example of Using Asterisks with an Edit Code</td>
<td>446</td>
</tr>
<tr>
<td>Edit Words</td>
<td>449</td>
</tr>
<tr>
<td>Editing Considerations</td>
<td>449</td>
</tr>
<tr>
<td>Blanks (b)</td>
<td>450</td>
</tr>
<tr>
<td>Constants</td>
<td>450</td>
</tr>
<tr>
<td>The First Zero (0)</td>
<td>450</td>
</tr>
<tr>
<td>The First Asterisk (*)</td>
<td>450</td>
</tr>
<tr>
<td>The Minus Sign (-) and the Character “CR”</td>
<td>451</td>
</tr>
<tr>
<td>Currency Symbols</td>
<td>451</td>
</tr>
<tr>
<td>The Ampersand (&amp;)</td>
<td>451</td>
</tr>
<tr>
<td>Examples of Edit Words</td>
<td>452</td>
</tr>
</tbody>
</table>
Example 1 – Suppressing decimal point and leading zeros ........  452
Example 2 – Forcing the decimal point ..........................  452
Example 3 – Forcing leading zeros ................................ 452
Example 4 – Printing a leading zero in the leftmost position of output field ........................................  452
Example 5 – Indicating a negative value .........................  453
Example 6 – Inserting commas .....................................  453
Example 7 – Inserting a floating currency symbol ...............  453
Example 8 – Inserting a fixed currency symbol .................  454
Example 9 – Replacing leading blanks/constant by asterisks ....  454
Example 10 – Leading blank/constant replaced in first position only .  454
Example 11 – Adding text to negative numbers ..................  454
Example 12 – Adding text to all numbers ..........................  455
Example 13 – Using an edit word to print a check ..............  455
Example 14 – Using an edit word to print dates .................  455
Example 15 – Using an edit word to print a telephone number ..  455
Example 16 – Second occurrences of special characters are treated as constants ..............................................  456
Example 17 – Defining a single occurrence of an asterisk or zero as a constant ..............................................  456
Example 18 – Defining a single occurrence of CR as a constant ...  456
Creating Edit Words .......................................................  457

Chapter 17. Changing the Hexadecimal Value of Characters ..........  459
Changing the Collating Sequence ....................................  461
Coding the Changes .......................................................  461
Coding the Control Specification .....................................  461
Coding the Translation Table and Alternate Collating Sequence Coding Sheet ........................................  461
Coding the Records That Change the Collating Sequence ........  465
Example of a Record That Changes the Collating Sequence ......  466
Translating a File ............................................................  466
Coding the Translation .....................................................  466
Coding the Control Specification .....................................  466
Coding the Translation Table and Alternate Collating Sequence Coding Sheet ........................................  466
Coding the Records That Translate a File ..........................  467
Example of File Translation ..............................................  468

Chapter 18. Control Specification .......................................  471
Columns 1-2 (Page) ..........................................................  472
Columns 3-5 (Line) ..........................................................  472
Column 6 (Form Type) .....................................................  472
Column 7 (Comments) .....................................................  472
Columns 7-9 .................................................................  472
Columns 7-12 (/EJECT) .....................................................  473
Columns 7-12 (/TITLE) .....................................................  473
Columns 7-14 (/SPACE) .....................................................  473
Column 10 .................................................................  473
Column 11 .................................................................  474
Columns 12-14 .............................................................  474
Column 15 (Debug) ..........................................................  474
Columns 16-17 .............................................................  474
Column 18 (Currency Symbol) .........................................  475
Chapter 19. File Description Specifications ........................................ 483
File Description Charts .......................................................... 483
How to Use the Charts .......................................................... 484
Columns 1-2 (Page) ............................................................. 491
Columns 3-5 (Line) ............................................................... 491
Column 6 (Form Type) ........................................................... 491
Column 7 (Comments) ........................................................... 491
Columns 7-12 (/EJECT) ............................................................ 492
Columns 7-12 (/TITLE) ............................................................. 492
Columns 7-14 (/SPACE) ........................................................... 492
Columns 7-14 (Filename) .......................................................... 492
Column 15 (File Type) ............................................................... 493
  Input Files ................................................................................. 494
  Output Files ............................................................................. 494
  Update Files ............................................................................ 494
  Combined Files ...................................................................... 494
Column 16 (File Designation) ......................................................... 494
  Primary Files .......................................................................... 495
  Secondary Files ...................................................................... 496
  Full-Procedural Files ............................................................ 496
  Chained Files ........................................................................ 496
  Record Address Files ............................................................ 496
  Array or Table Files ............................................................. 496
  Demand Files ......................................................................... 497
Column 17 (End Of File) ................................................................. 497
Column 18 (Sequence) ................................................................. 497
Column 19 (File Format) .............................................................. 498
Columns 20-23 .......................................................................... 498
Columns 24-27 (Record Length) .................................................... 498
Column 28 (Mode of Processing) .................................................... 500
Columns 29-30 (Length of Key Field or Record Address Field) ........ 501
# Chapter 22. Telecommunications Specifications

<table>
<thead>
<tr>
<th>Columns 1-2 (Page)</th>
<th>532</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns 3-5 (Line)</td>
<td>532</td>
</tr>
<tr>
<td>Column 6 (Form Type)</td>
<td>532</td>
</tr>
<tr>
<td>Column 7 (Comments)</td>
<td>532</td>
</tr>
<tr>
<td>Column 7-12 (/EJECT)</td>
<td>532</td>
</tr>
<tr>
<td>Columns 7-12 (/TITLE)</td>
<td>533</td>
</tr>
<tr>
<td>Columns 7-14 (/SPACE)</td>
<td>533</td>
</tr>
<tr>
<td>Columns 7-14 (Filename)</td>
<td>533</td>
</tr>
<tr>
<td>Columns 15-17 (Line Number—Number of Lines per Page)</td>
<td>533</td>
</tr>
<tr>
<td>Columns 18-19 (Form Length)</td>
<td>534</td>
</tr>
<tr>
<td>Columns 20-22 (Line Number—Overflow Line)</td>
<td>534</td>
</tr>
<tr>
<td>Columns 23-24 (Overflow Line)</td>
<td>534</td>
</tr>
<tr>
<td>Columns 25-74</td>
<td>534</td>
</tr>
<tr>
<td>Columns 75-80 (Program Identification)</td>
<td>535</td>
</tr>
</tbody>
</table>

## Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Column 15 (Configuration)</th>
<th>540</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 16 (Type of Station)</td>
<td>541</td>
</tr>
<tr>
<td><strong>Column 17 (Type of Control)</strong></td>
<td>541</td>
</tr>
<tr>
<td><strong>Column 18 (Type of Code)</strong></td>
<td>541</td>
</tr>
<tr>
<td><strong>Column 19 (Transparency)</strong></td>
<td>541</td>
</tr>
<tr>
<td><strong>Column 20 (Switched)</strong></td>
<td>542</td>
</tr>
<tr>
<td>Columns 21-31</td>
<td>542</td>
</tr>
<tr>
<td><strong>Column 32 (Location of Identification—This Station)</strong></td>
<td>542</td>
</tr>
<tr>
<td><strong>Columns 33-39 (Identification—This Station)</strong></td>
<td>542</td>
</tr>
<tr>
<td><strong>Column 40 (Location of Identification—Remote Station)</strong></td>
<td>543</td>
</tr>
<tr>
<td><strong>Columns 41-47 (Identification—Remote Station)</strong></td>
<td>543</td>
</tr>
<tr>
<td>Columns 48-51</td>
<td>543</td>
</tr>
<tr>
<td><strong>Column 52 (ITB)</strong></td>
<td>543</td>
</tr>
<tr>
<td>Columns 53-54 (Permanent-Error Indicator)</td>
<td>544</td>
</tr>
<tr>
<td><strong>Columns 55-57 (Wait Time)</strong></td>
<td>545</td>
</tr>
<tr>
<td>Columns 58-59 (Record-Available Indicator)</td>
<td>545</td>
</tr>
<tr>
<td>Column 60 (Last File)</td>
<td>545</td>
</tr>
<tr>
<td><strong>Columns 61-62 (Polling Characters)</strong></td>
<td>546</td>
</tr>
<tr>
<td>Columns 63-64 (Addressing Characters)</td>
<td>546</td>
</tr>
<tr>
<td>Columns 65-74</td>
<td>546</td>
</tr>
<tr>
<td>Columns 75-80 (Program Identification)</td>
<td>546</td>
</tr>
</tbody>
</table>

## Chapter 23. Input Specifications

<table>
<thead>
<tr>
<th>File and Record-Type Identification Entries</th>
<th>548</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns 1-2 (Page)</td>
<td>548</td>
</tr>
<tr>
<td>Columns 3-5 (Line)</td>
<td>548</td>
</tr>
<tr>
<td>Column 6 (Form Type)</td>
<td>548</td>
</tr>
<tr>
<td>Column 7 (Comments)</td>
<td>548</td>
</tr>
<tr>
<td>Columns 7-12 (/EJECT)</td>
<td>548</td>
</tr>
<tr>
<td>Columns 7-12 (/TITLE)</td>
<td>549</td>
</tr>
</tbody>
</table>
Chapter 24. Calculation Specifications
Chapter 25. Output Specifications

Columns 1-2 (Page) ........................................ 603
Columns 3-5 (Line) ......................................... 603
Column 6 (Form Type) ...................................... 604
Column 7 (Comments) ...................................... 604
Columns 7-12 (/EJECT) ..................................... 604
Columns 7-12 (/TITLE) ..................................... 605
Columns 7-14 (/SPACE) .................................... 605
Columns 7-14 (Filename) .................................. 605
Columns 14-16 (AND/OR) .................................. 606
Column 15 (Type) ......................................... 606
  Heading Records (H) ..................................... 607
  Detail Records (D) ....................................... 607
  Total Records (T) ....................................... 607
  Exception Records (E) ................................... 607
Columns 16-18 (ADD/DEL) ................................. 608
  ADD ..................................................... 608
  DEL ..................................................... 608
Column 16 (Fetch Overflow or Release) ..................... 609
  Fetch Overflow ......................................... 609
  Release ................................................. 609
Columns 17-22 (Spacing and Skipping) ....................... 610
  Column 17 (Space Before) .............................. 610
  Column 18 (Space After) ............................... 610
  Columns 19-20 (Skip Before) ........................... 610
  Columns 21-22 (Skip After) ............................ 610
Columns 23-31 (Output Indicators) .......................... 611
Columns 32-37 (Field Name) ................................ 612
  Field Names ............................................ 612
  Rules for Field Names .................................. 612
Special Words ............................................. 612
  Page Numbering (PAGE, PAGE1-PAGE7) ............... 612
  Repeating Output Fields (*PLACE) ..................... 613
  Date Fields (UDATE, UMONTH, UDAY, UYEAR) ........ 616
  EXCPT Names ............................................ 617
Column 38 (Edit Codes) .................................. 617
Column 39 (Blank After) .................................. 618
Columns 40-43 (End Position in Output Record) .......... 618
Column 44 (Packed-Decimal or Binary Field) ............. 619
Chapter 26. The Auto Report Feature

The Auto Report Source Program
Compiling the Auto Report Source Program
Format of Created Specifications
Order of Created Specifications
Calculation Specifications
Output Specifications
Comment Statements
Restriction
Auto Report Option Specifications (U)
Column 6 (Form Type)
Column 7 (Source)
Columns 8-24 (Source Member Reference)
Columns 25-26
Column 27 (Date Suppress)
Column 28 (*Suppress)
Column 29
Column 30 (List Options)
Columns 31-74
Auto Report /COPY Statement Specifications
Changing Copied Specifications
Changing File Description Specifications
Changing Input Field Specifications
Auto Report Page-Heading Specifications
Records-Description Specifications
Columns 7-14 (Filename)
Column 15 (Type)
Column 16
Columns 17-22 (Spacing and Skipping)
Columns 23-31 (Output Indicators)
Columns 32-37 (*AUTO)
Columns 38-70
Field-Description Specifications
Columns 7-31
Columns 32-37 (Field Name)
Column 38 (Edit Codes)
Column 39 (Blank After)
Columns 40-44
Columns 45-70 (Constant or Edit Word)
Auto Report Output Specifications
Records-Description Specifications
Columns 7-14 (Filename)
Column 15 (Type)
Column 16 (Fetch Overflow)
Columns 17-22 (Spacing and Skipping)
Columns 23-31 (Output Indicators)
Columns 32-37 (*AUTO)
Using AUTOC to Compile an Auto Report Source Program in the System/36 Environment ............................................. 682
Calling AUTOC .................................................................. 682
Using the First AUTOC Display ........................................ 682
Using the Second AUTOC Display .................................... 684
Examples of Using Auto Report ........................................ 686
Example 1—Sales Report using "AUTO .......................... 686
Example 2—Sales Report with Three Levels of Totals ............ 690
Example 3—Sales Report with Group Heading ................ 692
Example 4—Sales Report with Cross-Column Totals .......... 694
Example 5—Copy Function for Specifications of Sales Report ... 696
Example 6—Override Copied Input Specifications ............... 699
Example 7—Cash Receipts Register ................................ 701
Control Specification ...................................................... 705
/COPY Statements ....................................................... 707
Calculation Specifications ............................................... 707
"AUTO Specifications ................................................... 711

Chapter 27. Operation Codes ............................................ 713
Arithmetic Operations ..................................................... 716
Move Operations ............................................................ 716
Move Zone Operations .................................................... 717
Compare and Testing Operations ...................................... 717
Structured Programming Operations ................................ 718
Bit Operations .............................................................. 719
SETON and SETOF Operations ........................................ 720
Branching within RPG ................................................... 720
Subroutine Operations ................................................... 721
Calling External Subroutines or Programs ......................... 721
WORKSTN Operations .................................................... 721
Programmed Control of Input and Output ......................... 721
OPERATION CODES ...................................................... 722
ACQ (Acquire) ............................................................. 722
ADD (Add) ................................................................. 723
BEGSR (Begin Subroutine) ............................................ 724
BITOF (Set Bit Off) ...................................................... 725
BITON (Set Bit On) ...................................................... 728
CALL (Call a Program) .................................................. 730
CASxx (Case) ............................................................. 732
CHAIN (Chain) ........................................................... 734
Random Processing ....................................................... 735
COMP (Compare) ......................................................... 737
DEBUG (Debug) .......................................................... 740
Records Written for DEBUG ........................................... 741
*LIKE DEFN (Field Definition) ........................................ 742
DIV (Divide) ............................................................... 743
DO (Do) ..................................................................... 745
DOUxx (Do Until) ........................................................ 748
DOWxx (Do While) ....................................................... 750
ELSE (Else Do) ........................................................... 752
END (End) ................................................................. 753
ENDSR (End Subroutine) ............................................... 754
EXCPT (Exception Output) ............................................ 755
EXIT (Exit to an External Subroutine) ............................. 761
<table>
<thead>
<tr>
<th><strong>EXSR (Execute Subroutine)</strong></th>
<th>763</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coding Subroutines</strong></td>
<td>763</td>
</tr>
<tr>
<td><strong>FORCE (Force)</strong></td>
<td>765</td>
</tr>
<tr>
<td><strong>FREE (Deactivate a Program)</strong></td>
<td>767</td>
</tr>
<tr>
<td><strong>GOTO (Branch To)</strong></td>
<td>768</td>
</tr>
<tr>
<td><strong>IFxx (If/Then)</strong></td>
<td>771</td>
</tr>
<tr>
<td><strong>KEY (Key)</strong></td>
<td>773</td>
</tr>
<tr>
<td><strong>LOKUP (Lookup)</strong></td>
<td>774</td>
</tr>
<tr>
<td><strong>Array LOKUP</strong></td>
<td>774</td>
</tr>
<tr>
<td><strong>Table LOKUP</strong></td>
<td>774</td>
</tr>
<tr>
<td><strong>MHHZO (Move High to High Zone)</strong></td>
<td>776</td>
</tr>
<tr>
<td><strong>MHLZO (Move High to Low Zone)</strong></td>
<td>776</td>
</tr>
<tr>
<td><strong>MLHZO (Move Low to High Zone)</strong></td>
<td>776</td>
</tr>
<tr>
<td><strong>MLLZO (Move Low to Low Zone)</strong></td>
<td>777</td>
</tr>
<tr>
<td><strong>MOVE (Move)</strong></td>
<td>778</td>
</tr>
<tr>
<td><strong>MOVEA (Move Array)</strong></td>
<td>781</td>
</tr>
<tr>
<td><strong>MOVEL (Move Left)</strong></td>
<td>797</td>
</tr>
<tr>
<td><strong>MULT (Multiply)</strong></td>
<td>800</td>
</tr>
<tr>
<td><strong>MVR (Move Remainder)</strong></td>
<td>801</td>
</tr>
<tr>
<td><strong>NEXT (Next)</strong></td>
<td>802</td>
</tr>
<tr>
<td><strong>PARM (Identify Parameters)</strong></td>
<td>803</td>
</tr>
<tr>
<td><strong>PLIST (Identify a Parameter List)</strong></td>
<td>805</td>
</tr>
<tr>
<td><strong>Rules for Specifying PLIST</strong></td>
<td>805</td>
</tr>
<tr>
<td><strong>POST (Post)</strong></td>
<td>806</td>
</tr>
<tr>
<td><strong>READ (Read)</strong></td>
<td>807</td>
</tr>
<tr>
<td><strong>READE (Read Equal Key)</strong></td>
<td>809</td>
</tr>
<tr>
<td><strong>READP (Read Prior Record)</strong></td>
<td>810</td>
</tr>
<tr>
<td><strong>REL (Release)</strong></td>
<td>811</td>
</tr>
<tr>
<td><strong>RETRN (Return to Caller)</strong></td>
<td>812</td>
</tr>
<tr>
<td><strong>RLABL (RPG Label)</strong></td>
<td>813</td>
</tr>
<tr>
<td><strong>Message-Retrieving Subroutine (SUBR23)</strong></td>
<td>815</td>
</tr>
<tr>
<td><strong>SET (Set)</strong></td>
<td>817</td>
</tr>
<tr>
<td><strong>SETLL (Set Lower Limits Operation)</strong></td>
<td>818</td>
</tr>
<tr>
<td><strong>SETOF (Set Off)</strong></td>
<td>819</td>
</tr>
<tr>
<td><strong>SETON (Set On)</strong></td>
<td>819</td>
</tr>
<tr>
<td><strong>SHTDN (Shutdown)</strong></td>
<td>820</td>
</tr>
<tr>
<td><strong>SORTA (Sort an Array)</strong></td>
<td>821</td>
</tr>
<tr>
<td><strong>SQRT (Square Root)</strong></td>
<td>822</td>
</tr>
<tr>
<td><strong>SUB (Subtract)</strong></td>
<td>823</td>
</tr>
<tr>
<td><strong>TAG (Tag)</strong></td>
<td>824</td>
</tr>
<tr>
<td><strong>TESTB (Test Bit)</strong></td>
<td>825</td>
</tr>
<tr>
<td><strong>Columns 54-55:</strong></td>
<td>825</td>
</tr>
<tr>
<td><strong>Columns 56-57:</strong></td>
<td>825</td>
</tr>
<tr>
<td><strong>Columns 58-59:</strong></td>
<td>825</td>
</tr>
<tr>
<td><strong>TESTZ (Test Zone)</strong></td>
<td>827</td>
</tr>
<tr>
<td><strong>TIME (Time of Day)</strong></td>
<td>828</td>
</tr>
<tr>
<td><strong>XFOOT (Summing the Elements of an Array)</strong></td>
<td>829</td>
</tr>
<tr>
<td><strong>Z-ADD (Zero and Add)</strong></td>
<td>829</td>
</tr>
<tr>
<td><strong>Z-SUB (Zero and Subtract)</strong></td>
<td>829</td>
</tr>
</tbody>
</table>

**Chapter 28. Using Double-Byte Data** ............................................ 831
**Specifying Double-byte Data** ................................................... 831
**Double-byte Literals and Constants** ........................................ 831
**Double-byte Fields, Tables, and Arrays** .................................. 832
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 licensed program or other IBM product in this publication is not intended to state or imply that only IBM’s program or other product may be used. Any functionally equivalent product, program, or service that does not infringe on 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.

Important changes or additions 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.

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

- Application System/400
- AS/400
- ILE
- MVS/SP
- Operating System/400
- OS/400
- RPG/400
- 400

© Copyright IBM Corp. 1994
About This Manual

This manual is a guide and reference for the RPG II programming language on the Application System/400 (AS/400)* system.

This manual refers to other IBM products. These publications are listed in the “Bibliography” on page 845 with their full title and base order numbers. When they are referred to in the text, a shortened version of the title is used.

Who Should Use This Manual

This manual is intended for application programmers who have a basic understanding of data processing concepts and the RPG II programming language, and who write or maintain applications in RPG II. For convenience, RPG II is often referred to as RPG.

Using this manual, you will be able to:

- Design RPG II programs
- Code RPG II programs
- Enter and compile RPG II programs
- Test and debug RPG II programs
- Study coded RPG II examples and sample programs.

What You Should Know

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

- You should know how to create job queues and output queues. This information is contained in Programming: Data Management Guide, SC41-9658.

- You should be familiar with your display station (also known as a work station), and its controls. Some of the keys on the keyboard are standard, regardless of the software system that is currently running at the display station or the hardware system that the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Function keys
  - Field exit keys
  - Insert and Delete keys
  - Error Reset key
  - Enter key.

For information about the display station, refer to:


- You should know how to operate your display station when it is attached to the IBM AS/400 system and running AS/400 software. This means knowing how to use the Operating System/400 (OS/400)* system and the Control Language (CL) to do such things as:
  - Sign on and sign off the display station
  - Interact with displays
  - Use Help
  - Enter control commands and procedure commands
About This Manual

- Call up utilities
- Respond to messages.

To find out more about this operating system and its control language, refer to:

- *Programming: Control Language Reference, SC41-0030*
- *Programming: Reference Summary, SX41-0028*
- *Programming: Control Language Programmer's Guide, SC41-8077*

- You should know how to call and use certain available utilities:
  - The Screen Design Aid (SDA) utility, which you can use to design and to code displays. This information is contained in *Application Development Tools: Screen Design Aid User’s Guide and Reference.*
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and to update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User’s Guide and Reference.*

- You should be familiar with how the RPG program cycle works, how indicators affect the program cycle, and how to code entries on the RPG specification sheets.

These general items about RPG are taught in an RPG II coding class.

- You should know how to interpret displayed and printed messages. This information is contained in “Using, Displaying, and Printing Messages” on page 61 in Chapter 4, “Testing and Debugging an RPG Program.”

### Naming Conventions

In this manual, the following conventions are used for program, display, and menu names:

- Program names use the format aannnR, where:
  - aa identifies the type of application, and can be:
    - AR for accounts receivable
    - IM for inventory management
    - OE for order entry.
  - nnn is a number that identifies the type of program, and can be:
    - 100 to 199 for data entry
    - 200 to 299 for inquiry
    - 300 to 349 for file maintenance
    - 350 to 399 for sort
    - 400 to 499 for file update
    - 900 to 949 for printing reports and program listings.
  - R identifies the programming language as RPG.

- Display names are formed by adding a D to the end of the name of the program that uses the display. For example, AR230RD is the name of a display used by an RPG accounts receivable inquiry program. If the program uses more than one display, a sequence number is added to the display name. For example, if program AR230R uses two displays, the displays are named AR230RD1 and AR230RD2.
Coding Conventions

Where specification sheets show the columns to code, shading indicates that no entry is allowed in the column. No coding in an unshaded column indicates that more than one entry is allowed in the column. A character in an unshaded column indicates that the character is the only entry allowed in the column.

How To Use the RPG II Coding Examples

Throughout this book there are programming examples to assist you in understanding RPG II. These programming examples are scaled in such a way that you can use the RPG Debugging Template, GX21-9129 to check the coding in the programs. The following figure is used to explain the format of the examples.

All examples do not contain line numbers. In the example as shown in the following figure each line starts with a C for calculation specification. To determine which column contains the coding, place the debugging template over the example, aligning the C on the template with the C in the example. The columns on the debugging template are the same width as the example. If you are not using a debugging template, use the prompt line at the top of the example to help you.

**Note:** All coding lines are highlighted, while all comment lines are not highlighted. All comment lines refer to the coding following the comment.
About This Manual

*.. 1 ..+.. 2 ..+.. 3 ..+.. 4 ..+.. 5 ..+.. 6 ..+.. 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C*
C* Indicator 01 is on for an input record which deletes a disk record.
C* Indicator 21 is set for use during update of the MASTER file.
C*
C  01 MR SETON 21
C*
C* The values in the MASTER file are updated by totals accumulated
C* from transaction records.
C*
C  02 MR ADD CHG TOTCHG 62
C  02 MR ADD PAYMT TOTPAY 62
C  02 MR ADD CRDT TOTCRD 62
C  03 MR ADD TOTCHG CHARGE
C*
C  03 MR ADD TOTPAY PAY
C  03 MR ADD TOTCRD CREDIT
C  03 MR SUB TOTPAY BAL
C  03 MR SUB TOTCRD BAL
C*
C* Set off indicator 21 for next cycle.
C*
CL1 SETOF 21
C*
C* Clear accumulator fields for the next cycle.
C*
CL1 SUB TOTCHG TOTCHG
CL1 SUB TOTPAY TOTPAY
CL1 SUB TOTCRD TOTCRD

An RPG II Coding Example

The following online information is available on the AS/400 system. After pressing
the Help key on any menu, press the Help key a second time to see an explanation
of how the online information works, including the index search function. You can
press either the Help key or F1 for help.

Index Search

Index search allows you to specify words or phrases that identify the information
that you want to see. To use index search, press the Help key, then press F11
(Search index). You can also use index search by entering the Start Index Search
(STRIDXSCH) command on any command line or by selecting option 2 on the User
Support and Education menu.
To access index search for AS/400 languages that have it available, you must be using the source entry utility (SEU) to edit source statements. When the cursor is in the data entry area, press Help and then F11 (Search index). To access index search for Application Development Tools when you are using SEU, make sure your cursor is not in the data area, press Help, and then F11 (Search index).

**Help for Control Language Commands**

To see prompts for parameters for a control language command, type the command, then press F4 (Prompt) instead of the Enter key. To see extended help for the command, type the command and press the Help key. You can see contextual help for CL commands also.
Chapter 1. An Introduction to RPG II and the AS/400 System

RPG II is a programming language designed to make it easier to create software applications that process data for businesses and commercial institutions.

RPG II is a language under evolution. A slightly different version of RPG II is available on each machine that supports it. The AS/400 system is the most recent of these computing systems. It offers a new enhanced version of RPG II, and supports the previous version of RPG II available on System/36 computers. For more information on differences between RPG II on the AS/400 system and on System/36 see Appendix A, “Differences between System/36 RPG II and AS/400 System/36-Compatible RPG II.”

The sections in this chapter describe the following:
- OS/400 and the Control Language
- Accessing RPG II from OS/400
- The two environments: AS/400 system or the System/36 Environment
- Files used by RPG II
- The RPG II programming cycle
- Designing your RPG II program
- Structured programming in RPG II
- The RPG Screen Design Aid Utility (RPGSDA).

OS/400 and the Control Language

The AS/400 system has an operating system and an associated control language.

The AS/400 Operating System

The operating system that controls all your interactions with AS/400 system is called the Operating System/400 (OS/400). From your display station, OS/400 allows you to:
- Sign on and sign off the AS/400 system
- Interact with the displays
- Use Help
- Enter control commands and procedures
- Respond to messages
- Manage files
- Call up utilities and run programs.

Refer to the Publications Guide for a complete list of publications, which describe the support provided on the AS/400 system.

The AS/400 Control Language

OS/400 is manipulated by the Control Language (CL). You interact with OS/400 by entering or selecting CL commands. Many AS/400 system menus display CL commands or command parameters. You can enter these commands or parameters directly from the menu. You can also enter CL commands directly into your AS/400 system.
The AS/400 CL commands that you will be using most often with RPG II are:

- STRSEU to call the Source Entry Utility (SEU), a full-screen editor that can be used to enter RPG II program code.
- CRTS36RPGR to create display formats for a CONSOLE file.
- CRTS36RPG to compile RPG II source programs.
- CRTS36RPT to compile an auto report source program.
- STRS36 to access the System/36 Environment.
- ENDS36 to exit from the System/36 Environment.

The Control Language and all its commands are described in detail in the Programming: Control Language Reference and the Programming: Control Language Programmer’s Guide.

Accessing RPG II from OS/400

When you start working on the AS/400 system you will see the following display.

**Note:** Your site may use a different sign-on display. If it does, contact your system operator for further information.

---

Figure 1. OS/400 Sign-On Display

When you enter your ID and password, the following display appears, and you can begin working in OS/400.

**Note:** If you have tailored your user profile, a different display can appear.
To begin working in RPG II you would enter or select the appropriate CL command, or you could select option 5 (Programming), and go through the menus.

Your Choice of Two Environments: AS/400 System or the System/36 Environment

The AS/400 system offers many enhancements over System/36 and System/38. However, because a great many RPG II programs have been written for the System/36 computer, and because many programmers are already familiar with the System/36, the AS/400 system also supports these programs. You should be aware of the differences between the two systems that are described in Appendix A, “Differences between System/36 RPG II and AS/400 System/36-Compatible RPG II.”

The System/36 Environment

The CL command STRS36 allows you to enter the System/36 Environment. The System/36 Environment emulates a System/36. When you are in this environment, you can enter and compile RPG II programs, as if you were using a System/36 machine. The displays are similar to the displays in System/36. File naming conventions and system responses are the same as in System/36.

System/36 RPG II programs must be compiled with a System/36 RPG II compatible compiler, but not necessarily while in the System/36 Environment. System/36 programs compiled in this way will run only in the System/36 Environment.

Using the System/36 RPG II Procedures

From within the System/36 Environment you can use any of the following System/36 procedures to enter and compile an RPG program:

- RPGSEU, which lets you create or change a procedure or an RPG or auto report source member.
- RPGC, which lets you compile an RPG source program.
- RPGX, which lets you print a cross-reference listing for an RPG source program.
- AUTOC, which lets you compile an RPG source program that contains auto report specifications.
- RPGR, which lets you create display formats from a console file.
- RPGSDA, which lets you create or change a display format.

Some of these procedures contain prompts, which were valid for the System/36, but which are no longer relevant in the System/36 Environment. For the convenience of those programmers who have created procedures or written operating instructions incorporating these prompts, the prompts will still appear on the System/36 Environment displays. Whatever you enter in response, however, will be ignored.

To begin one of these procedures, use any one of the following methods:

- Include any of the RPG procedures in one of your own procedures and run your own procedure.
- Call the procedure directly by using one of the following methods:
  - Type HELP, a space, and the procedure name (with or without parameters) on the command line, and press Enter.
  - Type the procedure name on the command line of the display, and press Enter.
  - Type the procedure name (with or without parameters) on the command line, and press F4.
  - Type the procedure name, a space, and the source member name on the command line, and press Enter.
Files Used by RPG II

Information is stored on the AS/400 system in a series of libraries. In each of these libraries is a series of files. To name a file, you must specify the file name and the library name. The convention is library-name/file-name.

Each file that your program uses must be assigned to an input/output device. You code that device name in columns 40 through 46 of the file description specifications. For example, if a file uses a disk as an input/output device, you code DISK as the device name. The coded device name is a convenient way to refer to each kind of file. Therefore, a disk is called a DISK file, a file assigned to a display station is called a WORKSTN file, and so on.

Some files (CONSOLE and KEYBORD) can be used only for input, some (CRT and PRINTER) can be used only for output, and some (DISK, WORKSTN, SPECIAL, and BSCA) can be used for both input and output. Therefore, when you use an RPG program, you must be familiar with each kind of file.

Typical RPG programs, and the kinds of files you might use for input and for output, are as listed below:

<table>
<thead>
<tr>
<th>What the RPG Program Does</th>
<th>Files You Can Use for Input</th>
<th>Files You Can Use for Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquire into a file</td>
<td>DISK</td>
<td>PRINTER</td>
</tr>
<tr>
<td></td>
<td>WORKSTN</td>
<td>WORKSTN</td>
</tr>
<tr>
<td></td>
<td>SPECIAL</td>
<td>SPECIAL</td>
</tr>
<tr>
<td>Process a file</td>
<td>DISK</td>
<td>DISK</td>
</tr>
<tr>
<td></td>
<td>WORKSTN</td>
<td>WORKSTN</td>
</tr>
<tr>
<td></td>
<td>CONSOLE</td>
<td>CRT</td>
</tr>
<tr>
<td></td>
<td>KEYBORD</td>
<td>PRINTER</td>
</tr>
<tr>
<td></td>
<td>SPECIAL</td>
<td>SPECIAL</td>
</tr>
<tr>
<td>Enter data</td>
<td>DISK</td>
<td>DISK</td>
</tr>
<tr>
<td></td>
<td>WORKSTN</td>
<td>WORKSTN</td>
</tr>
<tr>
<td></td>
<td>CONSOLE</td>
<td>CRT</td>
</tr>
<tr>
<td></td>
<td>KEYBORD</td>
<td>PRINTER</td>
</tr>
<tr>
<td></td>
<td>SPECIAL</td>
<td>SPECIAL</td>
</tr>
<tr>
<td></td>
<td>BSCA</td>
<td>BSCA</td>
</tr>
<tr>
<td>Print a report</td>
<td>DISK</td>
<td>PRINTER</td>
</tr>
<tr>
<td></td>
<td>WORKSTN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONSOLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEYBORD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPECIAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BSCA</td>
<td></td>
</tr>
</tbody>
</table>

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

For example, if you wanted to create a program to display accounts receivable information about customers, you would probably use a DISK file for input, a WORKSTN file for both input and output, and possibly a PRINTER file for output as well. The DISK input file would be a customer master file containing all the accounts receivable records for your customers. As an input file, the WORKSTN file would select the desired records from the DISK file. As an output file, the WORKSTN file would display the selected records from the DISK file. The program might do no processing other than to read records from the DISK file and to write records to the WORKSTN file. Sample program AR230R in Chapter 7, “Using a WORKSTN File” shows this example. If you used a PRINTER file also, you could write the output records to the printer.

When you are ready to code your program, read the chapters that explain how to code a program that uses the kinds of files your program uses. For example, for information about coding a program that uses a DISK file, a WORKSTN file, and a PRINTER file, you should read Chapter 6, “Using a DISK File,” Chapter 7, “Using a WORKSTN File,” and Chapter 8, “Using a PRINTER File.”

RPG Program Cycle

The RPG program cycle controls certain operations done on each record, so the program cycle partly determines how you can process your data. The phrase program cycle refers to the series of operations that an RPG program does automatically on each record that it reads.

Each RPG program goes through the same general cycle of operations. This cycle of operations has three basic logic steps:

- Reading information (input)
- Doing calculations (processing)
- Writing results (output).

These basic logic steps can be divided into several substeps in which you can assign indicators to control when calculation and output operations occur. For more information on indicators, see Chapter 13, “Using Indicators.”

Calculation and output operations occur at two different times in a program cycle: detail time and total time (see Figure 3 on page 8). At detail time, the RPG program calculates and writes data for one or more records. For example, if a customer uses his charge account three times and we print a record of each charge, we are printing three detail records. At total time, the program calculates and writes data for a series of related records. If a customer uses his charge account three times and we print only one record that shows the total of the three charges, we are printing one total record.

Generally, totals are calculated and written for data accumulated from a group of related records, called a control group. A control group is a set of records all having the same information in a control field. In an accounts receivable program, for example, you could use the customer account number as a control field; in an inventory program, you could use the part number. Each time a record is read, the program checks the information in the control field to determine if it differs from the control-field information in the previous record. When the information differs, a control break occurs. A control break means that all records from a control group have been read and that a new group is starting. When all records from a control
After the program has read the group that have been read, the program does the total operations coded for that group. Data from the record that starts the new control group is not included in the total operations.

To indicate a field that is a control field, you assign one of the control-level indicators (L1 through L9) to that field in columns 59 and 60 of the input specifications. To indicate calculations that are total calculations, write this same control-level indicator in columns 7 and 8 of those calculation specifications. Those calculations that do not have a control-level indicator written in columns 7 and 8 are detail calculations. On the output specifications, you do not use control-level indicators to identify detail and total records. Instead, you use a T in column 15 of the output specifications to indicate a total output operation, and you use an H (for heading) or a D (for detail) to indicate a detail output operation.

The program does detail calculations and detail output operations for each record it reads (that is, for each program cycle) if all conditioning indicators are satisfied, regardless of whether it does total calculations or total output. Detail calculations and detail output operations occur in either of the following cases:

- All total calculation and total output operations are complete for a control group, but the last record is not processed.
- No total operations are to be done (the information in the control field has not changed).

Figure 3 on page 8 shows the basic steps in the RPG program cycle. Figure 4 on page 9 shows a flowchart for the same steps. A program cycle begins with step 1 and continues through step 11. Then the next cycle begins again with step 1. Steps 7 and 8 are known as total time, and steps 1 and 11 are known as detail time.
Figure 3. Steps in the RPG Program Cycle
RPG PROGRAM CYCLE

Figure 4. Flowchart of the RPG Program Cycle

Chapter 1. An Introduction to RPG II and the AS/400 System 9
The following steps briefly describe what the RPG program does at each step in the program cycle. The steps are the same as those shown in Figure 3 on page 8 and Figure 4 on page 9.

1. If the conditioning indicators are satisfied, the program does the heading or detail output (those lines that have H or D in column 15 of the output specifications).

2. The program turns off all control-level and record-identifying indicators.

3. The program reads a record and turns on the appropriate record-identifying indicator.

4. The program determines if a control break occurred. (A control break occurs when the control field of the record just read differs from the control field of the previous record.)

5. If a control break occurs, the program turns on the proper control-level indicator and all lower control-level indicators except L0, which is always on.

6. If this is the first cycle, the program goes to step 9.

7. The program does total calculations (those conditioned by control-level indicators in columns 7 and 8 of the calculation specifications) if the appropriate control-level indicators are on.

8. The program does total output operations (those lines that have T in column 15 of the output specifications) if the indicators on those lines are on.

9. The program determines if the last-record indicator is on. If it is, all records have been processed, and the program ends.

10. The program makes data from the record read at the beginning of the cycle (step 3) available for use in detail calculations and output.

11. The program does all detail calculations (those not conditioned by control-level indicators in columns 7 and 8 of the calculation specifications) on the data from the record read at the beginning of the cycle.

The first and last cycles of a program differ somewhat from the other cycles. Before the first record is read in the first cycle, the program prints lines conditioned by the first-page (1P) indicator. The program also processes any heading or detail output operations having no conditioning indicators or all negative conditioning indicators. Heading lines printed before the first record is read might consist of constants, page headings, or fields for reserved words such as PAGE and UDATE. In addition, the program bypasses total calculations and total output steps.

During the last program cycle, when no more records are available, the last-record indicator turns on, automatically causing all control-level indicators to turn on. The program processes the total calculations and total output operations, and the program ends.
Designing Your RPG II Program

The design and creation of a program in RPG II is basically the same as for any program:

1. Design the program.
2. Manually code the program onto specification forms.
3. Enter the code into the system to produce the source file(s).
4. Compile the source file(s) to produce the program object.
5. Test and debug the program.
6. Put the program into production.

Figure 5 on page 12 illustrates these steps.

Nowadays, with interactive online editors like the Source Entry Utility (SEU), manual coding is not always necessary. Step 2 can be eliminated.

This section summarizes general program design and then describes the things you should consider when you design an RPG II program. Such items as the RPG II read/write cycle, the way in which RPG II handles sequential branching, conditional branching, repeated operation, and subroutines, and the use of other available AS/400 system utilities could all affect the way in which you design your RPG II program.
Figure 5. Steps in Creating an RPG Program
Steps in Designing Your Program

Designing a program includes:

- Deciding the output that you need from your program
- Deciding the processing that will produce the output you need
- Deciding the input that 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). The reason for designing the output first is to make sure that you start with a clear understanding of what your program is to do. If you know the output that you need, you can decide on the calculations that are necessary to produce that output. 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 will produce output records. You must decide what you will 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 on the information that 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. Then you can use the Screen Design Aid (SDA) utility to create your displays. For more information about SDA, see Application Development Tools: Screen Design Aid User’s Guide and Reference.

If you want to print the output records, you also have to decide on the information that you want printed (which fields from which records) and how you want it 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 still have to decide the information that 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 file description specifications and output specifications.

Designing the Processing

Designing the processing means planning the calculations that will produce the necessary output. When you design the processing, you must be aware of how the RPG program cycle works. The RPG program cycle controls certain read/write operations done on each record, so the program cycle partly determines how you can process your data. The program cycle is described in more detail under “RPG Program Cycle” on page 6.
**Designing the Input**
After you decide on the output that you need and the calculations that will produce that output, the next step is to find out where the input data for your program will come. 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 this information, you can describe your input records on the RPG input specifications.

**Structured Programming in RPG II**
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.

This section discusses these structures and how they can be implemented in RPG II.

**Sequential Operation**
Sequential operation means any series of instructions that are processed one after the other, rather than transferring control to some other part of the program.

**Conditional Branching**
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 6 is a flowchart of a conditional branch.

In RPG, the If-Then-Else structure is implemented through the operation codes IFxx, ELSE, and END. Figure 7 on page 16 shows a design for a conditional branch using the IFxx, ELSE, and END operations.
You can also use the CASxx operation to create conditional branches. You can also create a branch to a subroutine with the EXSR operation and conditioning indicators.

Repeating an Operation

RPG implements the three repeat operations—Do While, Do Until, and Do—by means of the DOWxx, DOUxx, and DO operation codes and the END operation code.


Do While

If you test the condition first and then process the operation(s), the structure is called a Do While. An example of a Do While 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 8 is a flowchart of a Do While, and Figure 9 on page 18 illustrates the coding of a Do While.

Figure 8. Flowchart of a Do While
Notice in Figure 9 (the Do While) that the program first determines if the condition is true (line 01). If it is true, the code between the DOW and the END operations is processed. Then the program goes back to line 1 to determine again if the condition is still true, and repeats the entire cycle. If the condition is no longer true, control passes to the instruction immediately following the END operation.

**Do Until**

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

1. Add 1 to a sum.
2. Compare the sum with 5.
3. Repeat steps 1 and 2 if the sum is less than 5.

Figure 10 on page 19 is a flowchart of a Do Until, and Figure 11 on page 20 illustrates the coding.
Figure 10. Flowchart of a Do Until
**Structured Programming**

<table>
<thead>
<tr>
<th>Line</th>
<th>Control</th>
<th>C</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicators</td>
<td>Factor 1</td>
<td>Factor 2</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
<td>Result Field</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>FLDA</td>
<td>DOUGT</td>
<td>FLDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td>ADD</td>
<td>INC</td>
<td>FLDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 11. Design for a Do Until Operation Using the DOUxx Operation Code**

**Do**

In its simplest form, the Do operation using the DO and the END operation codes is coded into the RPG calculation specifications as shown in Figure 13 on page 22.

This is how the Do operation works:

1. Set index field (result field) to 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 incremented 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.

Figure 12 is a flowchart of a Do operation, and Figure 13 on page 22 illustrates the coding.
Figure 12. Flowchart of a Do Operation
Figure 13. Design for a Do Operation Using the DO and END Operation Codes

Summary of Structured Programming Operation Codes

The structured programming operation codes are:

- IFxx (If/Then)
- ELSE (Else Do)
- END (End)
- DO (Do)
- DOWxx (Do While)
- DOUxx (Do Until)
- CASxx (Case).

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 running). This is valid for CASxx operation only.

See Chapter 27, “Operation Codes” for more information.
The RPG II Screen Design Aid (RPGSDA) makes it easier for you to create and change the displays your program requires. Typing the RPGSDA procedure name, and pressing the Enter key, displays the following:

```
Screen Design Aid (SDA)
Select one of the following:
1. Design menus and help text
2. Design display formats and help text
3. Build RPG II WORKSTN file specifications
4. Edit source and procedure members
5. View display formats
6. Print display formats
7. Compile display formats

Selection

F3=Exit    F12=Cancel
```

Figure 14. The RPGSDA Display

For a complete explanation of SDA, see Application Development Tools: Screen Design Aid User's Guide and Reference.
Chapter 2. Entering RPG II Specifications

After you have designed your program, you must write the individual statements that make up your source program. These statements can be coded on RPG 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.

You can also use an editor, such as the Source Entry Utility (SEU), to enter this information directly into the system and in this way begin creating your source program.

The RPG II Specifications

There are eight kinds of specifications in RPG II:

- Control specification
- File description specifications
- Extension specifications
- Line counter specifications
- Telecommunications specifications
- Input specifications
- Calculation specifications
- Output specifications.

Each of these specifications is described briefly in this chapter.

RPG programs do not have to use all specifications. A typical program uses only control, file description, input, calculation, and output specifications.

When your source program is compiled, the specifications you use must be in the order shown in Figure 15 on page 26. The specifications can be coded in any order, but normally you code them in the same order in which you design the program: first output, then calculations, then input.

For more information on the file translation specifications see “Translating a File” on page 466. For more information on collating sequence specifications see Chapter 17, “Changing the Hexadecimal Value of Characters.” For more information on coding compile-time table or array data see Chapter 14, “Using Arrays and Tables.”
Control Specification

The control specification provides the RPG compiler with the following information about your program and system:

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

For a detailed description of the control specification, see Chapter 18, “Control Specification.”
File Description Specifications

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

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

For a detailed description of the file description specifications, see Chapter 19, “File Description Specifications.”

Extension Specifications

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

- Name of the file, array, or table
- 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.

For a detailed description of the extension specifications, see Chapter 20, “Extension Specifications.”

Line Counter Specifications

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

- Number of lines per page
- Line of the page at which overflow occurs.

For a detailed description of the line counter specifications, see Chapter 21, “Line Counter Specifications.”

Telecommunications Specifications

Telecommunications specifications describe each BSCA file in the program. The information includes:

- Name of the file
- Description of the communication network used
- Type of station
- Type of control
- Type of code used
- Station identification.

For a detailed description of the telecommunications specifications, see Chapter 22, “Telecommunications Specifications.”
You must define and activate your data communications network before you can use a BSCA file. If you have System/36 RPG II programs that use binary synchronous communications, you will need to do some preparation in order to make them usable on your AS/400 system. See Chapter 11, “Using a BSCA File” on page 313 for more information.

Input Specifications

Input specifications describe the records and fields in the input files used by the program. The information for each record includes:

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

For a detailed description of the input specifications, see Chapter 23, “Input Specifications.”

Calculation Specifications

Calculation specifications describe the calculations done on the data and the order in which the calculations are processed. 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 used in the operation
- The operation done
- If resulting indicators are set after the operation is done.

For a detailed description of the calculation specifications, see Chapter 24, “Calculation Specifications.”

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

For a detailed description of the output specifications, see Chapter 25, “Output Specifications.”

**Entering Your Program into the System**

After you have written your RPG II program, you must enter it into source files in the system. You can enter the source program in two ways:

- Interactively by using the Source Entry Utility (SEU)

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

See the *Programming: Data Management Guide, SC41-9658* for 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, lowercase letters can only be used in literals, constants, comments, array data, and table data. All other information must be in uppercase letters.

**Using SEU to Enter or Change an RPG II Source Program**

SEU is a general full-screen editor that you access by typing the CL command STRSEU. SEU can be used to create or to edit any source file member. It is most useful, however, as a tool to enter source programs.

SEU has displays that you can use to enter or to edit your source code, or to specify edit services to use with a file. SEU can display special format lines and high-level language prompts that correspond to the RPG coding forms to help you type your RPG source programs accurately.

You can type specifications position by position, or you can type a specification field by field using SEU prompts.
The RPG II Syntax Checker

If you specify the TYPE(RPG36) parameter on the STRSEU command, SEU calls the RPG II syntax checker which checks each statement for RPG II syntax errors as it is entered. The RPG II syntax checker finds incorrect entries and displays error messages that allow you to correct the errors before compiling the program.

The RPG II syntax checker checks the specification lines for errors as you enter them. The functions of the RPG II syntax checker include:

- Checking each position of the specification line for valid entries
- Checking that all field, indicator, and operation code names are valid
- Checking that the proper fields are specified for each operation code (for example, the arithmetic operations must have a result field entry)
- Checking that literals are specified correctly
- Checking the compiler directives /TITLE, /SPACE, and /EJECT.

The syntax checker has some limitations:

- It does not check comment statements, nor whether records have the required 80-column record format
- It cannot detect logical or relational errors between two or more statements (for example, if no TAG exists for a GOTO operation).

These are errors that the RPG compiler detects when you compile the program.

For a complete description of how to enter or to update an RPG II source program using SEU, see the Application Development Tools: Source Entry Utility User’s Guide and Reference.

Using the RPGSEU Display

Entering the RPGSEU procedure name allows you to use SEU to create or to change an RPG II program. The display for the RPGSEU procedure is shown in Figure 16.
RPGSEU PROCEDURE

Creates or updates an RPG II or auto report source or procedure member with SEU

Name of member to be created or updated

Type of SEU member

Name of member containing SEU formats

Length of statement

Name of library containing member

F3=Exit

(C) COPYRIGHT IBM Corp. 1994

Figure 16. The RPGSEU Procedure Display

Respond to each prompt by entering the appropriate information:

Name of member to be created or updated: Enter the name of the library member to be created or changed.

Type of member: Enter R, A, or P:

  R    for an RPG source member.
  A    for an RPG source member containing auto report specifications.
  P    for a procedure member.

If no option is specified, R is assumed.

Name of member containing SEU formats: This prompt applies only to the System/36 processor. Any value that you enter will be syntax checked but not processed.

Length of statement: This prompt applies only to the System/36 processor. Any value that you enter will be syntax checked but not processed.

Name of library containing member: Enter the name of the library that contains or is to contain the member being created or changed. If no library name is specified, the name of the current library is assumed.
You can use the Help key from RPGSEU to see which function keys to use, and for additional information about parameters.

You then press Enter to begin the SEU session.
Chapter 3. Compiling an RPG II Program

There are two environments you can compile from: the AS/400 system environment, and the System/36 Environment. Consequently, there are two ways of compiling source programs and this chapter describes each one:

- Using the CL command CRTS36RPG to compile an RPG II source program in the AS/400 system or in the System/36 Environment
- Using the RPGC procedure to compile an RPG II source program in the System/36 Environment.

This chapter describes how to compile an RPG II source program, the results of a compilation, and the more common errors that may occur during compilation.

Chapter 26, “The Auto Report Feature,” contains information on compiling auto report source programs using the CL command CRTS36RPT.

Using CRTS36RPG to Compile an RPG II Source Program in the AS/400 system

When you use the CL command CRTS36RPG (Create RPG II Program) you call the RPG compiler, which then proceeds to check the syntax of your RPG source program line by line; it also examines the relationships between the lines. For example, it makes sure that all field names are defined and, if defined more than once, that each definition has the same attributes.

**Error Tracking:** If the RPG compiler stops because of errors, the escape message QRG9001 is issued. If the RPG program object stops while being run, the escape message RPG9001 is issued. You can write a CL program that uses the CL command MONMSG to anticipate this exception by monitoring for these escape messages.

**Status of Last Compile:** To help track compilation, the RPG compiler creates and updates a data area with the status of the last compile. This data area is named RETURNCODE, is 400 characters long, and is placed into library QTEMP. You can access this RETURNCODE data area by using the CL command DSPDTAARA (Display Data Area Command). See the CL Reference for more information. Each record in the data area RETURNCODE has the following format:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Content and Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Character ‘1’ means program was created</td>
</tr>
<tr>
<td>2</td>
<td>Character ‘1’ means compilation failed because of compiler errors</td>
</tr>
<tr>
<td>3</td>
<td>Character ‘1’ means compilation failed because of source errors</td>
</tr>
<tr>
<td>4</td>
<td>Character ‘1’ means compiled from source generated by auto report</td>
</tr>
<tr>
<td>5</td>
<td>Character ‘1’ means program resolution monitor was not called because *NOGEN option was selected on CRTS36RPG command</td>
</tr>
<tr>
<td>6-10</td>
<td>Number of source statements</td>
</tr>
<tr>
<td>11-12</td>
<td>Severity level from command</td>
</tr>
<tr>
<td>13-14</td>
<td>Highest severity on message diagnostic</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1994
### Using the CRTS36RPG Command

You can call the CRTS36RPG command in one of three ways:

- **Interactively from the CRTS36RPG command displays, using prompts.** You call up these displays by typing the CL command CRTS36RPG and pressing F4.
- **Entering CRTS36RPG followed by only those parameters that override the default settings.** This statement is entered on the command line interactively, or may be done as part of a batch input stream.
- **Entering CRTS36RPG followed only by the values of the parameters, but in the proper sequence.** This method is used mostly when you are submitting the compile request as part of a batch input stream, or if you are including the compile request as part of a CL program. This method may also be used interactively, but you are then limited by CL to entering only the first three parameter values.

**Note:** Any default on the CRTS36RPG command or the CRTS36RPT command can be changed using the CL command CHGCMDFT (Change Command Defaults). Refer to the *CL Reference* for more information.
Elements of the CRTS36RPG Command Line
The descriptions that follow refer to three elements of the compiler command line:

- The CL command CRTS36RPG.
- The parameter, which is usually referred to by an abbreviation such as PGM, SRCFILE, GENOPT, and so on.
- The value for the parameter. This can be either a keyword or an object name.

All object names must consist of alphabetic characters. The name cannot exceed eight characters in length. The full AS/400 system naming convention is allowed.

Entering Only Certain Parameter Values
All of the CRTS36RPG parameters have default values. You may change up to three parameter values; the rest cannot be modified. In such an instance it is quicker simply to type CRTS36RPG, followed by only those parameters whose default settings you want to override. Use spaces to separate parameters; enter values for each parameter keyword by enclosing the value or values in parentheses.

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

```
CRTS36RPG PGM(newlibrary/newname)
```

Entering Only the Parameter Values
You have the choice of entering only the parameter values, keywords, or object-names, without specifying the parameter acronyms. In this case, there is no acronym to tell the system to which parameter a value belongs, you must enter all the values in the sequence shown below. You need not enter the entire set of parameter values, but you must enter all of the parameter values up to the one that you are interested in. The system will use the default values for the remaining parameters.

For example, to compile program ABC into library QTEMP, with the source program in member ABC in source file QS36SRC in library SRCLIB, you would enter:

```
CRTS36RPG QTEMP/ABC SRCLIB/QS36SRC S[PGM
```

Notice that you also had to enter the name of the file and library containing the source program, and the default value for the SRCMBR parameter, *PGM. The system recognizes the parameter to which each value belongs by the position of the value on the compiler command line. Up to three parameters can be specified in this way. For more information see CL Reference.
Completing the First CRTS36RPG Display

Type CRTS36RPG on the OS/400 operating system, and press ENTER to bring up the first display. Press F10 to display additional parameters as shown in the following display:

```
Create RPG II Program (CRTS36RPG)
Type choices, press Enter.
Program ................. +CTLSPEC Name, +CTLSPEC
Library .................. +CURLIB Name, +CURLIB
Source file ............... QS36SRC Name, QS36SRC
Library .................. +LIBL Name, +LIBL, +CURLIB
Source member ........... +PGM Name, +PGM
Generation severity level ... 21 0-99
Never ending program .... +NO +NO, +YES
Maximum MRT devices ..... 0 0-99
Text 'description' ......... +SRCMBRTXT

Additional Parameters
Source listing options .... *SOURCE, +NOSOURCE, +SRC...
+ for more values

Generation options ......... *LIST, +NOLIST, +XREF...

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

Figure 17. First CRTS36RPG Display, Showing Valid Options

This first display shows the first set of parameters and the valid options for each parameter.

Press F11 to display the parameters and their keywords for the CRTS36RPG command as shown on the following display:

```
Create RPG II Program (CRTS36RPG)
Type choices, press Enter.
Program ................. PGM +CTLSPEC
Library .................. +CURLIB
Source file ............... SRCFILE QS36SRC
Library .................. +LIBL
Source member ........... +PGM
Generation severity level ... GENLVL 21
Never ending program .... NEP +NO
Maximum MRT devices ..... MRTMAX 0
Text 'description' ......... TEXT +SRCMBRTXT

Additional Parameters
Source listing options .... OPTION
+ for more values

Generation options ......... GENOPT

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

Figure 18. First CRTS36RPG Display, Showing Keywords
Press F11 again to return to the CRTS36RPG display showing valid options for each parameter.

Each parameter on the display has 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 the setting of a particular parameter, type a question mark (?) as the first character in that field and press Enter to receive more detailed information. The ? must be followed by a blank.

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

You must enter values for the library and program name by which the compiled program will be known, and the name of the source file that contains the source program that will be compiled.

All other parameters have default values, which you can change if necessary. Press F10 to display additional parameters, and press the Roll Up key (page down). See Figure 20 on page 41. Press F5 to return to the first display showing keywords.

In the parameter description that follows, all defaults are explained first and are underlined. The parameters are presented in sequence. Use this sequence if you are entering only the parameter values without the corresponding parameter abbreviation.

The parameters on this first display, and their possible values are:

**PGM**
Specifies the library and program name by which a compiled RPG program will be known. If no library is specified, the created program is stored in the current library. The program name must not already exist in the library.

**File**

*CTLSPEC: The program name specified in positions 75 through 80 of the control specification.

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

If a program name is not specified on the control specification, and if a member name is not specified by the actual SRCMBR name, the default program name is RPGOBJ.

*program-name: The name by which the program will be known.

**Library**

*CURLIB : The current library will be used. If you have not specified a current library, QGPL will be used.

*library-name: Enter the name of the library in which the compiled program will be stored.

**SRCFILE**
Specifies the name of the source file that contains the RPG source program to be compiled and the library in which the source file is located.
USING CRTS36RPG TO COMPILE

File

**QS36SRC**: The default source file, QS36SRC, contains the RPG II source program to be compiled.

*source-file-name*: Enter the name of the source file that contains the RPG source program to be compiled.

Library

**LIBL**: The system searches the library list to find the library in which the source file is located.

**CURLIB**: The current library will be used. If you have not specified a current library, QGPL will be used.

*library-name*: Enter the name of the library in which the source file is stored.

**SRCMBR**

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

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

*source-file-member-name*: The name of the member that contains the RPG source program.

**GENLVL**

Specifies if a program object is generated, depending on the severity of the errors found. 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 level equal to or greater than the specified value, the compilation is ended. For example, if you do not want a program generated if you have messages with a severity level of 30 or greater, specify 30 in this parameter.

**21**: Does not generate a program object if error messages produced during compilation have a severity level of 21 or greater. If the severity level is 21 or greater, the program may contain errors that could cause unpredictable results if the program were compiled and run.

*severity-level-value*: Enter a number from 0 through 99.

**Note**: The severity level value of RPG messages does not exceed 50.

**NEP**

Specifies if this is a never-ending program. A never-ending program is one that uses system resources (such as disk storage, display stations, or printers) that are not shared with other programs.

**NO**: This is not a never-ending program.

**YES**: This is a never-ending program.

**MRTMAX**

Specifies the maximum number of requesting devices that can use a single copy of the program at the same time. A value of 0 indicates that the program
allows only one requester (it is a SRT program). Values of 1 to 99 indicate that the program allows one or more requesters (it is a MRT program).

0: This is a SRT program.

maximum-terminals: Enter a number from 0 through 99, to show the maximum number of requesters (display stations or ICF sessions) for this program.

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

*SRCMBRTXT: The text of the source member is used.

*BLANK: No text appears.

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

OPTION
Specifies the options to use when the source program is compiled. Any or all of the following keyword values can be specified in any order. Separate the keywords with a delimiter (usually a blank space).

*SOURCE: Produce a source listing, consisting of the RPG program input and all compile-time errors.

*NOSOURCE: A source listing is not produced.

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

*XREF: Produce a cross-reference listing and key field information table (when appropriate) for the source program. For an example of a cross-reference listing, see “Cross-Reference Listing” on page 50.

*NOXREF: A cross-reference listing is not produced.

*NOXREF is the default when either *NOSOURCE or *NOSRC is specified.

*GEN: Create 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.

*SECLVL: Print second-level message text on the line following the first-level message text.

*CONSOLE: Generate a display file for a CONSOLE file.
**NOCONSOLE**: Do not generate a display file for a CONSOLE file.

**GENOPT**
Specifies the options used to create the program object: the printing of the IRP (intermediate representation of a program), a cross-reference listing of objects defined in the IRP, an attribute listing from the IRP, and the program template. GENOPT also reserves a program patch area, and specifies the optimization of a program for more efficient running. These results may be useful if a problem occurs when trying to run the compiled program. Any or all of the following values can be specified in any order. Separate the keywords with a delimiter (usually a blank space).

**NOLIST**: Do not produce a listing of the IRP.

**LIST**: Produce a listing of the IRP, its associated hexadecimal code and any error messages.

**NOXREF**: Do not print a cross-reference listing for the IRP.

**XREF**: Print a cross-reference listing for the IRP.

**NOATR**: Do not produce a listing of the attributes for the IRP source program.

**ATR**: Produce an attribute listing for the IRP source program.

The listing includes field descriptions and statement numbers on which the fields are defined.

**NODUMP**: Do not dump the program template when an error occurs.

**DUMP**: Dump the program template when an error occurs.

**NOPATCH**: Do not reserve a program patch area in the compiled program.

**PATCH**: Reserve a program patch area in the compiled program.

This area can be used for debugging purposes; its size is based on the size of the generated program.

**NOOPTIMIZE**: Do not process program optimization.

**OPTIMIZE**: Process program optimization. With *OPTIMIZE the compiler generates a program for more efficient processing, one that will possibly require less storage. Specifying *OPTIMIZE, however, substantially increases the time required to create the program. Existing program objects may be optimized with the CL command CHGPGM.
Completing the Second CRTS36RPG Display

Press F10 to display additional parameters at the bottom of the first display. Then press the Roll Up key (page down) to see the following display:

![CRTS36RPG Display](image)

Figure 19. Second CRTS36RPG Display, Showing Valid Options

This second display shows the second set of parameters and the valid options for each parameter.

Press F11 to display the parameters and their keywords for the CRTS36RPG command as shown in the following display:

![CRTS36RPG Display](image)

Figure 20. Second CRTS36RPG Display, Showing Keywords
Press F11 again to return to the CRTS36RPG display showing valid options for each parameter.

The additional parameters and their descriptions are listed below.

**PRTFILE**
Specifies the name of the file in which the compiler listing is placed and the library in which the file is located.

**File**
- **QSYSPR**: If a file name is not specified, the compiler listing is placed in the IBM-supplied file QSYSPR. The file QSYSPR has a record length of 132.
- **file-name**: Enter the name of the file in which the compiler listing is placed. If you specify a file whose record length is less than 132, information will be lost.

**Library**
- **LIBL**: The system searches the library list to find the library.
- **CURLIB**: The current library will be used. If you have not specified a current library, QGPL will be used.

- **library-name**: Enter the name of the library in which the file is located.

**ICFLIB**
Specifies the library containing the Intersystem Communications Function (ICF) record format definitions used with the program.

- **ICF-library-name**: Enter the name of the library that contains the ICF files.

**REPLACE**
Specifies if a new program object is created without first deleting any existing object.

- **YES**: A new program object is created and any existing object of the same name in the specified library is moved to library QRPLOBJ.
- **NO**: A new program object will not be created if an object of the same name already exists in the specified library.

**TGTRLS**
Specifies that the program object is generated to run with the current release or the previous release of the OS/400 operating system. Only one release of downward compatibility will be provided. A program can be modified and compiled on one system at the current release and distributed for use on another system at the previous release. The user cannot use a new function in a program on the current release and compile the program on a previous release.

- **CURRENT**: The current version of the compiler is used to compile the source program and to generate an object to run on the current release of the OS/400 operating system. If a save with TGTRLS(*PRV) is done to a program compiled with TGTRLS(*CURRENT), the save operation will fail.
- **PRV**: The previous version of the compiler is used to compile the source program and to generate an object to run on the previous release of the OS/400 operating system. The user must also save the object with *PRV to run it on the previous release. The previous and current versions of the compilers must be installed to use this option. Only program objects that have
USING CRTS36RPG TO COMPILE

been compiled on the current release with TGTRLS(*CURRENT) need to be recompiled with TGTRLS(*PRV) to be restored onto the previous release.

USRPRF
Specifies the user profile under which the compiled RPG program runs. This profile controls the objects that can be used by the program (including the authority that the program has for each object).

*USER: The program runs under the user profile of the program’s user.

*OWNER: The program runs under the user profiles both of the program’s owner and of the program’s 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 while the program is running are owned by the program’s user.

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

When using the command:

crtrpgpgm pgm(xxxx/yyyy) usrprf(*owner)

the system creates the RPG program but does not change the owner if the object already exists and was created with the default usrprf(*user). Therefore, the USRPRF parameter will not be updated.

AUT
Specifies the authority that is granted to the public for the program and its description. The authority can be altered after program creation for all or for specified users with the CL command GRTOBJAUT (Grant Object Authority), RVKOBJAUT (Revoke Object Authority) or EDTOBJAUT (Edit Object Authority).

(For further information on these commands, see the CL Reference.)

*CHANGE: The public has only operational rights for the compiled program. Any user can run and debug the program but cannot change it.

*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: The 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 Security Concepts and Planning.

Note: Use the AUT parameter to reflect the security requirements of your installation. The security facilities available on the AS/400 system are described in detail in the Security Concepts and Planning.

PHSTRC
Specifies if information about compiler phases is provided on the listing. A phase trace consists of the names of the compiler phases being printed on the compiler listing in the order that they run. The number of RXT messages (such as compiler headings) are also listed as they are retrieved.

*NO: Do not provide information about compiler phases.

*YES Provide information about compiler phases.
USING CRTS36RPG TO COMPILE

ITDUMP
Cause intermediate text produced after the compilation of one or more specified phases of the source program to be listed dynamically. Dynamic listing means that the intermediate text is printed, during compilation, as each intermediate text record is being built. This parameter also states if a flow is to be printed of the major routines run in one or more specified phases.

*NONE: Do not produce an intermediate text dump.

phase-name: Enter the fourth and fifth characters of each phase name.

As many as 25 phases, each phase identified by the fourth and fifth characters of its name, can be specified. Enclose the list in parentheses. For example, to list dynamically intermediate text produced by QRGSES36, QRGOS36, and QRGSCS36, enter the fourth and fifth characters of each phase name:

ITDUMP(SESOSC)

SNPDUMP
Produces a listing of major data areas and intermediate text (snap dump) following the compilation of one or more specified phases of the source program.

*NONE: Do not produce a snap dump.

phase-name: Enter the fourth and fifth characters of each phase name.

As many as 25 phases, each phase identified by the fourth and fifth characters of its name, can be specified. Enclose the list in parentheses. For example, to list major data areas and intermediate text following the processing of phases QRGGCS36, QRGSCS36, and QRGOS36, enter the fourth and fifth characters of each phase name:

SNPDUMP(SISCSO)

CODELIST
Causes a dynamic listing of the IRP produced by a specific phase. Dynamic listing means that the intermediate text is printed, during compilation, while the specified phase runs.

*NONE: Do not produce a dynamic listing of IRP.

*ALL: A dynamic listing of IRP is produced for each of the code-generating phases processed.

phase-name: Enter the fourth and fifth characters of each phase name.

As many as 25 phases, each phase identified by the fourth and fifth characters of its name, can be specified on the CODELIST parameter. For example, to list dynamically the IRP produced by phases QRGGCS36, QRGGOS36, QRGEC36, enter the fourth and fifth characters of each phase name:

CODELIST(GCGOEC)
FIXDECDTA
Specifies if bad decimal data is fixed or an error is signaled.

*YES: Fix bad data. Digit values in the range A through F are set to zero; invalid signs are set to plus.

*NO: Interpret bad decimal data and notify the program. Message RPG9078 is issued. Use this option when you are developing a new program, where decimal data errors are often caused by program logic errors.

Using RPGC to Compile an RPG II Source Program in the System/36 Environment

The RPGC procedure compiles an RPG source program from the System/36 Environment. The RPGC procedure has two displays.

Calling RPGC

To begin the RPGC procedure, use any one of the following methods:

- Type HELP RPGC (with or without parameters) on the command line, and press Enter.
- Type RPGC on the command line of the display, and press Enter.
- Type RPGC (with or without parameters) on the command line, and press F4.
- Type RPGC, a space, and the source member name on the command line, and press Enter.

Using the First RPGC Display

The first display for the RPGC procedure looks like this:

<table>
<thead>
<tr>
<th>RPGC PROCEDURE</th>
<th>Optional-* Ignored-%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles an RPG II source program</td>
<td></td>
</tr>
</tbody>
</table>

| Name of source program to be compiled   | ____                  |
| Name of library containing source program | #LIBRARY              |
| Create diagnosed source member          | DSM, NODSM            |
| Output option for compiler listings      | PRINT, NOPRINT, CRT   |
| Create cross-reference listing           | XREF, NOXREF          |
| Maximum number of requesting display stations | 0-99 0_               |
| Never-ending program                    | NEP, NONEP            |
| Name of library to contain compiled program | ____ *               |

Press Enter for more options
F3=Exit  F6=Put on job queue  F17=Procedure complete
(C) COPYRIGHT IBM Corp. 1994

Figure 21. First RPGC PROCEDURE Display, Showing Valid Options

Respond to each prompt by entering the appropriate information.

Name of source program to be compiled: Type the name of your source program.
Name of library containing source program: Type the name of the library that contains the source member to be compiled. If no library name is specified, the current library is assumed.

Create diagnosed source member: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

Output option for compiler listings: Type PRINT, NOPRINT, or CRT.

PRINT Print the compiler listing of the source program.

NOPRINT Do not print the compiler listing of the source program.

CRT Display the compiler listing at the display station.

If no option is specified, PRINT is assumed.

Create cross-reference listing: Type NOXREF or XREF.

NOXREF Do not create a cross-reference listing for the program.

XREF Create a cross-reference listing for the program. The cross-reference listing is created only if the program is not ended by errors. The cross-reference listing is part of the compiler listing, so the cross-reference listing is displayed or printed depending on your response to the preceding prompt.

If no option is specified, NOXREF is assumed.

Maximum number of requesting display stations: Type the number (0 through 99) of display stations that can use a single copy of the program at the same time. If no number is entered, a value of 0 is assumed (the program is not a MRT program). A value of 0 means that the program is a SRT program. Values of 1 to 99 specify MRT programs.

Never-ending program: Type NONEP or NEP.

NONEP Will not be a never-ending program.

NEP Will be a never-ending program. A never-ending program is one that uses system resources (such as disk storage, display stations, or printers) that are not shared with other programs. Use this option if your program will be requested frequently.

If no option is specified, NONEP is assumed.

Name of library to contain compiled program: Type the name of the library that is to contain the compiled program. If this option is not specified, the source input library is assumed.

You can use the Help key from the first RPGC Procedure display to see which function keys to use, and for additional information about parameters.
Using the Second RPGC Display

The second RPGC display looks like this:

```
RPGC PROCEDURE

RPG XX,#LIBRARY,NODSM,PRINT,NOREF,0,NONEP,,

Override print option in source . . . . . SOURCE,PSOURCE,NOSOURCE
Override debug option in source . . . . . DEBUG,NODEBUG
Override size-to-execute option in source . . . 2-64
Halt on serious program error . . . . . . NOHALT,HALT
Replace duplicate members . . . . . . . . REPLACE,NOREPLAC
Create program that can be run . . . . . . . LINK,NOLINK
Create program that must be link-edited . . . . NOOBJECT,OBJECT
Name of subroutine input library . . . . . . . GEN
Generate CONSOLE file display formats . . . . . GENERATION
Size of work files in blocks . . . . . . . . . . . . 1-9999
Name of data dictionary to be used . . . . . .
Create program with memory resident overlays . . . . . . . NOMRO,MRO

F6=Put on job queue   F12=Cancel   (C) COPYRIGHT IBM Corp. 1994
```

**Figure 22. Second RPGC PROCEDURE Display, Showing Valid Options**

Respond to each prompt by entering the appropriate information.

*Override print option in source:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Override debug option in source:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Override size-to-execute option in source:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Halt on serious program error:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Replace duplicate members:* Type REPLACE or NOREPLAC.

**REPLACE**

If a program object is being created, and if a program object with the same name already exists in the output library, you want the newly compiled program to replace the existing program object.

**NOREPLAC**

If a program object is being created, and if a program object with the same name already exists in the output library, you want an error message to be displayed.

If no option is specified, REPLACE is assumed.

*Create program that can be run:* Type LINK or NOLINK.

**LINK**

Creates a program that can be run.
NOLINK  Does not create a program that can be run.

*Create program that must be link-edited:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Name of subroutine input library:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Generate CONSOLE file display formats:* Type GEN or NOGEN.

**GEN**  Creates and compiles source specifications for 24-line, 1920-character display formats for a CONSOLE file. The specifications are created and compiled only if the program contains no terminal errors.

**NOGEN**  Does not create or compile the source specifications for the display formats for a CONSOLE file.

If no option is specified, GEN is assumed. The procedure ignores this option if your program does not use a CONSOLE file.

*Size of work files in blocks:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

*Name of the data dictionary to be used:* Enter the name of the library that contains the ICF file to be used during the compilation, if you are using user-defined communication formats. On a System/36, these formats were located in a data dictionary.

*Create program with memory-resident overlays:* This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

You can use the Help key from the second RPGC Procedure display to see which function keys to use, and for additional information about the parameters.

---

**Using the SEU Split Display to Correct Errors**

SEU allows you to look at a compiler listing that is on an output queue. The SEU split display can have both edit and browse sections. It can show records from the source program on the top part and records from the spooled compiler listing on the bottom part of the display. Using this display, you can:

- Scan the listing for errors
- Correct the source statements that have errors.

For complete information on using the SEU split display, see the *SEU User’s Guide and Reference*.

The display depicted in Figure 23 on page 49 shows SEU’s split display. The top section shows an RPG II source program in edit mode. The bottom section is a compiler listing in browse mode, indicating where the errors in the source program are located.
Printing an RPG Cross-Reference Listing (RPGX Procedure) in System/36 Environment

The RPGX procedure prints a cross-reference listing for an RPG source program that can be run. Diagnostic checking is provided with the RPGX procedure.

To start the RPGX procedure, use any one of the following methods:

- Type HELP RPGX (with or without parameters) on the command line, and press Enter.
- Type RPGX on the command line of the display, and press Enter.
- Type RPGX (with or without parameters) on the command line, and press F4.
- Type RPGX, a space, and the source member name on the command line, and press Enter.
The display for the RPGX procedure looks like this:

```
RPGX PROCEDURE

  Creates a cross-reference Listing for an RPG II source program

Name of source program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ________
Size of $SOURCE file in blocks . . . . . . . . . . . . . 1-9999 40__ __ __ __ __ __ __ __ __ __
Name of library containing source program . . . . . . . . . . . #LIBRARY

F3=Exit    F6=Put on job queue   (C) COPYRIGHT IBM CORP.1994
```

**Figure 24. The RPGX PROCEDURE Display, Printing an RPG Cross-Reference Listing**

Respond to the prompts by entering the appropriate information.

**Name of source program:** Type the name of your source program.

**Size of $SOURCE file in blocks:** This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

**Name of library containing source program:** Type the name of the library that contains the source program to be listed. If no library name is specified, the name of the current library is assumed.

You can use the Help key from the RPGX Procedure display to see which function keys to use, and for additional information about the parameters.

**Cross-Reference Listing**

The RPGX procedure or the XREF option in the CRTS36RPGR, RPGC, and AUTOC procedures provides a cross-reference listing of the symbols defined and referenced in the respective RPG and auto report source programs. The auto report cross-reference listing can be very helpful when you are modifying or expanding your program. Running the cross-reference listing step in the CRTS36RPGR, RPGC, or AUTOC procedure depends upon the listing being provided only when XREF is specified for the RPGC or AUTOC procedure. The default is no cross-reference listing (NOXREF), for the RPGC and the AUTOC procedures, and cross-reference listing (XREF) for the CRTS36RPGR procedure.

The symbols used in an RPG or auto report program are sorted and placed in the following categories in the cross-reference listing:

- File and record names
- Fields, data structures, labels, literals, tables, and arrays
- Indicators.
Sample Cross-Reference Listing
The information that is printed in the cross-reference listing for each symbol type is shown in Figure 25.

Cross Reference

1 File and Record References:

<table>
<thead>
<tr>
<th>FILE/RCD</th>
<th>DEV/RCD</th>
<th>REFERENCES (D=DEFINED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 DFB01</td>
<td>DISK</td>
<td>2000 600 1500</td>
</tr>
</tbody>
</table>

2 Field References:

<table>
<thead>
<tr>
<th>FIELD</th>
<th>ATTR</th>
<th>REFERENCES (M=MODIFIED D=DEFINED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(10)</td>
<td>A(1)</td>
<td>3000</td>
</tr>
<tr>
<td>A,I</td>
<td></td>
<td>800</td>
</tr>
<tr>
<td>EXIT</td>
<td>TAG</td>
<td>700 13000</td>
</tr>
<tr>
<td>FIELD</td>
<td>Z(2,0)</td>
<td>400D 600 800 1000M 1100</td>
</tr>
<tr>
<td>I</td>
<td>TAG</td>
<td>500D 1200</td>
</tr>
<tr>
<td>1</td>
<td>LITERAL</td>
<td>400 1000</td>
</tr>
<tr>
<td>10</td>
<td>LITERAL</td>
<td>1100</td>
</tr>
</tbody>
</table>

3 Indicator References:

<table>
<thead>
<tr>
<th>INDICATOR</th>
<th>REFERENCES (M=MODIFIED D=DEFINED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>1400M</td>
</tr>
<tr>
<td>99</td>
<td>600M 700 1100M 1200</td>
</tr>
</tbody>
</table>

* * * * * END OF CROSS REFERENCE * * * * *

Figure 25. Sample Cross-Reference Listing

The information in the cross-reference listing is described below.

1 File and Record References

FILE/RCD is the file or record name. In the above example the file name is DFB01.

DEV/RCD is the device or record type. In the above example the device type is DISK.

REFERENCES lists the line numbers in which the files or records are referenced. A reference with a D beside it means that the file or record is defined in this reference.

2 Field References

FIELD is the field, data structure, label, literal, table, or array name.

ATTR defines the field attributes; if the field is alphanumerical (A), zoned (Z), or packed (P), the length and number of decimal positions are enclosed by parentheses.

REFERENCES lists the line numbers in which the fields are referenced. A reference with an M beside it means that the field was last modified in this reference. A reference with a D beside it means that the field is defined in this reference.
Interpreting a Compiler Listing

After your program is compiled, RPG can provide you with a compiler listing. This listing contains the source program that was compiled, as well as other information such as errors, data dumps, cross-references, which can be useful in debugging a program. You specify the listing options when you call up the compiler. If you compile using CRTS36RPG or CRTS36RPT, you use the OPTION parameter to request a listing. If you compile using RPGC or AUTOC, you would type PRINT next to the Output option for compiler listings: prompt to request a listing.

Figure 26 is an example of a compiler listing. The item numbers refer to the numbers on that figure.

<table>
<thead>
<tr>
<th>Command Options:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program . . . . . . . . . . . : EXAMPLES/TEST</td>
</tr>
<tr>
<td>Source file . . . . . . . . . : EXAMPLES/QS36SRC</td>
</tr>
<tr>
<td>Source member . . . . . . . . : PGM</td>
</tr>
<tr>
<td>Source listing options . . . . : +SOURCE +XREF +GEN +NODUMP +NOSECLVL</td>
</tr>
<tr>
<td>Generation options . . . . . . : +NOLIST +NOXREF +NOATR +NODUMP +NOOPTIMIZE</td>
</tr>
<tr>
<td>Generation severity level . . . : 21</td>
</tr>
<tr>
<td>Print file . . . . . . . . . : +LIBL/QSYSPRT</td>
</tr>
<tr>
<td>Never ending program . . . . . : *NO</td>
</tr>
<tr>
<td>Maximum MRT devices . . . . . : 0</td>
</tr>
<tr>
<td>Library for ICF files . . . . :</td>
</tr>
<tr>
<td>Replace program . . . . . . . : +YES</td>
</tr>
<tr>
<td>User profile . . . . . . . . : +USER</td>
</tr>
<tr>
<td>Authority . . . . . . . . . : +CHANGE</td>
</tr>
<tr>
<td>Text . . . . . . . . . . : +SRCMBRTXT</td>
</tr>
<tr>
<td>Phase trace . . . . . . . . : +NO</td>
</tr>
<tr>
<td>Intermediate text dump . . . . : +NONE</td>
</tr>
<tr>
<td>Snap dump . . . . . . . . . : +NONE</td>
</tr>
<tr>
<td>Codelist . . . . . . . . . : +NONE</td>
</tr>
<tr>
<td>Fix decimal data . . . . . . . : +YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actual Program Source:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member . . . . . . . . . : TEST</td>
</tr>
<tr>
<td>File . . . . . . . . . . : QS36SRC</td>
</tr>
<tr>
<td>Library . . . . . . . . : EXAMPLES</td>
</tr>
<tr>
<td>Last Change . . . . . . . : 11/16/88 16:08:10</td>
</tr>
<tr>
<td>Description . . . . . . : TEST WORKSTN FILE</td>
</tr>
</tbody>
</table>

Figure 26 (Part 1 of 7). Compiler Listing
SEQUENCE NUMBER: *...1......2......3......4......5......6......7......*

Source Listing  

100 H  01/08/94 TEST
200 H++ PROGRAM NAME : TEST  01/18/94 *
400 H++ *
500 H++

600 FWORKSTN CP 1920 WORKSTN
700 F KID WSID
800 E A 1 6 35
900 INWORKSTN NS 02 1 CS 2 C1
1000 I NS 01 1 CS 2 C2
1100 I NS 01

1200 C*  
1300 C SETOF 98 1
1400 C 03 WSID REL WORKSTN 97 2
1500 C KG SETON 99 1
1600 C KG GOTO END
1700 C 15 SETON 16 1
1800 C 15 GOTO END
1900 C U1 GOTO NEXT5
2000 C NB1 GOTO NEXT

2100 C*  
2200 C* TEST RETURN CODE 0 WITH LEVEL 1 MESSAGE:
2300 C MOVE MIC 40
2400 C MOVE 1 LEVEL 10
2500 C*
2600 C EXIT SUBR23
2700 C*
2800 C RLABL MIC
2900 C RLABL TEXT1 75
3000 C RLABL LEVEL
3100 C RLABL RTNCOD 10
3200 C Z-ADD1 X 10
3300 C ADD RTNCOD X
3400 C RTNCOD COMP 0 98 3
3500 C N98 SETON 99 1
3600 C GOTO END
3700 C*  
3800 C NEXT TAG
3900 C*
4000 C 10 GOTO NEXT1
4100 C*  
4200 C* TEST RETURN CODE 0 WITH LEVEL 2 MESSAGE:
4300 C SETON 10 1
4400 C*  
4500 C MOVE 0002 MIC
4600 C MOVE 2 LEVEL
4700 C MOVE $BLANKS TEXT1
4800 C*  
4900 C EXIT SUBR23
5000 C*

Figure 26 (Part 2 of 7). Compiler Listing
SEQUENCE NUMBER  */....*/....2....*/....3....*/....4....*/....5....*/....6....*/....7....* USE NUM UPDATE LINE ID

5100 C RLABL MIC
5200 C RLABL TEXT2 225
5300 C RLABL LEVEL
5400 C RLABL RTNCOD
5500 C Z-ADD1 X
5600 C ADD RTNCOD X
5700 C RTNCOD COMP 0 98 3
5800 C N98 SETON 99 1
5900 C GOTO END
6000 C*
6100 C NEXT1 TAG
6200 C*
6300 C 11 GOTO NEXT2
6400 C*
6500 C* TEST RETURN CODE 1 RETRIEVED WITH TRUNCATION:
6600 C SETON 11 1
6700 C*
6800 C MOVE 0001 MIC
6900 C MOVE 1 LEVEL
7000 C*
7100 C EXIT SUBR23
7200 C*
7300 C RLABL MIC
7400 C RLABL TEXT3 50
7500 C RLABL LEVEL
7600 C RLABL RTNCOD
7700 C Z-ADD1 X
7800 C ADD RTNCOD X
7900 C RTNCOD COMP 1 98 3
8000 C N98 SETON 99 1
8100 C GOTO END
8200 C*
8300 C NEXT2 TAG
8400 C*
8500 C* TEST RETURN CODE 2 MESSAGE NOT FOUND:
8600 C 1Z GOTO NEXT3
8700 C*
8800 C SETON 12 1
8900 C*
9000 C MOVE 0002 MIC
9100 C MOVE 1 LEVEL
9200 C*
9300 C EXIT SUBR23
9400 C*
9500 C RLABL MIC
9600 C RLABL TEXT1
9700 C RLABL LEVEL
9800 C RLABL RTNCOD
9900 C Z-ADD1 X
10000 C ADD RTNCOD X

Figure 26 (Part 3 of 7). Compiler Listing
Figure 26 (Part 4 of 7). Compiler Listing
SEQUENCE IND DD LAST PAGE PROGRAM
NUMBER 15000 C+ SYSTEM WILL NOT CREATE LOAD MEMBER WHEN MESSAGE TEXT LENGTH EXCEEDS THE LEVEL 1 MAXIMUM.

15400 C+ TEST RETURN CODE 5 MEMBER NOT FOUND:
15430 C OR SETON 15 1
15700 C NU1 SETON 16 1
15800 C NU1 GOTO END
15900 C+
16000 C MOVE 0001 MIC
16100 C MOVE 1 LEVEL
16200 C+
16300 C EXIT SUBR23
16400 C+
16500 C RLABL MIC
16600 C RLABL TEXT1
16700 C RLABL LEVEL
16800 C RLABL RTNCOD
16900 C 2-AUD1 X
17000 C ADD RTNCOD X
17100 C RTNCOD COMP 5 98 3
17200 C N98 SETON 99 1
17300 C+
17400 C END TAG
17500 C+
17600 C 03 SETON LR 1
17700 C+
17800 OWORKSTN D 01
17900 0 OR 02 12N13
18000 0 OR 02 13N14
18100 0 OR 02 14N15
18200 0 OR 02 15N16
18300 0 KB 'RGGS8001'
18400 0 MIC
18500 0 LEVEL
18600 0 RTNCOD
18700 0 A,X
18800 0 TEXT1
18900 0 D 02 10N11
19000 0 KB 'RGGS8001'
19100 0 MIC
19200 0 LEVEL
19300 0 RTNCOD
19400 0 A,X
19500 0 TEXT1
19600 0 TEXT2
19700 0 D 02 11N12
19800 0 KB 'RGGS8001'
19900 0 MIC
20000 0 LEVEL
20100 0 RTNCOD
20200 0 A,X
20300 0 TEXT3
20400 0 D 02 16
20500 0 OR KG
20600 0 KB 'RGGS8002'
20700 0 23 ' TEST WAS SUCCESSFUL '
20800 0 99 23 ' TEST FAILED '

Figure 26 (Part 5 of 7). Compiler Listing
Additional Diagnostic Messages

Table/Array ..........: A
MESSAGE FOUND
MESSAGE TRUNCATED
MESSAGE NOT FOUND
LEVEL INVALID
MIC INVALID
MEMBER MISSING OR TEXT > LEV 1 MAX

Table of End Position Offsets for Fields Described Using Position Notation.

Cross Reference

File and Record References:

Field References:

Figure 26 (Part 6 of 7). Compiler Listing
Figure 26 (Part 7 of 7). Compiler Listing

1 The compiler heading line contains the compiler release number, the current date, the current time and the page number. The dates on the listing are in the format specified by the job attribute date format (DATFMT).

2 The Command Options section lists most of the parameters and options you specify with the CRTS36RPG or CRTS36RPT command. If you specify an option, the value is listed; if you take the default, the default value is listed.
Chapter 3. Compiling an RPG II Program

The **Actual Program Source** section lists:
- The source member name
- The source file name
- The library that contains the source file
- The date and time of the last change to the source member
- The source member text, if specified.

The consecutive position rule (positions 6 through 74) helps you to determine the position of any character in the source record or output listing.

The **Ind Use** (Indicator Usage) column helps you to determine the resulting indicator(s) that you have specified in calculation specifications:
- 1 indicates an indicator is specified in positions 54 and 55
- 2 indicates an indicator is specified in positions 56 and 57
- 3 indicates an indicator is specified in positions 58 and 59.

The entries in the **Do Num** (Do Number) column indicate the nested level of a statement in a DO group or an IFxx/END or IFxx/ELSE/END group. The B (Beginning) prefix indicates the beginning statement in a DO group or an IFxx/END or IFxx/ELSE/END group. The E (End) prefix indicates an end statement in a DO group or an IFxx/END or IFxx/ELSE/END group.

The entry in the **Last Update** column indicates the date that the source statement (for an RPG II program, not an auto report) was last changed. The job attribute values for date format (DATFMT) and for separator character (DATSEP) are used.

The **Page Line** column indicates the consecutive page and line numbers of the statements on the source listing. The page and consecutive line numbers correspond to the entries in positions 1 through 5 of the RPG specifications.

The **Program ID** column indicates the program name (positions 75 through 80 of each source record).

Statement 10100 shows how syntax and relational errors are indicated on the source listing. An "*" in the leftmost position of the source statement followed by an error message number indicates that a syntax error has been noted. For easy identification, asterisks are also placed immediately under the positions in error. For example, error number 5023 has occurred in positions 33 through 42. The error numbers and their explanations are listed in the Cross Reference and Message Summary sections (see 14 and 15).

The **Compile-Time Tables** section lists the tables and arrays.

The end positions for the fields are described using position notation.

The **Cross Reference** listing allows easy determination of where files, fields, and indicators are defined, referenced, and modified. For example, the field LEVEL, Z(1,0), is a zoned field of one position with no decimal positions and is defined on the line that has the sequence number 2400.

This line shows how undefined and unreferenced files, fields, and indicators are indicated. An "*" in the leftmost position of the cross-reference statement is followed by an error message number that indicates that an undefined or unreferenced file, field, or indicator error has occurred. In this example, the indicator 97 is referenced. The error message numbers and their explanations are listed in the Message Summary section.
The **Message Summary** section contains a list of all message numbers for all errors that have occurred in the program, along with the severity level, the number of times each message occurred, and a detailed message description for each error.

The **Final Summary** section contains a list of the number of messages at each severity level. Remember that the severity level value of RPG messages does not exceed 50. This section also lists the source program totals, such as:

- The number of source records
- The number of RPG specifications
- The number of table records
- The number of comment statements.

**Note:** /EJECT and /COPY statements are not counted under specifications, although they are counted under source records.
Chapter 4. Testing and Debugging an RPG Program

RPG and OS/400 provide functions that you can use to test and debug the programs you develop.

- OS/400 provides:
  - Breakpoints
  - Traces.

- RPG provides:
  - DEBUG operation code.

RPG DEBUG can be used independently of the OS/400 functions, or in combination with them, to debug a program.

Externally described files are not allowed with the DEBUG operation.

You can also produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables, by responding to a run-time message.

This chapter describes how to use these features to test and debug an RPG program.

Using, Displaying, and Printing Messages

The following section contains a short description of an RPG II message and information on displaying and printing messages on the AS/400 system. For more information on messages see CL Programmer's Guide.

Using Messages

This manual refers to messages you receive during run time or compile time.

These messages fall into two basic categories:

- **Error messages** inform you of an error situation found as the system was processing your command.

- **Informational messages** provide you with information about the running of the program and/or its result.

These messages are shown on your display or printed on your compiler listing. There are no message manuals for this product.
Each compiler message contains a minimum of three parts as illustrated below:

<table>
<thead>
<tr>
<th>A</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Syntax of Program-Identification entry is invalid. Defaults to RPGOBJ.</td>
</tr>
<tr>
<td>C</td>
<td>Cause: The Program-Identification entry (positions 75-80) of a control specification has an invalid syntax: the first character is not alphabetic or it is not left-justified, or it contains embedded blanks or special characters. Defaults to RPGOBJ.</td>
</tr>
<tr>
<td></td>
<td>Recovery: Specify RPGOBJ or a valid entry (positions 75-80) for the Program-Identification option. Recompile.</td>
</tr>
</tbody>
</table>

Figure 27. Sample Compiler Error Message

A A number indicating the seriousness of the error. Numbers 40 and above are reserved for run-time errors.

Severity Meaning

00 An informational message displayed during entering, compiling, and running a program. This level is used to convey information to the user. No error has been detected and no corrective action is necessary.

10 A warning message displayed during entering, compiling, and running a program. This level indicates that an error was 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 compilation. This level indicates that an error was made, but that the compiler is taking a recovery that might yield the desired code. The program may not work as the author intended.

30 A severe error message displayed during compilation. This level indicates that an error too serious for automatic recovery was detected. Compilation is completed, but running the program cannot be attempted.

40 An abnormal end of program or function message displayed during running. This level indicates an error that forces cancellation of processing. The operation may have ended because it was unable to handle valid data, or possibly because the user cancelled it.

50 An abnormal end of job message displayed during running. This level indicates an error that forces cancellation of job. The job may have ended because a function failed to perform as required, or possibly because the user cancelled 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 that you see online or on a listing. This text is a brief description of the problem, generally one sentence.
The text you see online when you press F4 from the display with the first-level text. This text will be printed on your listing if you specify *SECLVL in your compile-time options. The IBM-supplied default for this option is *NOSECLVL. This text contains an expanded description of the message and a section detailing the correct user response.

Displaying and Printing Messages
To display or print a particular message or messages, use the DSPMSG or DSPMSGD commands. These commands are described in the CL Reference. The library name for your product is #RPGLIB, and the file name of the compiler messages file is QRPG2MSG. Run-time messages are in a file called QRPG2MSGE, which is in the system library QSSP.

Note: If you have any comments or suggestions concerning the messages, please use the reader comment form included with this manual and send it to us.

Running an RPG Program
There are three ways to run an RPG program:

- Enter the Operation Control Language (OCL) LOAD and RUN statements from the display station keyboard. If the program uses DISK files, you must also include an OCL FILE statement for each DISK file. You can use the OCL SWITCH statement to set any external indicators (U1 through U8) used by the program. To attach a display station to a program that uses a WORKSTN file, you can use the OCL WORKSTN statement.
- Enter the name of a procedure that contains the required OCL statements.
- Select a menu option.

For a complete explanation of the Operation Control Language statements and of how to write a procedure, see the System Reference for the System/36 Environment.

Example of OCL Statements Used to Run a Program
The following OCL statements load and run an RPG program object named PROG1 that uses an input DISK file named INPUT and an output DISK file named OUTPUT:

```
// LOAD PROG1
// FILE NAME-INPUT
// FILE NAME-OUTPUT,BLOCKS-10
// RUN
```

RPG Halt Messages
Errors in an RPG program can cause the program to halt while it is being compiled or run. When the program halts, a halt message is displayed. If the program is run from the input job queue, the halt messages are sent to the system operator message queue QSYSOPR. Using the CL command CHGMSGQ (described in the CL Reference), you can put the QSYSOPR queue in break mode. This causes those messages to appear on a display automatically. If you put the QSYSOPR queue in break mode, only the terminal that is in break mode can be used to respond to messages. If the program is run from a display station (and not placed on the input job queue), the halt messages are displayed at the display station. If
the program is a Multiple Requester Terminal (MRT) program, the messages go to
the message queue QSYSOPR.

When a halt message is displayed, the person at the display station must respond
by entering one of the following options:

- **0**: Control is returned to the program, and processing continues.
- **1**: The remainder of the program cycle is bypassed, and the next record is
  read. For some messages, option 1 means that you should try the operation
  again. This information is in the second-level text for these messages.
- **2**: End-of-job operations specified by the program are done, tables are dumped,
  and file labels are cataloged.
- **3**: The job is canceled, but control is not returned to the RPG program. New
data entered for this job is not preserved.

### Using Breakpoints

A breakpoint is a statement number or a label in your program where you want the
program to stop running. If you use a statement number, it can be a sequence
number that appears on the compiler listing of the RPG source 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 program cycle (for example, *TOTC indi-
  cates the beginning of total calculations and *TOTL indicates the beginning of
total output)
- Associated with a function done by your RPG program (for example, .SQRT
  indicates the square root function).

When a breakpoint is about to run for an interactive job, the system displays the
breakpoint at which the program has stopped and, if requested, the values of
program variables. After you get this information (in a display) you can go to a
Command Entry display and then 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 about to run for a batch job, a breakpoint program can be
called. You must create this breakpoint program to handle the breakpoint informa-
tion.

### Example of Using Breakpoints

Figure 28 shows a portion of a sample RPG program, DBGPGM, and the CL com-
mands that add breakpoints at statements 1200 and 1500.
USING BREAKPOINTS

Source Listing

100 FTESTX IF F 5 DISK 01/03/94
200 FTESTA UF F 10 DISK 01/03/94

300 ITESTX NS 01 01/03/94
400 I 1 5 PART 01/03/94
500 ITESTA NS 02 01/03/94
600 I 1 5 FLD1 01/03/94
700 *************************************************************** 01/22/94
800 * MAINLINE 01/22/94
900 *************************************************************** 01/22/94

1000 C LOOP TAG 01/20/94
1100 C READ TESTX 66 3 01/03/94
1200 C 66 GOTO ENDPGM 01/20/94
1300 C READ TESTA 67 3 01/20/94
1400 C N67 MOVE PART FLD1 01/20/94
1500 C N67 EXCPMAST 01/20/94
1600 C N66 GOTO LOOP 01/20/94
1700 C ENDPGM TAG 01/20/94
1800 C SETON LR 1 01/20/94

1900 OTESTA E MAST 01/03/94
2000 O FLD1 5 01/03/94

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

Figure 28. Example Source Listing for Program DBGPGM

The following CL commands will display the values of the variable *IN when the breakpoint at statement 1200 is reached and the values of the variables FLD1 and PART when the breakpoint at statement 1500 is reached.

STRDBG PGM(EXAMPLES/DBGPGM)
ADDBKP STMT(1200) PGMVAR((IN))
ADDBKP STMT(1500) PGMVAR((FLD1) (PART))

All CL commands are explained in the CL Reference.
The following is displayed as a result of reaching the first breakpoint.

```
Display Breakpoint
Statement/Instruction . . . . . . . . . . : 1200 /004A
Program . . . . . . . . . . . . . . . . : DBGPGM
Invocation 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'
11 '0' '0' '0' '0' '0' '0' '0' '0' '0'
21 '0' '0' '0' '0' '0' '0' '0' '0' '0'
31 '0' '0' '0' '0' '0' '0' '0' '0' '0'

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

**Figure 29. Part One of First Breakpoint Display**

The continuation character + indicates that you need to press the Roll key to scroll to part two.

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

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

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

**Figure 30. Part Two of First Breakpoint Display**
The following is displayed as a result of reaching the second breakpoint.

<table>
<thead>
<tr>
<th>Display Breakpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement/Instruction: 1500 /0060</td>
</tr>
<tr>
<td>Program: DBGPGM</td>
</tr>
<tr>
<td>Invocation level: 1</td>
</tr>
<tr>
<td>Start position: 1</td>
</tr>
<tr>
<td>Format: +HEX</td>
</tr>
<tr>
<td>Length: +DCL</td>
</tr>
<tr>
<td>Variable: FLD1</td>
</tr>
<tr>
<td>Type: CHARACTER</td>
</tr>
<tr>
<td>Length: 5</td>
</tr>
<tr>
<td>Variable: PART</td>
</tr>
<tr>
<td>Type: CHARACTER</td>
</tr>
<tr>
<td>Length: 5</td>
</tr>
</tbody>
</table>

Press Enter to continue.

F3=Exit program F10=Command entry

Figure 31. Part One of Second Breakpoint Display

The continuation character + indicates that you need to press the Roll key to scroll to part two.

Figure 32. Part Two of Second Breakpoint Display
At this point you could change the value of one of these variables to alter how your
program runs. Press F10 to get to the command entry display. You can use the
CL command CHGPGMVAR (Change Program Variable) to change the value of a
variable. This command is explained in the CL Reference.

To leave DEBUG mode, enter ENDDBG.

Considerations for Using Breakpoints

The following characteristics of breakpoints should be known before breakpoints
are used:

- If a breakpoint is part of a conditional statement, that breakpoint request is
  processed even if the conditions are not met.
- If a breakpoint is bypassed by a branching 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 RLABL, PLIST,
  PARM, and DEFN operations.
- When a breakpoint is requested for a statement, the breakpoint occurs before
  that statement is processed.
- When a breakpoint is requested for a statement that is not processed, such as
  a TAG operation, the breakpoint is set on in the next statement that will run.
- Breakpoint functions are specified with CL commands.
- Input fields not used in your program cannot be specified in the PGMVAR
  parameter of the debug commands. However, the entire input or output buffer
  for a record may be displayed 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 the input buffer for the second file in your program may be displayed by
  specifying PGMVAR (ZZ02BIN).

Breakpoint functions include adding breakpoints to programs, displaying breakpoint
information, removing breakpoints from programs, and resuming running a program
after a breakpoint has been displayed. Refer to the CL Reference for descriptions
of these commands. Refer to the CL Programmer's Guide for a further description
of breakpoints.

Using a Trace

A trace is a record of the statements that were processed in a program. Optionally,
you can also show the values of variables used in the statements.

A trace is different from a breakpoint. A breakpoint marks a statement where you
want to stop the program. A trace merely records that the statement was reached
and processed. In a trace, you must request a display of the recorded trace
information. The display shows the sequence in which the statements were processed,
and, optionally, the values of variables used in the statements.

You specify those statements and variables that you want traced. Also, you are
able to request that variables be traced only when their values change. You can
specify a trace of one statement in a program, a number of statements in a program, or an entire program.

Example of Using a Trace
The following example shows a trace of the sample RPG program, DBGPGM (see Figure 28 on page 65), and the CL commands that add a trace of statements 1000 through 1800 in that program.

The CL commands to add the trace would be:

```
STRDBG PGM(EXAMPLES/DBGPGM)
ADDTRC STMT((1000 1800))
```

Following is an example of a listing of the traced information. The OS/400 command to display this information would be:

```
DSPTRCDTA OUTPUT(*PRINT) CLEAR(*YES)
```

DSPTRCDTA and all other CL commands are explained in the CL Reference.

Considerations When Using a Trace
You should know the following characteristics of traces before you try to trace an RPG program:

- A conditioned statement is recorded in the trace even if the conditions are not met.
- Statements bypassed by branching operations are not included in the trace.
- Trace functions are specified by using 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.

Refer to the CL Programmer’s Guide for a further description of traces.
Using an RPG Formatted Dump

To obtain an RPG formatted dump while running a program, type a F in response to any RPG II run-time message. The dump is normally written to the job output queue.

Figure 34 shows an example of the RPG formatted dump produced by the following code:

```
S      1     2     3     4     5     6     7
H      OLExED..CDYI....S.....I...1.F.H.........T.............*
H      1

*     1     2     3     4     5     6     7   *
FFilenameIPEAFBlenRlenLKI0vKlocEDevice++++Exit++.....A...U+.*
FTSTF2  CD     80     WORKSTN
F*

*     1     2     3     4     5     6     7   *
IFilenameSqNORiPos1NCCPos2NCCPos3NCC............................*
ITSTF2  NS     01
I *

*     1     2     3     4     5     6     7   *
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C       READ TSTF2    77
C       SETON        LR
C*

*     1     2     3     4     5     6     7   *
OName+++DFBASbSaN01N02N03Excptn................................
OTSTF2  D
0           K5 'TTTTT'
0           5 'TTTT'
0*
```

The formatted dump includes field contents, data structure contents, array and table contents.
Figure 34 (Part 1 of 5). RPG II Formatted Dump
Figure 34 (Part 2 of 5). RPG II Formatted Dump
Figure 34 (Part 3 of 5). RPG II Formatted Dump
USING AN RPG FORMATTED DUMP

Figure 34 (Part 4 of 5). RPG II Formatted Dump
**Using an RPG Formatted Dump**

---

**Figure 34 (Part 5 of 5). RPG II Formatted Dump**

- **A** Qualified program name and library.
- **B** RPG source statement in error.
- **C** RPG routine in which the exception/error occurred.
- **D** CPF or MCH for a machine exception.
- **E** Machine instruction number.
- **F** Information about the last file used in the program before an exception/error occurred. *Last File Status* is included for WORKSTN files only.
- **G** Program information.

---

**Chapter 4. Testing and Debugging an RPG Program**

---

**75**
Using the DEBUG Operation Code

You can put one or more DEBUG operation codes among your RPG calculations to help you debug a program that is not working properly. Whenever the DEBUG operation is processed, one or two types of 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 started; the second type of record is optional and shows the contents of the result fields specified for the DEBUG operation. Refer to the description of the DEBUG operation code in Chapter 27, “Operation Codes” for a description of the information that is provided for DEBUG.

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.

Considerations for using the DEBUG operation code are:

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

You can apply the OS/400 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.

Factor 1 of the DEBUG operation can contain a literal or the name of a field. The literal or the contents of the specified field are written in the first record. If factor 1 is left blank, the statement number of the DEBUG operation is written in the first record.

Factor 2 must contain the name of the output file to which the records are written. The file cannot be a WORKSTN file. The same file name must be used as factor 2 for all DEBUG statements in a program.
The result field can contain the name of a field or array whose contents are written in the second output record. If the result field is left blank, only one record is written when that DEBUG operation occurs.

To use the DEBUG operation, you must also code a 1 in column 15 of the control specification. If you leave that column blank, the DEBUG operation is treated as a comment.

**Records Written by the DEBUG Operation**

The DEBUG operation always writes at least one record. That record has the following format:

<table>
<thead>
<tr>
<th>Column Positions in the Output Record</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DEBUG =</td>
</tr>
<tr>
<td>9-18</td>
<td>Statement number of the DEBUG operation code in the program</td>
</tr>
<tr>
<td>19-26</td>
<td>Literal or contents of field coded in factor 1 (optional)</td>
</tr>
<tr>
<td>27-28</td>
<td>Blanks</td>
</tr>
<tr>
<td>29-44</td>
<td>INDICATORS ON =</td>
</tr>
<tr>
<td>45 - any position</td>
<td>The names of all indicators that are on, each separated by a blank. More than one record may be needed.</td>
</tr>
<tr>
<td>(depending on length of output record)</td>
<td></td>
</tr>
</tbody>
</table>

The second record is written only when an entry is coded in the result field. The second record has the following format:

<table>
<thead>
<tr>
<th>Output Positions</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-14</td>
<td>FIELD VALUE =</td>
</tr>
<tr>
<td>15 - any position</td>
<td>The contents of the result field (up to 256 characters). If the result field is an array, more than one output record may be needed to contain the array.</td>
</tr>
<tr>
<td>(depending on length of field)</td>
<td></td>
</tr>
</tbody>
</table>

**Debugging a Program That Uses a WORKSTN File**

Because the logic for WORKSTN file processing is supplied by both the RPG program and the display format specifications, it may be more difficult to find coding errors for the WORKSTN file than for other files. The following techniques may help you debug a WORKSTN program:

- Always compare the $SFGR listing to the RPG input and output specifications. The From, To, and End positions used on the RPG specifications should normally match the From, To, and End positions listed for the $SFGR input and output buffers.
- If the wrong format is displayed, check the status of the indicators to be certain that the status is as you expected. If the status of the indicators is incorrect, the wrong format may be displayed or a correct format may be followed by an additional format that overlays and thereby hides the correct format. The speci-
debugging a program that uses a workstn file

Modification of erase input (columns 31 and 32 of the S specifications) or override fields (columns 33 and 34 of the S specifications) may also display a partial format that overlays the correct format.

- Always include a record type for blank records. Blank records can occur in one of two ways:
  - If the record is the first input record for a display station (in most programs the first input record for a display station is blank)
  - If N (no) is specified in column 22 (return input) of the display format S specification and no data keys were pressed.

- If the program goes to end of job prematurely, check if all display stations have been released or if Y (yes) was specified in column 35 (suppress input) of the S specification. Either situation can result in no display stations being allowed to enter input, which causes end of file on the WORKSTN file and the job will end prematurely.

- If the command display unexpectedly follows a program display, the program may have gone to end of job before any data was entered for the display (see the RESTORE parameter of the OCL WORKSTN statement in the System Reference for the System/36 Environment). If RESTORE-NO is specified, the information from the program may be on the display after the program has gone to end of job, so it appears as if the program is still running. If RESTORE-YES is specified, the command display appears on the display immediately when the program goes to end of job.

- Avoid using multiple formats on the same section of the display until the program logic is debugged.

- During the debugging operations, display a constant on the display for every format. This should help you analyze the display contents.

- Use the DEBUG operation code in selected locations to trace the program flow. Suggested locations and the resulting debug information are as follows:

<table>
<thead>
<tr>
<th>Location</th>
<th>Debug Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>As first calculation</td>
<td>Shows the contents of the specified input record and the indicator status for a primary file</td>
</tr>
<tr>
<td>After a READ operation</td>
<td>Shows the contents of the specified input record and the indicator status for a demand file</td>
</tr>
<tr>
<td>Before an EXCPT operation</td>
<td>Shows the status of the indicators that control the records (formats) being produced as output</td>
</tr>
<tr>
<td>As last detail calculation</td>
<td>Shows the indicator status before heading and detail output</td>
</tr>
<tr>
<td>After an ACQ operation</td>
<td>Shows the work station ID or session ID and the indicator off if the operation was successful</td>
</tr>
<tr>
<td>After a REL operation</td>
<td>Shows the display stations or communications sessions that are released from the program</td>
</tr>
<tr>
<td>After a TAG operation</td>
<td>Shows the program flow</td>
</tr>
<tr>
<td>As first statement in each subroutine</td>
<td>Shows the program flow</td>
</tr>
<tr>
<td>Conditioned by LR</td>
<td>Shows when the program ends</td>
</tr>
</tbody>
</table>
After each WORKSTN output record, define a record with the same conditioning indicators and write that record to the DEBUG file (see Figure 35 on page 79). The record should contain:

- The format name
- The work station ID, if used in the program
- The release status, if the display station is released in the output specifications
- SLN (starting line number), if used in the program
- Data fields as needed.

If the following types of error messages occur, check the probable causes listed:

- Error messages involving program checks to the WORKSTN device are probably caused by:
  - Invalid use of erase input fields (columns 31 and 32 of the S specification)
  - Clearing all or a portion of the display containing the input fields.

- Error messages involving invalid WORKSTN identifiers are probably caused by an earlier release of the display station in either calculation or output operations.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn...................................................*
0*
0* Use the same conditioning indicators (01 and 02) for both files.
0* Indicator 02 shows release status of display station.
0*
OWK   D   01
0          ORR   02
0..................N01N02N03Field+YBEnd+PConstant/editword+++++++++++...*
0                 K8 'FORMATXX'
0*
0* Use the same file (TEST) that is used for DEBUG operations.
0*
OTEST   D   01
0          OR   02
0              8 'FORMATXX'
0  ID     11
0              20 'RELEASED'

Figure 35. Writing the WORKSTN Output Record to the DEBUG File
Chapter 5. The Detailed RPG II Program Cycle

The RPG program cycle was introduced in Chapter 1, “An Introduction to RPG II and the AS/400 System.” This chapter presents the RPG program cycle in more detail.

Figure 36 on page 82 shows the steps in the RPG program cycle in more detail. Steps 1 and 2 are for the first record only. The program cycle, which occurs for each record read, begins with step 3. The program cycle continues through step 26; however, the program may branch to steps out of the actual cycle when specified indicators are set on or off, or certain conditions are met.
3. Set overflow indicator on if overflow line is reached.

4. Perform first-page (1P) output.

5. Open all files.


7. Get external indicators and display station local data area, if specified.

8. Set off record-identifying indicators and 1P, L1 through L9, and H1 through H9 indicators.


10. Set on or off resulting indicators.

11. Perform first-page (1P) output.


13. Perform L0 through L9 and LR calculations, also EXCPT, CHAIN, KEY, and READ, if requested.

14. Perform fetch overflow if required by exception output.

15. Perform L0 through L9 and LR output.

16. Set overflow indicator on if overflow line is reached as a result of exception output.

17. Control break.

18. Set on appropriate control-level indicators.

19. First program cycle or first control break.

20. Perform LR indicator on.

21. LR indicator on.

22. L0 through L9 indicator on.

23. KEYBORD primary file or no primary file specified.

24. KEYBORD primary file or no primary file specified.

25. End of job requested.

26. gateway to previous cycle or during detail output of current cycle.

27. Set on LR, L1 through L9.

28. Start.
For processing of WORKSTN input files, see Figure 7-10 in Chapter 7, "Using a WORKSTN File."
The following steps describe in detail what the RPG program does at each step in
the detailed program cycle. The steps are the same as those shown in Figure 36
on page 82.

1. The program reads in the external indicators and the display station local data
   area, if specified, and opens all data files that will be used; that is, the files are
   prepared to be processed. Before the first program cycle, data structures are
   blanked, and prerun-time arrays and tables are loaded.

2. The program writes all output conditioned by the first-page indicator (1P). This
   output is written only once for each job and is not part of the program cycle
   (steps 3 through 26).

3. The program writes all headings and detail output whose conditions are satis-
   fied. This output includes specifications that are conditioned by the overflow
   indicator if the overflow routine has been fetched.

4. The program determines if the overflow line was reached during detail calcu-
   lations in the previous cycle or when heading and detail records were written in
   the current cycle. If the printed output from the program reaches the overflow
   line, the overflow indicator is set on. Otherwise, the indicator is set off unless
   the overflow routine was fetched in step 3.

5. The program tests the halt indicators. If the halt indicators are off, the program
   branches to step 6. If the halt indicators are on, the program stops once for
   each halt indicator that is on. Every time the program stops, you select one of
   three options:
       a. Continue (the program returns to step 5 to test for other halt indicators)
       b. Controlled cancel (the program branches to step 35)
       c. Immediate cancel (the program branches to step 35)

6. The program sets off all record-identifying indicators and indicators with the
   two-character entries 1P, L1 through L9, and H1 through H9.

7. The program determines if the last-record indicator (LR) is on. If it is on, the
   program branches to step 27.

8. The program determines if KEYBORD is specified as the device for the primary
   file, or if no primary file is specified. For either condition, the program branches
   to step 18.

9. The program reads (and translates, if necessary) the next input record. At the
   beginning of processing, one record from each input file (except forced files,
   CHAIN files, full-procedural files, and DEMAND files) is read. If the file has
   look-ahead fields, the file is read only on the first cycle. After that, only records
   with look-ahead fields are identified. If this is a WORKSTN file and the SAVDS
   or IND option is specified, the common SAVDS or IND area is moved to the
   active display station’s SAVDS or IND hold area. The next record is accepted,
   and the current display station’s SAVDS or IND area is moved from its hold
   area to the common SAVDS or IND area.

10. The program tests to determine if the record is an end-of-file record. If it is an
    end-of-file record, the program branches to step 12.
11. If the record is not an end-of-file record, the program determines if the input
records are in the order specified on the input specifications sheet. If the order
is incorrect, the program branches to step 33. The program also branches to
step 33 if input records are not specified in order and the record cannot be
identified.

12. If end-of-job conditions have been met, the program branches to step 27. All
files for which an E is specified in position 17 of the file description specifica-
tions must be at end of file.

13. When more than one input file is used, the program must select the next record
to process and must branch to step 28.

14. If there is only one input file, no record selection is needed. The program
determines if sequence checking is requested. If so, the program branches to
step 31.

15. The program sets on the record-identifying indicator specified for the current
record type. Data from the current record type is not available for processing
until step 25.

16. If the record contains control fields, the program determines if a control break
has occurred. (A control break occurs when the contents of the control field
are not equal to the contents of the previously stored control field.) If a control
break has not occurred or if control fields are not specified, the program
branches to step 18.

17. If a control break has occurred, the program sets on the control-level indicator
showing the condition. All lower control-level indicators are also set on.

18. The program determines if the total time calculations and total time output
should be done. If no control-level indicators are specified on the input specifici-
cations, the totals are bypassed only on the first cycle. If control-level indica-
tors are specified on the input specifications, totals are bypassed until after the
first record containing control fields is processed. Totals are always processed
when the last-record indicator (LR) is on.

19. The program does all calculations conditioned by control-level indicators (in
positions 7 and 8 of the calculation specifications) and sets resulting indicators
on or off as specified. If the last-record indicator (LR) is on, calculations condi-
tioned by LR are done after other total calculations. File translation, if speci-
fied, is done for exception output and for CHAIN, READ, READE, and READP
operations. Fetch overflow is done if it is required by exception output. If the
overflow line has been reached because of the exception output, the overflow
indicator is set on.

20. The program writes all total output that is not conditioned by an overflow indi-
cator. The program determines if an overflow condition has occurred. If an
overflow condition has occurred at any time during this cycle, the overflow indi-
cator is set on. If the last-record indicator is on, output conditioned by LR is
written after other total output. File translation, if specified, is done for total
output. Fetch overflow is done if required.

21. The program determines if the last-record indicator is on. If the indicator is on,
the program branches to step 38.
22. The program determines if any overflow indicators are on. If no overflow indicators are on, the program branches to step 24.

23. The program does all output operations conditioned by a positive overflow indicator (no N before the indicator). File translation, if specified, is done for overflow output.

24. The program sets on the matching-record (MR) indicator if this is a job that processes more than one input file and if the record processed is a matching record. Otherwise, the matching-record indicator is set off.

25. The program sets the field indicators on or off as specified. Data from the last record read and from specified look-ahead fields is made available for processing. For a WORKSTN file only, the program sets off function-key indicators (KA through KN, KP through KY). If you press a command key for the WORKSTN file being processed, the program sets that function-key indicator on.

26. The program does any calculations not conditioned by control-level indicators (in columns 7 and 8 of the calculation specifications), and sets resulting indicators on or off as specified. The program translates files for exception output and for CHAIN, READ, READE, READP, and FORCE operations if specified. Fetch overflow is done if it is required by exception output. If the overflow line is passed because of the exception output, the program sets the overflow indicator on. Processing continues with step 3.

27. The program sets on the last-record (LR) indicator and all control-level indicators (L1 through L9), and processing continues with step 19.

28. If a file was forced or if NEXT was specified, the program selects the next record in that file for processing, and the program branches to step 15.

29. If a record with no match fields is found in a normal input file that is not at end of file, the program selects the record for processing, and the program branches to step 15.

30. When match fields are specified, the program selects the normal file with the highest priority matching record field. If two or more files have equal and highest priority matching record fields, the highest priority file is selected. (The primary file has the highest file priority, the first specified secondary file is next, and so forth.)

31. The program compares the match field value with the match field value of the last record. If it is in sequence, the record is accepted, and processing continues with step 15.

32. The program stops because a file with match fields is out of order. The operator’s options, indicated in step 34, are to bypass the record (read the next record from the same file) or to cancel the job.

33. The program stops because a record type was out of order or because a record was unidentified.

34. The program tests the operator’s decision either to bypass the record that caused the error condition (branch to step 4) or to cancel the job.
35. If the operator chooses to end the job by a controlled cancel, the program does steps 36 through 40. If the operator chooses an immediate cancel, the job ends.

36. The program does all calculation operations conditioned by the last-record (LR) indicator.

37. The program does all output operations conditioned by the last-record (LR) indicators.

38. The program writes any tables or arrays for which a filename entry is specified on the extension specifications (columns 19 through 26). Output tables or arrays are translated, if necessary.

39. The program closes all the files it used and writes the external indicators and display station local data area, if specified.

40. End of job occurs.
Chapter 6. Using a DISK File

A DISK file is a file that contains data read from a disk or written to a disk. A DISK file can be organized in one of three ways:

- Sequential
- Direct
- Indexed.

The DISK file can also be a distributed data management (DDM) file.

This chapter explains how to code RPG specifications so that you can create, read, update, and add records to each of these kinds of DISK files.

SEQUENTIAL FILES

In a sequential file, the position of a record depends on the order in which records are placed in the file. The first record placed in the file occupies the first record position in the file, the second record placed in the file occupies the second record position, and so on. Figure 37 shows how a sequential file is organized.

Records are stored on disk in the same order in which they are read. No index is kept, and no spaces are left between disk records.

Figure 37. Organization of a Sequential File
Creating a Sequential File

To create a sequential file, you define the DISK file as an output file and write records to the file. The OCL statements for the program must include a FILE statement. That FILE statement must use either the RECORDS parameter or the BLOCKS parameter to specify the size of the file, and it must use the DFILE-YES parameter if you want to allow records to be deleted from the file. For information about the OCL FILE statement, see the *System Reference for the System/36 Environment*.

Define the file by using the unshaded portions of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain O to indicate that the file is an output file.

Column 19 must contain F or blank to indicate that all records in the file must be the same length.

Columns 20 through 23 must contain blanks or the block length. The block length must equal the record length or be a multiple of the record length. The maximum block length is 9999. If you leave these columns blank, the block length equals the record length.

Columns 24 through 27 must contain the length of the record you are creating. The record length can be any number from 1 to 9999.

Column 32 can contain a number from 1 through 9 to indicate that the program uses two input/output areas for the file, or a blank to indicate that the program uses only one input/output area.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8, to condition the use of this file.
Example of Creating a Sequential File

Suppose you want to create a customer file on disk. Customer numbers are sequential; that is, you assign each new customer the next higher number. Figure 38 shows how to code the file description, input, and output specifications to create this sequential file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FINFILE IP F 96 96 DISK
FSEQDISK O F 256 128 DISK
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
INFILE 'zerodot11 2'zerodot 1 C2
I 2 7 CUSTNO
I 15 34 NAME
I 35 54 ADDR
I 55 74 CITST
I 8 74 DATA2
I 'zerodot21 3'zerodot 1 C3
I 2 7 CUSTNO
I 8 47 DATA3
```

Figure 38 (Part 1 of 2). Creating a Sequential Customer File
READING A SEQUENTIAL FILE

Since both input record types are needed to write a DISK record, we don’t want to write it until input record type 3 is processed. Indicator 30 specifies that the DISK record is written after input record 3 is processed.

**OSEQDISK D 30**

CM is added to the DISK record. This code is a record code that can be used to identify a customer master record in other programs.

<table>
<thead>
<tr>
<th>Field</th>
<th>Constant/editword</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 'CM'</td>
<td></td>
</tr>
<tr>
<td>CUSTNO</td>
<td>8</td>
</tr>
<tr>
<td>DATA2</td>
<td>75</td>
</tr>
<tr>
<td>DATA3</td>
<td>115</td>
</tr>
</tbody>
</table>

*Figure 38 (Part 2 of 2). Creating a Sequential Customer File*

---

**Reading a Sequential File**

Sequential files can be read in any of the following ways:

- Consecutively
- Randomly by relative record number
- Randomly by relative record number and/or consecutively
- Randomly by address output (addrout) file.

**Note:** An alternative index file can also be created for a sequential file to provide another method of reading the records in the file. For information about creating an alternative index file, see “Creating an Alternative Index File for an Indexed File” on page 137.
Reading Consecutively

Reading consecutively means reading records in the order in which they occur in the file. If you want to read all the records in the file, code the file for consecutive processing as shown in the file description specifications below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P, S, T, or D:

- If you code P (primary) or S (secondary), the file is read as part of the RPG program cycle. For an explanation of how primary and secondary files are read, see Chapter 12, “Primary/Secondary/Multifile Processing.”

- If you code T (table), column 39 must contain E. Your program must also include an extension specification for a prerun-time table or array. The file name on that extension specification must be the same as the file name on this file description specification. For information about extension specifications, see Chapter 20, “Extension Specifications.”

- If you code D (demand), you must code a READ operation code in the calculation specifications in order to read the file. For information about the READ operation, see Chapter 27, “Operation Codes.”

Column 17 can contain E or blank if column 16 contains P or S. E indicates that the program must process all records from the file before the program can end. Blank indicates that the program can end whether or not all records from the file are processed.

Column 18 can contain A, D, or blank if column 16 contains P or S:

- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records in the file are in descending sequence.
- Blank indicates that the program does not check the sequence of records in the file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.
COLUMN 32 can contain a number from 1 through 9 or a blank. A number indicates that the program uses two input/output areas. Blank indicates that the program uses only one input/output area.

COLUMN 39 must contain E if column 16 contains T.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

**Reading Randomly by Relative Record Number**

Sometimes you want to read only some of the records in the file. Reading consecutively can be slow in this case, because reading consecutively means reading every record in a file. It would be faster to read only the records that you specifically identify. Reading only specific records is called random processing.

One way to identify the records to be read is to identify the position of each record in the file. A number that identifies the position of each record relative to the beginning of the file is called a relative record number. For example, the relative record number of the first record in a file is 1, the relative record number of the second record in the file is 2, and so on.

You can process files randomly by relative record number if the files are chained files (that is, if there is a C in column 16 of the file description specifications). Chained files are not read at input time of the RPG program cycle. Instead, they are read only when the CHAIN operation occurs during the calculation part of the cycle. For information about the CHAIN operation code, see Chapter 27, “Operation Codes.” Chained records can be read during total calculations or during detail calculations.

If you want to read records randomly by relative record number, code the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Column 7</th>
<th>Column 8</th>
<th>Column 9</th>
<th>Column 10</th>
<th>Column 11</th>
<th>Column 12</th>
<th>Column 13</th>
<th>Column 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>C</td>
<td>F</td>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain C to indicate that the file is a chained file. You must code a CHAIN operation in the calculation specifications in order to read a chained file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.
Columns 24 through 27 must contain the record length.

Column 28 must contain R to indicate that the file is processed randomly by relative record number.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

**Reading Randomly by Relative Record Number and/or Consecutively**

If you want to read a file both randomly and consecutively, use a full-procedural file. You can read a full-procedural file randomly like a chained file and/or consecutively like a demand file. That is, you can chain to a specific relative record number in the file and then read records consecutively from that point. To read the file randomly, you use a CHAIN operation in the calculation specifications; to read it consecutively, you use a READ or READP operation. You cannot use a READE operation to read the file consecutively, because the READE operation cannot read by relative record number. For example, if you code a CHAIN operation to relative record number 10 and then code a READ operation, the program chains to relative record number 10 and then reads the following record.

It is not necessary to code both a CHAIN and a READ or READP operation, but you must code at least one CHAIN, one READ, or one READP operation in order to read a full-procedural file. For information about the CHAIN, READ, and READP operation codes, see Chapter 27, “Operation Codes.”

Code a full-procedural file as an input file. Code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain F to indicate that the file is a full-procedural file. You must code a CHAIN, READ, or READP operation in the calculation specifications in order to read a full-procedural file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.
Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

### Reading Randomly by Address Output (Addrout) File

An **address output (addrout) file** is a record address file produced by a sort program. (A **record address file** is an input file that tells the program the records that are to be read from a DISK file and the order in which to read them.) An address output file contains the relative record numbers of the records in a DISK file. The advantages of an address output file are that:

- The space required for the address output file is much less than the space required for a sorted sequential file.
- The sort runs much faster.
- The original file is unchanged.

You can have only one address output file in a program. When an RPG program uses an address output file, it automatically reads the relative record numbers consecutively from the address output file. You do not have to code a READ operation for the address output file. Then, using the relative record number, the program randomly reads the DISK file to process the corresponding record. In this way, the program can process a sequential DISK file in a new sequence without actually sorting the records and creating a new file. Also, once the file description and extension specifications are coded for the DISK file and for its associated address output file, you can code the DISK file as an ordinary sequential file. If the DISK file is a full-procedural file, you must code a READ operation in the calculation specifications; you cannot use a READE, READP, or CHAIN operation to read a full-procedural file randomly by an address output file. No input specifications are required for the address output file.

If you want to read records randomly by an address output file, code the file description specifications for the sequential file as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>File Designation</th>
<th>Mode of Processing</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>Name of Label Exit</th>
<th>Extent Exit for DAM</th>
<th>Extent Exit</th>
<th>Number of Extents</th>
<th>Type of File</th>
<th>Number of Tracks for Cylinder Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>F</td>
<td></td>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P, S, or F:

- If you code P (primary) or S (secondary), the record is read as part of the RPG program cycle. For an explanation of how primary and secondary records are read, see Chapter 12, “Primary/Secondary/Multifile Processing.”
If you code F (full-procedural), you must code a READ operation in the calculation specifications. CHAIN, READE, and READP operations are not allowed with the address output files.

Column 18 can contain A, D, or blank if column 16 contains P or S:
- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records in the file are in descending sequence.
- Blank indicates that the program does not check the sequence of records in the file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain R if column 16 contains P or S. The R indicates that the file is processed randomly by an address output file.

Column 31 must contain I if column 16 contains P or S. The I indicates that relative record numbers from the address output file are used to process the file.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

For the address output file, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain R to indicate that the file is a record address file.

Column 17 must contain E or blank. E indicates that all records from the file must be processed before the program can end. Blank indicates that the program can end whether or not all the records from the file are processed.

Column 19 must contain F or blank.
UPDATING A SEQUENTIAL FILE

Columns 20 through 23 must contain the block length or blanks.

Column 27 must contain 3 because each record in an address output file is a relative record number, which is always three positions long.

Column 30 must contain 3 because relative record numbers in address output files are always three positions long.

Column 31 must contain I to indicate that relative record numbers are used in processing.

Column 32 must contain T to indicate that the file is an address output file.

Column 39 must contain E to indicate that the file is further described on extension specifications.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Two entries are required on the extension specifications:

Columns 11 through 18 must contain the name of the address output file. This name must be the same one coded on the file description specifications for the address output file.

Columns 19 through 26 must contain the name of the sequential file to be processed by the address output file. The name must be the same one coded on the file description specifications for the sequential file.

Updating a Sequential File

Updating records in a sequential file involves reading a record, changing some data in the record, and writing the record back to its original location in the file. If you try to update a record that was not the last record read, error message RPG9043 [Program name] tried to update a record before reading it from file [filename], is displayed. The fields updated must be described on both the input and the output specifications.

When you update records in a sequential file, the file can be processed in any of the following ways:

- Consecutively
- Randomly by relative record number
- Randomly by relative record number and/or consecutively

98  System/36-Compatible RPG II User’s Guide and Reference
Deleting Records from a Sequential File

Updating a file can include deleting records from the file. To allow deletion of records from the file, the DFILE-YES parameter must be specified on the OCL FILE statement when the file is created. For information about the OCL FILE statement, see System Reference for the System/36 Environment. If you try to delete a record from a file that does not allow deletions, error message RPG9067 Invalid operation attempted to [file name], is displayed.

To delete a record, you first read the record (either randomly or consecutively) and then, with DEL coded in the output specifications, write the record back to the same file. Code entries in the unshaded columns of the output specifications as shown below:

| Columns 1 through 6 must contain the name of the file if this is the first record on the output specifications or if the previous record on the output specifications is for a different file. |
| Column 15 must contain D, T, or E to indicate if the record is to be written at detail, total, or exception output time of the RPG program cycle. |
| Columns 16 through 18 must contain DEL to indicate that the record is deleted. |
| Columns 23 through 31 can contain output indicators. |
| Columns 32 through 37 can contain an EXCPT name if column 15 contains E. |
| Records are not physically removed from a file when they are deleted. Instead, deleted records are filled with hexadecimal FFs. That is, all the bits for every character in the deleted record are set on. |

Updating Consecutively

You can update records in a sequential file consecutively. If the file is a primary or secondary file (P or S in column 16 of the file description specifications), the program reads a record from the file at input time in the RPG program cycle, and the program writes a record to update the file during detail output or exception output time in the program cycle. If the file is a demand file (D in column 16), the program reads a record when a READ operation occurs in the calculation specifications, and it writes a record to update the file at detail output, total output, or exception output time in the program cycle.
Code the file description specifications as shown below:

| File Type | Mode of Processing | Length of Key Field or Record Address Field | Device | Device | S/F | Storage Index | Type of File | Number of Extents | File Condition | Extent Exit for DAM | Number of Tracks | Extents Exit for Cylinder Overflow | Number of Records | Form Type | Record Length | Key Field | Extension Code | Image | Key Field | Overflow Indicator | Form Type | Key Field | Extension Code | Image | Key Field | Overflow Indicator |
|-----------|--------------------|-------------------------------------------|--------|--------|----|----------------|-------------|----------------|---------------|----------------|----------------|----------------|-------------------------------|----------------|-----------|----------------|-----------|---------------|--------|-----------|------------------|-----------|-----------|---------------|--------|-----------|---------------|--------|-----------|---------------|
| F         |                    |                                           | Disk   |        |    |                |              |                |               |                |                |                |                               |                |           |               |          |               |        |           |                  |          |           |               |        |           |               |
| 02        | F                  |                                           |        |        |    |                |              |                |               |                |                |                |                               |                |           |               |          |               |        |           |                  |          |           |               |        |           |               |
| 03        | F                  |                                           |        |        |    |                |              |                |               |                |                |                |                               |                |           |               |          |               |        |           |                  |          |           |               |        |           |               |
| 04        | F                  |                                           |        |        |    |                |              |                |               |                |                |                |                               |                |           |               |          |               |        |           |                  |          |           |               |        |           |               |

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading consecutively.

**Example of Updating and Deleting Records**

Sometimes you want to update the records in your customer master file. The transaction file contains two input record types. One type (those with D in column 1) identifies records deleted from the master file. The other type (those with 3 in column 1) contains information needed to update the master file. Figure 39 on page 101 shows how to code the RPG specifications to update records and delete records from the master file.
UPDATING A SEQUENTIAL FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME IPEAFBLENRLENLK1AIODEV+.........EXIT++......A+++U++.*
FTRANS IP AF 96 96 DISK
FMASTER US AF 256 128 DISK
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME SQNORIPOS1NCCPOS2NCCPOS3NCC

............................PFromTo++DFIELD+L1M1FrPoNeEq...*
I 2 7 CUSTNOL1M1
I AB 02 1 C3
I 2 7 CUSTNOL1M1
I 12 172CHG
I 18 232PAYMT
I 24 292CRDT
IMASTER NS 03 1 CC 2 CM 128NCD
I OR 04 1 CC 2 CM 128 CD
I 3 8 CUSTNOL1M1
I 16 35 NAME
I 76 76 CRLIM
I 80 852CHARGE
I 86 912PAY
I 92 972CREDIT
I 98 1032BAL

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* Indicator 01 is on for an input record which deletes a disk record.
C* Indicator 21 is set for use during update of the MASTER file.
C*
C 01 MR SETON 21

Figure 39 (Part 1 of 2). Updating and Deleting Records in a Sequential File
C* The values in the MASTER file are updated by totals accumulated
C* from transaction records.
C*

C  02 MR  ADD  CHG    TOTCHG    62
C  02 MR  ADD  PAYMT  TOTPAY    62
C  02 MR  ADD  CRDT   TOTCRD    62
C  03 MR  ADD  TOTCHG CHARGE
C*
C  03 MR  ADD  TOTPAY  PAY
C  03 MR  ADD  TOTCRD CREDIT
C  03 MR  ADD  TOTCHG BAL
C  03 MR  SUB  TOTPAY BAL
C  03 MR  SUB  TOTCRD BAL
C*
C* Set off indicator 21 for next cycle.
C*
CL1   SETOF       21
C*
C* Clear accumulator fields for the next cycle.
C*
CL1   SUB  TOTCHG TOTCHG
CL1   SUB  TOTPAY TOTPAY
CL1   SUB  TOTCRD TOTCRD

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn............................................
OMASTER D  03 MRN21
0....................N01N02N03Field+YBEnd+PConstant/editword++++++++++...*
 0 CHARGE    85
 0 PAY       91
 0 CREDIT    97
 0 BAL      103
 0 DDEL  04 MR 21

Figure 39 (Part 2 of 2). Updating and Deleting Records in a Sequential File
Updating Randomly by Relative Record Number

You can update records in a sequential file randomly by relative record number. The file is defined as a chained file (C in column 16 of the file description specifications). This means that the program reads a record from the file when a CHAIN operation occurs in the calculation specifications, and it writes a record to update the file during detail output, total output, or exception output time in the RPG program cycle.

Code the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading randomly by relative record number.

Updating Randomly by Relative Record Number and/or Consecutively

You can update records in a sequential file randomly by relative record number and/or consecutively. That is, the file is defined as a full-procedural file (F in column 16 of the file description specifications). The record updated can be read either randomly by relative record number with a CHAIN operation or consecutively with a READ or READP operation. The record cannot be read consecutively with a READE operation, because the READE operation cannot read by relative record number. The output operation to update the record can occur during detail output, total output, or exception output time of the RPG program cycle.

Code the file description specifications for the sequential file as shown below:

Column 15 must contain U to indicate that the file is an update file.

Chapter 6. Using a DISK File 103
Entries in the other columns are the same as those for reading randomly by relative record number and/or consecutively.

**Updating Randomly by Address Output (Addrout) File**

You can update records in a sequential file processed by an address output (addrout) file. The sequential file can be a primary, secondary, or full-procedural file (P, S, or F in column 16 of the file description specifications). If the file is a primary or secondary file, the program reads a record at input time of the RPG program cycle, and it writes a record to update the file at detail output or exception output time of the program cycle. If the file is a full-procedural file, the program reads a record when the READ operation occurs in the calculation specifications, and it writes a record to update the file at detail output, total output, or exception output time of the program cycle.

Code the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading randomly by address output file.

Entries on the file description specifications for the address output file and on the extension specifications are the same as those for reading randomly by address output file.

**Adding Records to a Sequential File**

After a file is created, you can add records to it in either of two ways:

- At the end of records in the file
- Between records in the file.
Adding Records at the End of a File

To add records at the end of a sequential file, code the file description and output specifications as shown below:

On the file description specifications, all entries except column 66 are the same as those for creating a sequential file. The A in column 66 indicates that you will add records to the file described on this line.

On the output specifications, columns 7 through 14 must contain the name of the file if this is the first record on the output specifications or if the previous record on the output specifications is for a different file.

Column 15 must contain H (heading), D (detail), T (total), or E (exception) to indicate the type of record written.

Columns 16 through 18 must contain ADD to indicate that the fields defined on the following lines form the record added to the file.

Columns 23 through 31 can contain conditioning indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

Example of Adding Records at the End of a File

As you get new customers, you want to add them to the sequential customer file you created in Figure 38 on page 91. Because you assign customer numbers sequentially, you can add each new customer record at the end of the records already in the file.

Figure 40 on page 106 shows how to code the RPG specifications to add records at the end of the sequential customer file.
ADDING RECORDS TO A SEQUENTIAL FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME IPEAFB\lenR\lenLK1AI0vK\locEDevice+......Exit++......A.....U+.*
FINFILE IP F 96 96 DISK
FSEQDISK 0 F 256 128 DISK A
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME SqNORiPos1NCCPos2NCCPos3NCC...........................................
IINFILE 011 20 1 D2
I..................................................PFromTo++DField+L1M1FrPoNeEq...*
I 2 7 CUSTNO
I 8 9 STATE
I 10 11 CO
I 12 14 CITY
I 15 34 NAME
I 35 54 ADDR
I 55 74 CITST
I 021 30 1 D3
I 2 7 CUSTNO
I 8 8 CRLIM
I 9 47 REST

Figure 40 (Part 1 of 2). Adding Records at the End of a Sequential File
Adding Records to a Sequential File

Adding Records at the End of a Sequential File

Adding Records between Records in a File

You can also add records between records in a sequential file that is processed randomly by relative record number. For example, you may have to add new records between existing records in order to keep the file in a particular order when the control fields of the new records are not higher in sequence than the control fields of records already in the file. Such a file must be one that allows deletion of records. That is, when the file was created, the DFILE-YES parameter must have been specified on the OCL FILE statement. For information about the OCL FILE statement, see System Reference for the System/36 Environment.

To add records between records in a sequential file, code the unshaded columns of the file description specifications as shown below:

In the first line of the file description specifications:

- Columns 7 through 14 must contain the name of the file.
- Column 15 must contain I or U to identify the file as an input file or an update file.
- Column 16 must contain C or F to identify the file as a chained or full-procedural file.
• Column 19 must contain F or blank to indicate that all records in the file have the same length.
• Columns 20 through 23 must contain the block length or blanks.
• Columns 24 through 27 must contain the record length.
• Column 28 must contain R if column 16 contains C.
• Columns 40 through 43 must contain DISK.
• Column 66 must contain A to indicate that you will add records to the file.
• Columns 71 and 72 can contain an external indicator, U1 through U8.

In the second line:
• Column 53 must contain K to indicate that this line is a continuation line that provides additional information about the file.
• Columns 54 through 58 must contain RECNO, which stands for relative record number.
• Columns 60 through 65 must contain the name of the field into which the relative record is placed. The field must be defined on either the input specifications or the calculation specifications as a 7-position numeric field with zero decimal positions. That field is called the RECNO field.

On the output specifications for the record or records added, columns 7 through 14 must contain the name of the output file.

Column 15 can contain D, T, or E, to indicate if the record is to be written at detail, total, or exception output time of the RPG program cycle.

Columns 16 through 18 must contain ADD to indicate that the fields defined in the following lines form the record or records added to the file.

Columns 23 through 31 can contain output indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 of the output specifications contains E.
The RECNO field identifies the position in the file where the output record is added. (That record is the one described on the output specification that contains ADD in columns 16 through 18.) You must place into the RECNO field the relative record number of the record added to the file. It must be the relative record number of a deleted record. One way to place the relative record number into the RECNO field is to code the following sequence of operations in the calculation specifications:

1. Code a CHAIN operation with the relative record number in factor 1, the name of the chained file in factor 2, and a resulting indicator in columns 54 and 55 that turns on when a record is not found.

2. Code a Z-ADD operation with the same indicator that you used for the CHAIN operation coded as a conditioning indicator (in columns 10 and 11, 13 and 14, or 16 and 17), the relative record number in factor 2, and the RECNO field in the result field.

When a CHAIN operation (for a chained or full-procedural file) or a READ, READE, or READP operation (for a full-procedural file) reads a nondeleted record, data management places into the RECNO field the relative record number of the record read.

When the program tries to add a record to a file, if the relative record number is not the number of a deleted record, the program stops and error message RPG9070 Output to a nondeleted record in file [file name], is displayed. If you respond to the message by choosing option 1, the program continues running but it does not add the record to the file.

You cannot use the RECNO field to add records at the end of a sequential file. For example, if a file contains relative record numbers 1 through 5 and 7 through 10, you can add a record at relative record number 6 but not at relative record number 11. If you try to add a record at the end of a sequential file by using a RECNO field, error message RPG9068 Relative record number beyond extent for file [file name], is displayed.

Example of Adding Records between Records in a File
Figure 41 shows how to code the RPG specifications to add records between records in a sequential file.
DIRECT FILES

A direct file is one in which records are assigned specific record locations on disk. Figure 42 on page 111 shows how direct files are organized. Each record is assigned a specific location in the file, regardless of the order in which it is put in the file. If the file allows deleted records (that is, if the OCL FILE statement used the DFILE-YES parameter when the file was created), unused records in the file are marked as deleted. If the file does not allow deletions, unused records contain blanks.

Direct file organization allows your program to find and read any record in the file directly without first checking other records or searching an index. Therefore, direct file organization has advantages over sequential organization.

The location assigned to a record is called the relative record number. The relative record number is not a disk address; rather, it is a number that states the position of a record in a file. For example, the fifth record in a file has relative record number 5.
Records are stored on disk in the order indicated by the relative record numbers. Spaces are left in the file for missing records (in this case, records 5 and 7).

The programmer usually derives relative record numbers from information in the records.

Figure 42. Organization of a Direct File

Creating a Direct File That Does Not Allow Deletions

To create a direct file that does not allow deletion of records, you must define a DISK file as a chained output file and then write records to the file. Before any output is written to the file, the disk space required for the file is automatically filled with blanks. To write a record to the file, you must first determine the relative record number that will be assigned to that record in the file to be created. Then use that relative record number as factor 1 in a CHAIN operation; as factor 2, use the name of the file to be created. When the CHAIN operation occurs, it reads the blank record at the specified relative record number. When the output operation occurs, the record is written to the same relative record number. The output operation can occur during the detail output, total output, or exception output time of the RPG program cycle.
To create a direct file that does not allow deletions, code entries in the unshaded columns of the file description specifications as shown below:

### Example of Creating a Direct File That Does Not Allow Deletions

In Figure 43 on page 113, the direct file being created, CUSTFILE, is defined on the file description specifications as a chained output file (O and C in columns 15 and 16). The CHAIN operation on the calculation specifications reads the relative record number from the CUSNO field of the input file RECIN, and writes a record from RECIN to the corresponding relative record number in the output file CUSTFILE. Indicator 04 turns on if the record is not found.
Creating a Direct File That Allows Deletions

To create a direct file that allows deletion of records, you must define a DISK file on the file description specifications as a randomly processed output file, and you must use the RECNO continuation line. Also, you must specify the DFILE-YES parameter on the OCL FILE statement for the file.

To write records to this output file, place into the RECNO field the relative record number of the record you want to write, and write data to that record during detail time, total time, or exception time of the RPG program cycle. This method of creating a direct file does not use a CHAIN operation to indicate the relative record number of the record to be written.

Before any output is written to the direct file that allows deletions, the disk space required for the file is automatically filled with deleted records. The relative record number that you place in the RECNO field indicates where the output record is written to the direct file. The information in the output record is written over the deleted record.

If a deleted record is not replaced with data, it remains in the file. A record can be added later at this relative record number (see “Adding Records to a Direct File” on
CREATING A DIRECT FILE THAT ALLOWS DELETIONS

A deleted record cannot be read; if a CHAIN operation chains to a deleted record, the indicator coded in columns 54 and 55 of the calculation specifications turns on to indicate that a record was not found at that relative record number.

If the direct file contains a record with the same relative number as the record you are writing, error message RPG9Sz`{]{\7Sz`{]{\ Output to a nondeleted record in file [file name], is displayed. If the person using the display station responds to the message with option 1, the program bypasses the duplicate record and continues processing.

To create a direct file that allows deletions, code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>Filename</th>
<th>File Type</th>
<th>Mode of Processing</th>
<th>Device</th>
<th>External Record Name</th>
<th>Length of Key Field or of Record Address Field</th>
<th>Overflow Type</th>
<th>Type of File Organization or Additional Area</th>
<th>Length of Key Field or of Record Address Field</th>
<th>Overflow Type</th>
<th>Type of File Organization or Additional Area</th>
<th>Length of Key Field or of Record Address Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2</td>
<td>F</td>
<td>OF R DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 3</td>
<td>F</td>
<td>OF R DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 4</td>
<td>F</td>
<td>OF R DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On the first line:

- Columns 7 through 14 must contain the name of the file.
- Column 15 must contain O to indicate that the file is an output file.
- Column 19 must contain F or blank.
- Columns 20 through 23 must contain the block length or blanks.
- Columns 24 through 27 must contain the record length.
- Column 28 must contain R to indicate that the file is processed randomly.
- Columns 40 through 43 must contain DISK.
- Columns 71 and 72 can contain an external indicator, U1 through U8.

On the second line:

- Column 53 must contain K to indicate that this is a continuation line that provides additional information about the file being described.
- Columns 54 through 58 must contain RECNO, which stands for relative record number.
- Columns 60 through 65 must contain the name of the field that contains the relative record number. The field must be defined on either the input specifications or the calculation specifications as a seven-position numeric field with zero decimal positions. That field is called the RECNO field.
Example of Creating a Direct File That Allows Deletions

In Figure 44, the direct file being created, CUSTFILE, is defined on the file description specifications as an output file that is processed randomly (O and R in columns 15 and 28). The file description continuation line indicates that CUSTNO, which is a field in the input file REcin, contains the relative record number of the record written to the output file CUSTFILE. An output record is written for each record read from REcin. No calculation specifications are required.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
FFilenameIPEAFBlenRlenLK1AILoKlocEDevice+........Exit++.......A....U+.*  
FREcin  IP F 60 60 DISK  
FCUSTFILE0 F 60 60R DISK  
KRECNO CUSTNO  
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
IFilenameSqNIORiPos1INCCPos2NCCPos3NCC..................................  
IREcin  NS 01 1 C1  
I..........................PFromTo+DField+L1M1FrPoNeEq....*  
I 2 80CUSTNO  
I 9 23 CUSTNM  
I 24 39 ADDR  
I 40 55 CITYST  
I 56 60 ZIP  
I 2 60 RECORD  
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
OName++++DFBASbSaN01N02N03Excptn........................................  
OCUSTFILED 01  
O..........................N01N02N03Field+YBEnd+PConstant/editword++++++++++++.*  
O 1 'A'  
O RECORD 60  
```

*Figure 44. Creating a Direct File That Allows Deletions*

Reading a Direct File

After the direct file is created, you can read records from it when you want to display the information, create or update other files, or print a report. You can read records from a direct file in the following ways:

- Consecutively
- Randomly by relative record number
- Randomly by relative record number and/or consecutively
- Randomly by address output (addrout) file.

**Note:** An alternative index file can also be created for a direct file to provide another method of reading the records in the file. For information about creating an alternative index file see “Creating an Alternative Index File for an Indexed File” on page 137.
Reading Consecutively

Reading a direct file consecutively means reading the records in the order in which they occur in the file. That is, the first record in the file is read first, the second record is read second, and so on. If the file allows deletion of records, the program does not read deleted records; it skips them and reads the next record present.

You read a direct file consecutively if you want to look at most or all of the records in the file. In this case, reading consecutively is much more efficient than reading randomly.

To read a direct file consecutively, code entries in the unshaded columns of the file description specification as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P, S, or D:

- If you code P (primary) or S (secondary), the program reads a record at input time of the RPG program cycle. For an explanation of how primary and secondary files are read, see Chapter 12, “Primary/Secondary/Multifile Processing.”

- If you code D (demand), the program reads a record when a READ operation occurs in the calculation specifications. For information about the READ operation code, see Chapter 27, “Operation Codes.”

Column 17 must contain E or blank if column 16 contains P or S. E indicates that all records from the file must be processed before the program can end. Blank indicates that the program can end whether or not all records from the file are processed.

Column 18 must contain A, D, or blank if column 16 contains P or S:

- A indicates that the program checks that the records in the file are in ascending sequence.

- D indicates that the program checks that the records are in descending sequence.

- Blank means that the program does not check the record sequence.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.
Columns 24 through 27 must contain the record length.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

**Example of Reading Consecutively**

Suppose you want to process a direct customer file, named CUSTFILE, to produce a monthly report. This report lists all customers that have not placed any orders during the month. Sales personnel can use this report to plan follow-up calls. The file is in sequence by customer number, and the program checks every record. Therefore, the file is processed consecutively.

Figure 45 shows how to code the specifications to read records consecutively from CUSTFILE to produce REPORT1, a list of recently inactive customers. The OR line on the input specifications causes the program to skip blank record locations, because record-identifying indicator 03 on the OR line is not used elsewhere in the program.

```
F CUSTFILE  IPEAFBlenRlenLKIAvKlocEDevice+++Exit++.....A....U+.*
FCUSTFILEIP F 256 128 DISK
FREPORT1 O F 100 100 OF PRINTER
FREPORT1IP F 256 128 DISK
IFilenameSqNRiPos1NCCPos2NCCPos3NCC................................
ICUSTFILENS 'zerodot2 1 C2 2 CA
I*
I* An OR line with a record-identifying indicator not used
I* elsewhere in the program causes unwanted records to be
I* bypassed, including blank records.
I*
I OR 03
I.........................PFromTo++DField+L1M1FrPoNeEq...*
I 3 6 CUSTNO
I 10 12 SLSMN#
I 13 30 CUSNAM
I 47 62 CTYSTA
I 68 69 CREDIT
I 70 750LSTORD
I 82 882CURPER
I 89 952LSTPER
I 96 1012ARLT30
I 102 1072AR3060
I 108 1132AR6090
I 114 1192AR0V90
```

*Figure 45 (Part 1 of 2). Reading Records Consecutively from a Direct Customer File*
READING A DIRECT FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++Op codeFactor2+++ResultLenDHHiLoEqComments+++++++
C  02  CURPER  COMP 0  04
C  02 04  Z-ADDARLT30  TOTAR  72
C  02 04  ADD  AR3060  TOTAR
C  02 04  ADD  AR6090  TOTAR
C  02 04  ADD  AROV90  TOTAR

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn.................................
OREPORT1 H  206  1P
  0  OR  0F
0...............N01N02N03Field+YBEnd+PConstant/editword+++++++++++*
  0     8 'CUSTOMER'
  0     21 'NAME'
  0     42 'CITY,STATE'
  0     56 'SALESMAN'
  0     69 'LAST ORDER'
  0     84 'SLS PREV PER'
  0     90 'CRDT'
  0     99 'TOT A/R'
  0  D 1  02 04
  0  CUSTNO  6
  0  CUSNAM  28
  0  CTYSTA  46
  0  SLSMN#  53
  0  LSTORDY  68
  0  LSTPERJ  82
  0  CREDIT  89
  0  TOTAR J 100

Figure 45 (Part 2 of 2). Reading Records Consecutively from a Direct Customer File
Reading Randomly by Relative Record Number

Reading a direct file randomly by relative record number means reading only those records that you specifically identify by their position relative to the beginning of the file. To read those records, you must use a CHAIN operation in the calculation specifications. Factor 1 of the CHAIN operation must contain the relative record number itself or the name of the field, table, or indexed array that contains the relative record number. For more information about the CHAIN operation code, see Chapter 27, “Operation Codes.”

To read a direct file randomly by relative record number, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain C to indicate that the file is a chained file. To read this file, you must code a CHAIN operation in the calculation specifications.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain R to indicate that the file is processed randomly.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Example of Reading Randomly by Relative Record Number

The “Example of Reading Consecutively” on page 117 processed the direct customer file CUSTFILE consecutively. Now suppose that you want to read records from that file randomly. For example, you might want to make inquiries each day about customer accounts whose records have record identification code I in position 1, followed by the customer account number (CSTMER).

The program reads an input record (the customer account number) from the primary input file, INFILE. The program uses the customer account number as the relative record number to chain to CUSTFILE. If the program finds a record in CUSTFILE that has the same customer account number as the record in INFILE, the computer prints sales and accounts receivable information for that customer. If
the program does not find a record in CUSTFILE that matches the customer account number, the message Record not found--Invalid record number is printed.

Figure 46 shows the printer output for this example. Figure 47 on page 121 shows how to code the specifications for this example.

Figure 46. Printer Output from Random Inquiries into a Direct Customer File
READING A DIRECT FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+........Exit++......A....U+.*
FINFILE IP F 96 96 DISK
F*
F* The direct file CUSTFILE is defined as a chained input file
F* to be retrieved randomly.
F*
FCUSTFILEIC F 256 128R DISK
FPRINTOUTO F 100 100 OF PRINTER
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.................................*
IINFILE NS 07 1 CI
I..............................PFromTo++DField+L1M1FrPoNeEq...*
I 2 50CSTMER
I*
I* A chained file must contain an alphabetic sequence entry.
I*
ICUSTFILENS
I 2 2 ACCODE
I 3 60CUSTNO 08
I 10 12 SLSMN#
I 68 69 CREDIT
I 70 750LSTORD
I 76 810LSTPAY
I 82 882CURPER
I 89 952LSTPER
I 96 1012ARLT30
I 102 1072AR3060
I 108 1132AR6090
I 114 1192AROV90

Figure 47 (Part 1 of 3). Reading a Direct Customer File Randomly by Relative Record Number
READING A DIRECT FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C*
C* The customer number from the input record is used as the
C* relative record number to chain to the direct file.
C* Indicator 13 will turn on if a record is not found
C* in the direct file.
C*
C SETOF 'zerodot8
C 'zerodot7 CSTMER CHAINCUSTFILE 13
C*
C* Indicator 13 is used to condition subsequent operations.
C*
C 07N13N08 Z-ADDARLT30 TOTAR 72
C 07N13N08 ADD AR3060 TOTAR
C 07N13N08 ADD AR6090 TOTAR
C 07N13N08 ADD AROV90 TOTAR

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn....................................
OPRINTOUTH 204 1P
O OR OF
O..................N01N02N03Field+YBEnd+PConstant/editword+++++++++++*
O 8 'CUSTOMER'
O 18 'ACTIVITY'
O 28 'SALESMAN'
O 36 'CREDIT'
O 48 'LAST ORDER'
O 58 'LAST PAY'
O 72 'SALES THIS PER'
O 86 'SALES LAST PER'
O 97 'TOTAL A/R'

Figure 47 (Part 2 of 3). Reading a Direct Customer File Randomly by Relative Record Number
* N13 means that this line will be printed if a record is found in the direct file.

* When a record is not found in the direct file, this line is printed.

* When a record is found in the direct file with a zero customer number, this line is printed.

Figure 47 (Part 3 of 3). Reading a Direct Customer File Randomly by Relative Record Number

Reading Randomly by Relative Record Number and/or Consecutively

You can use a full-procedural file to read a file both randomly and consecutively. You can read a full-procedural file randomly like a chained file and/or consecutively like a demand file. That is, you can chain to a specific relative record number in the file and then read records consecutively from that point. To read the file randomly, you use a CHAIN operation in the calculation specifications; to read it consecutively, you use a READ or READP operation. You cannot use a READE operation to read the file consecutively, because the READE operation cannot read by relative record number. For example, if you code a CHAIN operation to relative record number 10 and then code a READ operation, the program chains to relative record number 10 and then reads the following record.

It is not necessary to code both a CHAIN and a READ or a READP operation, but you must code at least one CHAIN, one READ, or one READP operation in order
to read a full-procedural file. For information about the CHAIN, READ, and READP operation codes, see Chapter 27, “Operation Codes.”

Code a full-procedural file as an input file. Code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>File Type</th>
<th>File Designation</th>
<th>Made of Processing</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>Name of Labels</th>
<th>Extent Exit for DAM</th>
<th>Storage Index Labels S/N/E/M</th>
<th>Extent Exit</th>
<th>File Type</th>
<th>File Designation</th>
<th>Mode of Processing</th>
<th>Length of Key Field or of Record Address Field</th>
<th>Key Field Starting Location</th>
<th>Overflow Indicator</th>
<th>Block Length</th>
<th>Record Address Type</th>
<th>Extension Code</th>
<th>Record Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2</td>
<td>F</td>
<td></td>
<td></td>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 3</td>
<td>F</td>
<td></td>
<td></td>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 4</td>
<td>F</td>
<td></td>
<td></td>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain F to indicate that the file is a full-procedural file. You must code a CHAIN, READ, or a READP operation in the calculation specifications in order to read a full-procedural file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Reading Randomly by Address Output (Addrouout) File

An address output (addrouout) file is a record address file produced by a sort program. (A record address file is an input file that tells the program the records that are to be read from a DISK file and the order in which to read them.) An address output file contains the relative record numbers of the records in a DISK file. The advantages of an address output file are that:

- The space required for the address output file is much less than the space required for a sorted sequential file.
- The sort runs much faster.
- The original file is unchanged.

You can have only one address output file in a program. When an RPG program uses an address output file, it automatically reads the relative record numbers consecutively from the address output file. You do not have to code a READ operation for the address output file. Then, using the relative record number, the program randomly reads the DISK file to process the corresponding record. In this way, the program can process a direct DISK file in a new sequence without actually sorting.
the records and creating a new file. Also, once the file description and extension specifications are coded for the DISK file and for its associated address output file, you can code the DISK file as though you were reading the direct file sequentially.

If the DISK file is a full-procedural file, you must code a READ operation in the calculation specifications; you cannot use a CHAIN, READE or READP operation to read a full-procedural file randomly by an address output file. No input specifications are required for the address output file.

If you want to read records randomly by an address output file, code the file description specifications for the direct file as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P, S, or F:

- If you code P (primary) or S (secondary), the record is read as part of the RPG program cycle. For an explanation of how primary and secondary records are read, see Chapter 12, "Primary/Secondary/Multifile Processing."
- If you code F (full-procedural), you must code a READ operation in the calculation specifications.

Column 18 can contain A, D, or blank if column 16 contains P or S:

- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records in the file are in descending sequence.
- Blank indicates that the program does not check the sequence of records in the file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain R if column 16 contains P or S. The R indicates that the file is processed randomly by an address output file.

Column 31 must contain I if column 16 contains P or S. The I indicates that relative record numbers from the address output file are used to process the file.
Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

For the address output file, code entries in the unshaded columns of the file description specifications as shown below:

---

<table>
<thead>
<tr>
<th>Line</th>
<th>File Type</th>
<th>Made of Processing</th>
<th>File Designation</th>
<th>File Description</th>
<th>Symbolic Device</th>
<th>Device</th>
<th>Name of Label Exit</th>
<th>Length of Key Field or of Record Address Field</th>
<th>Record Address Type</th>
<th>Type of File Organization or Additional Area</th>
<th>Extent Exit for DAM</th>
<th>Storage Index</th>
<th>Labels SNEM</th>
<th>File Condition</th>
<th>Number of Tracks for Cylinder Overflow</th>
<th>Number of Extents for DAM</th>
<th>Tape Rewind RURU</th>
<th>External Record Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>I</td>
<td>R</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain R to indicate that the file is a record address file.

Column 17 must contain E or blank. E indicates that all records from the file must be processed before the program can end. A blank indicates that the program can end whether or not all records from the file are processed.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Column 27 must contain 3 because each record in an address output file is a relative record number, which is always three positions long.

Column 30 must contain 3 because relative record numbers in address output files are always three positions long.

Column 31 must contain I to indicate that relative record numbers are used in processing.

Column 32 must contain T to indicate that the file is an address output file.

Column 39 must contain E to indicate that the file is further described on extension specifications.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.
Two entries are required on the extension specifications:

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Table or Array Name</td>
</tr>
<tr>
<td>1</td>
<td>Length of Entry</td>
</tr>
<tr>
<td>2</td>
<td>Table or Array Name</td>
</tr>
<tr>
<td>3</td>
<td>Length of Entry</td>
</tr>
<tr>
<td>4</td>
<td>Table or Array Name</td>
</tr>
<tr>
<td>5</td>
<td>Length of Entry</td>
</tr>
</tbody>
</table>

Columns 11 through 18 must contain the name of the address output file. This name must be the same one coded on the file description specifications for the address output file.

Columns 19 through 26 must contain the name of the direct file processed by the address output file. This name must be the same one coded on the file description specifications for the direct file.

**Updating a Direct File**

Updating records in a direct file involves reading a record, changing some data in the record, and writing the record back to its original location in the file. If you try to update a record that was not the last record read, error message RPG9043 [Program name] tried to update a record before reading it from file [file name], is displayed. The fields to be updated must be described on both the input and the output specifications.

When you update records in a direct file, the file can be processed in any of the following ways:

- Consecutively
- Randomly by relative record number
- Randomly by relative record number and/or consecutively
- Randomly by address output (addrout) file.

**Deleting Records from a Direct File**

Updating a file can include deleting records from the file. To allow records to be deleted from the file, the OCL FILE statement coded when the file was created must use the DFILE-YES parameter. For information about the FILE statement, see the Programming: System Reference for the System/36 Environment. If you try to delete a record from a file that does not allow deletions, error message RPG9067 Invalid operation attempted to file [file name], is displayed.
To delete a record, you first read the record (either randomly or consecutively) and then, with DEL coded in the output specification, write the record back to the same file. Code entries in the unshaded columns of the output specifications as shown below:

Columns 7 through 14 must contain the name of the output file.

Column 15 must contain D, T, or E, to indicate that the record is to be deleted at detail, total, or exception output time of the program cycle.

Columns 16 through 18 must contain DEL to indicate that the record is to be deleted.

Columns 23 through 31 can contain output indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

Records are not physically removed from a file when they are deleted. Instead, deleted records are filled with hexadecimal FFs. That is, all the bits for every character in the deleted record are set on.

**Updating Consecutively**

You can update records in a direct file consecutively. If the file is a primary or secondary file (P or S in column 16 of the file description specifications), the program reads a record from the file at input time in the RPG program cycle, and the program writes a record to update the file during detail output or exception output time in the program cycle. If the file is a demand file (D in column 16), the program reads a record when a READ operation occurs in the calculation specifications, and the program writes a record to update the file at detail output, total output, or exception output time in the program cycle.
### Updating a Direct File

Code the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>File Addition/Unordered</th>
<th>Device</th>
<th>Name of Symbolic Device</th>
<th>Option Entry</th>
<th>Storage Index</th>
<th>Type of File</th>
<th>Organization or Additional Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading consecutively.

### Updating Randomly by Relative Record Number

You can update records in a direct file processed randomly by relative record number. The file is defined as a chained file (C in column 16 of the file description specifications). This means that the program reads a record from this file when a CHAIN operation occurs in the calculation specifications and that the program can write a record to update the file during detail output, total output, or exception output time in the RPG program cycle.

Code the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>File Addition/Unordered</th>
<th>Device</th>
<th>Name of Symbolic Device</th>
<th>Option Entry</th>
<th>Storage Index</th>
<th>Type of File</th>
<th>Organization or Additional Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading randomly by relative record number.

### Example of Updating Randomly by Relative Record Number

In this example, you want to prepare daily invoices for customer orders for the file described in the “Example of Reading Consecutively” on page 117 and use the information from the invoices to update the customer file CUSTFILE. The records in the invoice file (INFILE) are unordered, so it is processed randomly.

The records in the invoice file contain the date and total amount of transactions for each customer. The records also contain new addresses when the customer addresses change. The program reads each record and uses the customer number (CUSTMR) to chain to the direct file. The amount of the transaction is
added to the total sales for the current period (CURPER) and to the accounts receivable amount (ARLT30). The transaction date is placed in the date of last order (LSTORD) field in the customer record.

If an address change is indicated (by X in column 18 of the input record), the new customer address replaces the old address.

If a record is not found in CUSTFILE because of an invalid relative record number, the input record and the message Above record not found--Invalid customer number are printed.

Figure 48 shows how to code the specifications for this example.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME IPEAFBlenRlenLK1AI0vKlocDevice+.......Exit++......A....U+.*
FINFILE IP F 96 96 DISK
FCUSTFILEUC F 256 128R DISK
FPRINT 0 F 96 96 PRINTER

Figure 48 (Part 1 of 2). Updating a Direct Customer File Randomly by Relative Record Number
Figure 48 (Part 2 of 2). Updating a Direct Customer File Randomly by Relative Record Number
Updating Randomly by Relative Record Number and/or Consecutively

You can update records in a direct file randomly by relative record number and/or consecutively because the file is defined as a full-procedural file (F in column 16 of the file description specifications). The record to be updated can be read either randomly by relative record number with a CHAIN operation or consecutively with a READ or READP operation. The record cannot be read consecutively with a READE operation because the READE operation cannot read by relative record number. The output operation to update the record can occur during detail output, total output, or exception output time of the RPG program cycle.

Code the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

Entries in the other columns are the same as those for reading randomly by relative record number and/or consecutively.

Updating Randomly by Address Output (Addrout) File

You can update records in a direct file processed by an address output (addrout) file. The direct file can be a primary, secondary, or full-procedural file (P, S, or F in column 16 of the file description specifications). If the file is a primary or secondary file, the program reads a record from the file at input time of the RPG program cycle, and writes a record to update the file at the detail output or exception output time of the program cycle. If the file is a full-procedural file, the program reads a record from the file when a READ operation occurs in the calculation specifications, and the program writes a record to update the file at detail output, total output, or exception output time of the program cycle.

Code the file description specifications as shown below:
Adding Records to a Direct File

You can add records to a direct file that is processed randomly by relative record number. The file must be one that allows records to be deleted; that is, when the file was created, DFILE-YES must have been specified on the OCL FILE statement. For information about the OCL FILE statement, see the Programming: System Reference for the System/36 Environment.

To add records to a direct file, code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>File Type</th>
<th>File Designation</th>
<th>Mode of Processing</th>
<th>Record Address Type</th>
<th>Overflow Indicator</th>
<th>Block Length</th>
<th>Record Length</th>
<th>External Record Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the first line of the file description specifications:

- Columns 7 through 14 must contain the name of the file.
- Column 15 must contain I or U to indicate that the file is an input file or an update file.
- Column 16 must contain C or F to indicate that the file is a chained file or a full-procedural file.
- Column 19 must contain F or blank.
- Columns 20 through 23 must contain the block length or blanks.
- Columns 24 through 27 must contain the record length.
- Column 28 must contain R if column 16 contains C.
- Columns 40 through 43 must contain DISK.
- Column 66 must contain A to indicate that records are to be added to the file.
- Columns 71 and 72 can contain an external indicator, U1 through U8.

In the second line:

- Column 53 must contain K to indicate that this line is a continuation line.
- Columns 54 through 58 must contain RECNO, which stands for relative record number.
- Columns 60 through 65 must contain the name of the field into which the relative record number is placed. The field must be defined on either the input
On the output specifications for the record to be added, columns 7 through 14 must contain the name of the output file.

Column 15 can contain D, T, or E, to indicate that the record is to be added at detail, total, or exception output time of the program cycle.

Columns 16 through 18 must contain ADD to indicate that the fields defined on the following lines form the record to be added to the file.

Columns 23 through 31 can contain output indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 of the output specifications contains E.

The RECNO field identifies the position in the file where the output record is to be added. (That record is the one described on the output specification that contains ADD in columns 16 through 18.) You must place into the RECNO field the relative record number of the record to be added to the file. It must be the relative record number of a deleted record. One way to place the relative record number into the RECNO field is to code the following sequence of operations in the calculation specifications:

1. Code a CHAIN operation with the relative record number in factor 1, the name of the chained file in factor 2, and a resulting indicator in columns 54 and 55 that turns on when a record is not found.

2. Code a Z-ADD operation with the same indicator that you used for the CHAIN operation as a conditioning indicator (in columns 10 and 11, 13 and 14, or 16 and 17), the relative record number in factor 2, and the RECNO field in the result field.

When a CHAIN operation (for a chained file) or a READ operation (for a full-procedural file) reads a nondeleted record, data management places into the RECNO field the relative record number of the record read.
CREATING AN INDEXED FILE

When the program tries to add a record to the file, if the relative record number is not the number of a deleted record, the program stops and error message RPG9078 Output to a nondeleted record in file [file name], is displayed. If you respond to the message by choosing option 1, the program continues but it does not add the record to the file.

INDEXED FILES

An indexed file has two parts: an index and the data records. The index contains an entry for each record in the file. Each index entry also has two parts: a key field and the location of the record in the file for that index entry. The key field contains data that identifies each record individually. For example, the customer account number could be the key field to identify each record in a customer master file. The second part of the index contains the location of the record in the file.

Records are stored in the data portion of the file in the same order in which they are written to the file. When a record is stored in the data portion, an entry for the record is made in the index.

Creating an Indexed File

You can create an indexed file by writing records in ascending order of key field or without reference to the key field. The AS/400 system will always do an unordered load of your file, regardless of how you write the records when you create your indexed file.

To create an indexed file, describe the file by coding the entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain O to indicate that the file is an output file.

Column 19 must contain F or blank to indicate that the record length is fixed. That is, all records in the file must be the same length.

Columns 20 through 23 must contain the block length or blanks. The maximum block length is 9999. If you do not enter any number in these columns, the block length equals the record length.

Columns 24 through 27 must contain the record length. The record length can be any number from 1 through 9999.
Columns 29 and 30 must contain the length of the key field in each record. The maximum length is 99 positions unless an indexed file is processed sequentially within key-field limits using a CONSOLE device, in which case the maximum length is 20 positions. Key fields in packed-decimal format can be up to 8 positions in length.

Column 31 must contain A if the key field is in zoned-decimal or alphameric format, or P if the key field is in packed-decimal format. If you create a file with a key field in packed-decimal format, you must also code the key field in packed-decimal format on the output specifications (P in column 44).

Column 32 must contain I to indicate that the file is an indexed file.

Columns 36 through 38 must contain the record position in which the key field begins. The maximum number you can use for the starting position of the key field is the record length minus the length of the key field + 1.

Columns 40 through 43 must contain DISK.

Column 66 (the former File Addition) has no relevance on the AS/400 system. The value in column 66 will be syntax-checked, but ignored. You can leave it blank.

Columns 71 and 72 can contain an external indicator, U1 through U8.
Example of Creating an Indexed File

Figure 49 shows how to code the file description, input, and output specifications to create an indexed file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEipeafblenr1enlk1ai0vklocedevice+..........exit++.......a....u+.*
FINPUT ipe f 96 96 disk
F*
F* The output file, MASTER, is an indexed file to be loaded and
F* processed by key fields.
F*
FMASTER o f 192 96 6ai 2 disk
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEsqnoriapos1nccpos2nccpos3ncc.................................*
I*
I* The key fields from which the index is to be built appear as
I* positions 2 to 8 of the output records.
I* As the DISK file is loaded, the key field is extracted from
I* the record into an index entry containing the location of
I* the record.
I*
IINPUT ns 01
I.................................................pfromto+dfield+l1mlfroneeq...*
I 1 96 RECORD
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
ONAME++++DFBASbSaN01N02N03Excptn.................................*
OMASTER d 01
O..................N01N02N03Field+YBEnd+PConstant/editword++++++++++...*
0 96 RECORD
```

Figure 49. Creating an Indexed File

Creating an Alternative Index File for an Indexed File

After you create an indexed file, you can create alternative index files for that file. Using an alternative index file is equivalent to using an address output (addrout) file to process an indexed file. Each alternative index file uses one or more different portions of the record in the physical file as the key field. (An alternative index file may also be created for a sequential or direct physical file.) Therefore, you can process records from the file in various sequences, depending on the index that you use. For example, for a personnel file you can use the employee number as the primary index key and the department number as a key in an alternative index file.
CREATING AN INDEXED FILE

An alternative index file has either a single field as its key, as have indexed files, or it can have up to three noncontiguous fields as its key. Only alternative index files can have keys with noncontiguous fields. For example, the fields in positions 1 through 6, 8 through 10, and 20 through 24 can be specified as one key:

```
Key
```

One field cannot overlap another field of the key. For example, if one key is specified in positions 10 through 15, another field cannot be specified in positions 14 through 20. For details on coding for keys with noncontiguous fields, see “Using an Alternative Index File with Noncontiguous Fields as its Key” on page 141.

The existing file from which you create alternative index files is called the physical file. To create an alternative index file for a physical file, use the OCL BLDINDEX procedure. On the BLDINDEX display, you specify:

- The name of the alternative index file to be created
- The starting location of each field that is to be part of the key (1 to 3 fields may be used, and the value for each field must be a decimal number from 1 through 4096)
- The length of each field that is to be part of the key (the total length of all fields that make up the key cannot be greater than 99 bytes)
- The name of the physical file
- Creation date of the physical file
- If duplicate key fields are allowed for the alternative index file
- The preferred disk location.

For more information about the BLDINDEX procedure, see the Programming: System Reference for the System/36 Environment manual.

You can use an alternative index file to do any of the following:

- Read records sequentially by key field
- Read records randomly by key field
- Read records within key-field limits by using the SETLL operation or a limits file
- Update records in the file
- Delete records from the file
- Add records to the file.
These are exactly the same operations that you can do with the primary index. The program does not know if the indexed file defined in the program uses the primary index or an alternative index file. All the program knows is that the file is an indexed file. Therefore, the program treats an alternative index file in the same manner as the primary index.


Example of Creating an Alternative Index File
Suppose that you want to create an alternative index for the file named MASTER that was created in Figure 49 on page 137. To do so, you use the BLDINDEX procedure. For example, you might enter the following values:

<table>
<thead>
<tr>
<th>BLDINDEX PROCEDURE</th>
<th>Optional-*</th>
<th>Ignored-%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an alternative index for a physical file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name of file to be created</td>
<td>CUSTMAST</td>
<td></td>
</tr>
<tr>
<td>Starting position for first field of key</td>
<td>1-4096 20</td>
<td></td>
</tr>
<tr>
<td>Length of first field</td>
<td>1-120 10</td>
<td></td>
</tr>
<tr>
<td>Starting position for second field of key</td>
<td>1-4096  ___</td>
<td>*</td>
</tr>
<tr>
<td>Length of second field</td>
<td>1-120 ___</td>
<td>*</td>
</tr>
<tr>
<td>Starting position for third field of key</td>
<td>1-4096 ___</td>
<td>*</td>
</tr>
<tr>
<td>Length of third field</td>
<td>1-120 ___</td>
<td>*</td>
</tr>
<tr>
<td>Name of physical file</td>
<td>MASTER ___</td>
<td></td>
</tr>
<tr>
<td>Creation date of physical file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allow duplicate keys</td>
<td>DUPKEY,NODUPKEY</td>
<td></td>
</tr>
<tr>
<td>Preferred disk location</td>
<td>A1,A2,A3,A4,block number</td>
<td></td>
</tr>
</tbody>
</table>

Example of Using an Alternative Index File with Only One Field as its Key
The following example shows how to use the alternative index named CUSTMAST, which we just created, to read records.
CREATING AN INDEXED FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEIPEAFBLENRLK1AI0VKL0CEDEVICE+........EXIT+++……A....U+.*
F*
F* The name of the alternative index is CUSTMAST.
F* The record length for the alternative index must be the same as for
F* the primary index.
F* The length of the key field for the alternative index is 10.
F* The starting position of the key field for the alternative index is
F* 20.
F*
FCUSTMASTID F 96 96 10AI 20 DISK
FPRT O F 132 132 PRINTER
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFILENAMESQNORIPOS1NCMPPOS2NCCPOS3NCCE+............................
ICUSTMASTNS
I.........................................PFromTo++DFIELD+L1ML1FrPoNeEq...
I
20 29 ALTKEY
I
1 96 DATA
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC101N02N03Factor1+++OpcodeFactor2+++ResultLenDHRiLoEqComments++++++
C READ CUSTMAST LR
C NLR EXCEPTPRINT
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
ONAME+++DFBASESbSaN01N02N03EXCPTN..................................*
OPRT E PRINT
0....................N01N02N03FIELD+YBEnd+PConstant/editword++++++++++*
0
ALTVKEY 15
0
DATA 115

Figure 50. Using an Alternative Index File with Only One Field as its Key
Using an Alternative Index File with Noncontiguous Fields as its Key

To use an alternative index file with noncontiguous fields as its key, describe the file in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the alternative index file with noncontiguous fields as its key.

Column 15 must contain I or U. I indicates that the file is an input file. U indicates that the file is an update file.

Column 16 must contain P, S, or F. P indicates that the file is a primary file. S indicates that the file is a secondary file. F indicates that the file is a full-procedural file.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length. The record length for the alternative index must be the same as for the primary index.

Column 28 must contain blank, R, or L.

Columns 29 and 30 must contain the total length of the key. The maximum total length a key can be is 99 positions.

Column 31 must contain A to indicate that the key is in zoned-decimal or alphanumeric format (packed-decimal format is not allowed).

Column 32 must contain I to indicate that the file is an indexed file.

Columns 35 through 38 must contain EXTK, to indicate that the file has noncontiguous fields as its key.

Columns 40 through 43 must contain DISK.

Reading an Indexed File

Records can be read from an indexed file in the following ways:

- Sequentially by key field
- Sequentially within key-field limits
- Randomly by key field
- Randomly and/or sequentially by key field
- Randomly by address output (addrout) file.
READING AN INDEXED FILE

Note: An indexed file can also be read without using the index. When this is done, only the data portion of the file is used. There are three ways to read an indexed file without using the index:

- Consecutively
- Randomly by relative record number
- Randomly by relative record number and/or consecutively.

To read an indexed file in any of these ways without using the index, code the file as a sequential file. For more information about coding a sequential file, see “SEQUENTIAL FILES” on page 89.

Reading Sequentially by Key Field

When a program reads an indexed file sequentially by key field, it reads the records in the order in which the key fields are sequenced, not in the order in which the records exist in the file.

If you want to read records sequentially by key field, code entries in the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 can contain P, S, F, or D:

- If you code P (primary) or S (secondary), the file is read as part of the RPG program cycle. For an explanation of how primary and secondary files are read, see Chapter 12, “Primary/Secondary/Multifile Processing” on page 341.
- If you code D (demand) or F (full-procedural) you must code a READ operation in the calculation specifications in order to read the file. For information about the READ operation code, see Chapter 27, “Operation Codes” on page 713.

Column 17 can contain E or blank if column 16 contains P or S. E indicates that the program must process every record in the file before the program ends. Blank indicates that the program can end before every record in the file is processed.

Column 18 can contain A, D, or blank if column 16 contains P or S:

- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records are in descending sequence.
• Blank indicates that the program does not check the record sequence.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain A or P. A indicates that the key fields are in zoned-decimal or alphameric format. P indicates that the key fields are in packed-decimal format.

Column 32 must contain I to indicate that this is an indexed file.

Columns 35 through 38 must contain the record position in which the key field begins, if the key has only one field. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK and column 16 must contain P, S, or F.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Reading Sequentially within Key-Field Limits

A program can process an indexed file sequentially within key-field limits in either of two ways:

• Using a limits record
• Using the SETLL operation.

Both methods allow you to limit the key fields of the records that you want your program to process. If you want to read records sequentially within key-field limits, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that this file is an input file.
Column 16 must contain P, S, F, or D:

- If you code P (primary) or S (secondary), the program reads a record from the file at input time of the RPG program cycle. For an explanation of how primary and secondary files are read, see Chapter 12, “Primary/Secondary/Multifile Processing” on page 341.

- If you code F (full-procedural), the program reads a record when a READ, READE, READP, or CHAIN operation occurs in the calculation specifications. Only the READ operation code may be used if the file is to be processed within limits using a limits record. When the file is to be processed within limits using the SETLL operation code, the SETLL must be immediately followed by a READ, READE, or READP operation. When a CHAIN operation occurs, the limits set by the SETLL operation are set off. For information about the READ, READE, READP and CHAIN operation codes, see Chapter 27, “Operation Codes” on page 713.

- If you code D (demand), the program reads a record when a READ operation code occurs in the calculation specifications. For information about the READ operation, see Chapter 27, “Operation Codes” on page 713.

Column 18 can contain A, D, or blank if column 16 contains P or S:

- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records are in descending sequence.
- Blank indicates that the program does not check the record sequence.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain L to indicate that the file is processed sequentially within limits.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain A or P. A indicates that the key fields are in zoned-decimal or alphanumerical format. P indicates that the key fields are in packed-decimal format.

Column 32 must contain I to indicate that this is an indexed file.

Columns 35 through 38 must contain the record position in which the key field begins, if the key has only one field. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK and column 16 must contain P, S, or F.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.
Using a Limits Record
When a program processes an indexed file sequentially within key-field limits by using a limits record, it first automatically reads a limits record from a record address file assigned to a DISK or CONSOLE device. A limits record contains the lowest key field and the highest key field of the records in the indexed file to be processed. The program reads records from the indexed file in ascending order of key field, from the low key field in the limits record to the high key field in the limits record. When the upper limit is passed, the program automatically reads another limits record from the record address file and sets new limits. This process continues until the program reaches the end of the record address file or until the program ends because another file reaches the end-of-file condition.

Each limits record contains only one set of limits (the low key field and the high key field). The low key field must begin in position 1 of the record, and the high key field must immediately follow the low key field. Therefore, a limits record is twice as long as a key field. The length of a key field can be from 1 to 99 positions for a file processed using a DISK device, and 1 to 29 positions for a file processed using a CONSOLE device. The low key field and the high key field must have the same length, and that length must be the same as the entry coded in columns 29 and 30 of the file description specifications. Therefore, you may have to code leading zeros when you use numeric key fields. An alphameric key field can contain blanks. For files with noncontiguous keys, the low and high key fields in the limits record contain all subfields that make up each noncontiguous key.

You can use the same set of limits in more than one limits record. Therefore, you can process data records within those limits as many times as you want. If the two key fields in a limits record are equal, the program reads only one data record. The key fields in the limits records can have a different format from the key fields in the files being processed by limits. For example, one can have a packed-decimal format, and the other can have a zoned-decimal format. If the formats differ, the format of the key fields from each file must be indicated by A or P in column 31 of the file description specifications, and the length of the zoned-decimal key field must be twice the length of the packed-decimal key field, minus one or two. See “Column 43 (Packed-Decimal or Binary Field)” on page 571 in Chapter 23, “Input Specifications” for information about this calculation. While the program is running, the format of the key fields in the limits records is changed to the format of the key fields in the files being processed by limits.

To use the record address file from which the limits are read, entries are required in the file description and extension specifications. No input specifications are required for the record address file.
READING AN INDEXED FILE

Code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>F</th>
<th>File Type</th>
<th>Mode of Processing</th>
<th>File Designation</th>
<th>Record Length</th>
<th>Block length</th>
<th>Record Address Type</th>
<th>Overflow Indicator</th>
<th>Name of Label Exit</th>
<th>Storage Index</th>
<th>Extent Exit for DAM</th>
<th>Number of Tracks for Cylinder Overflow</th>
<th>Number of Extents</th>
<th>File Addition/Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 - 4</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain R to indicate that the file is a record address file.

Column 17 must contain E or blank. E indicates that all records from the file must be processed before the program can end. Blank indicates that the program can end before all records from the file are processed.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length, which should be two times the length of the key field.

Columns 29 and 30 must contain the length of the key field.

Column 31 must contain A, P, or blank. A or blank means that the key field is in zoned-decimal or alphanumerical format. P means that the key field is in packed-decimal format. Column 31 must contain A or blank if columns 40 through 46 contain CONSOLE.

Column 39 must contain E to indicate that the file is further described on extension specifications.

Columns 40 through 46 must contain DISK or CONSOLE.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Two entries are required on the extension specifications:
Columns 11 through 18 must contain the name of the limits file. This name must be the same one coded on the file description specifications for the limits file.

Columns 19 through 26 must contain the name of the indexed file to be processed by the limits file. This name must be the same one coded on the file description specifications for the indexed file.

**Using the SETLL Operation**

The SETLL (set lower limit) operation allows you to set the lower limit for the key field during the calculations part of the RPG program cycle. You can use the SETLL operation to process any indexed file that is used as a demand or full-procedural file (that is, any file that has D or F in column 16, L in column 28, and I in column 32 of the file description specifications). However, you cannot process an indexed demand or full-procedural file with the SETLL operation if you are using a record address file to set the key-field limits for the file. The maximum number of files that you can process with the SETLL operation is limited by the number of demand and full-procedural files that an RPG program can use. The number of demand, chained, and full-procedural files that an RPG program can use cannot total more than 50.

Factor 1 of the SETLL operation must contain either the name of a field that contains the lower limit being set or a literal that is used as the lower limit. The name of the field containing the lower limit may be the name of a data structure subfield. Noncontiguous keys may be created to specify the lower limit by using alternative indexes and data structures to process the file. See “Creating an Alternative Index File for an Indexed File” on page 137.

Factor 2 must contain the name of the file for which the lower limit is being set.

The SETLL operation must be followed by a READ, READE, or READP operation. Otherwise, the SETLL operation is ignored. The READE operation reads the next record if factor 1 of READE matches that record’s key. The READP operation reads the record prior to the lower limit specified. Other operations (except for input operations) can come between the SETLL and READ, READE, or READP operations.

Figure 51 on page 148 shows an example of SETLL coding for the SMASTER file and an example of using a limits record to process the MASTER file.
READING AN INDEXED FILE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..* FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice++Exit++.....A....U+. *

F* The input DISK file, MASTER, is an indexed file to be processed F* within the key-field limits contained in the record address F* file, LIMITS.
F* The LIMITS file, which is further described on the extension F* sepcifications, is entered from the CONSOLE device.
F*
F* Each set of limits read from LIMITS consists of the low and F* high account numbers to be processed.
F* Because the account number key field (ACCT) is eight positions F* long, each set of limits includes two 8-position key fields.
F*
FLIMITS IR F 16 16 8 ECONSOLE
FMMASTER IP F 256 64L 8AI 1 DISK
FPRINT O F 132 132 OF PRINTER
FSMASTER ID F 256 64L 8AI 1 DISK
F*

Figure 51 (Part 1 of 3). Processing an Indexed File Sequentially within Key-Field Limits by Using the SETLL Operation for SMaster and by Using a Limits Record for MASTER
READING AN INDEXED FILE

As MASTER is processed within each set of limits, the corresponding records are written to the PRINTER output file, PRINT.

Processing is complete when all sets of limits have been processed.

LIMITS MASTER

File SMASTER is processed by the SETLL operation code.
It uses no extension specifications, and its file name appears in factor 2 of the SETLL operation code.
In this example, the first record read from file SMASTER would be the one whose key field is equal to or the next higher than the literal 'AAAAAAAA'.
Records are read sequentially to end-of-file unless the cycle is interrupted by additional SETLL operations.

CLR START TAG
CLRAN99 'AAAAAAAA' SETLL SMASTER
CLRAN99 SETON 99
CLR READ SMASTER 98
CLRAN98 GOTO START

Figure 51 (Part 2 of 3). Processing an Indexed File Sequentially within Key-Field Limits by Using the SETLL Operation for SMASTER and by Using a Limits Record for MASTER

Chapter 6. Using a DISK File 149
READING AN INDEXED FILE

Figure 51 (Part 3 of 3). Processing an Indexed File Sequentially within Key-Field Limits by Using the SETLL Operation for SMASTER and by Using a Limits Record for MASTER

Reading Randomly by Key Field

You can process an indexed file randomly by key field only if it is a chained file (that is, if it has C in column 16 of the file description specifications). You must use the CHAIN operation to read a record from the file during the calculation part of the RPG program cycle.

If you want to read an indexed file randomly by key field, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

If the keys are contiguous column 16 must contain C or F to indicate that the file is a chained file. If the keys are noncontiguous specify a F only.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain R to indicate that the file is processed randomly.

Columns 29 and 30 must contain the total length of the key.
READING AN INDEXED FILE

Column 31 must contain A or P. A indicates that the key field is in zoned-decimal or alphameric format. P indicates that the key field is in packed-decimal format.

Column 32 must contain I to indicate that the file is an indexed file.

Columns 35 through 38 must contain the record position in which the key field begins, if there is only one field in the key. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK. Column 16 must contain F if columns 35 through 38 contain EXTK.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Reading Randomly and/or Sequentially by Key Field

If you want to read records both randomly by key field and sequentially by key field, use a full-procedural file (F in column 16 of the file description specifications). You can read a full-procedural file randomly by key field by using the CHAIN operation, sequentially by key field by using the READ, READE, or READP operation, or both randomly by key field and sequentially by key field by using the CHAIN and READ, READE, or READP operations. It is not necessary to use both a CHAIN and a READ, READE, or READP operation, but you must code at least one CHAIN, READ, READE, or READP operation in the calculation specifications in order to read a full-procedural file.

Code the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain F to indicate that the file is a full-procedural file.

Column 19 must contain F or blanks.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain A or P. A indicates that the key field is in zoned-decimal or alphameric format. P indicates that the key field is in packed-decimal format.
READING AN INDEXED FILE

Column 32 must contain I to indicate that the file is an indexed file.

Columns 35 through 38 must contain the record position in which the key field begins, if the key has only one field. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Example of Reading an Indexed File Randomly and Sequentially by Key Field

Suppose you want to print a list of all the employees in a particular department. Figure 52 shows how to do so. The department number is entered in the first 5 positions of the local data area. The program uses this department number as the key field. The CHAIN operation reads the first record that has the desired department number in the DEPTNO field of the EMPLOYEE file. The READ operation then reads sequentially all the other records that have the same department number.

```
.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+........Exit+......A.....U+.*
FEMPLOYEEIF F 256 256 5AI 1 DISK
FLIST 0 F 132 132 OA PRINTER
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC....................................
IEMPLOYEENS
I..................PFromTo++DField+L1M1FrPoNeEq...*
I
I 1 5 DEPTNO
I 20 50 EMPNAM
I*
I* The number of the department to be printed is in position 1
I* through 5 of the local data area.
I*
I UDS
I
I 1 5 DNUM
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CCIN01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* The CHAIN operation reads the first record that has the requested
C* department number.
C* If the department number is not found, an error message is printed.
C*
C DNUM CHAINEMPLOYEE 99
C 99 EXCPTBADNUM
C 99 GOTO END
```

Figure 52 (Part 1 of 2). Reading an Indexed File Randomly by Key Field and Sequentially by Key Field
Now, reading sequentially, we can read the records for the rest of the employees in the department.

```
LOOP  TAG
EXCPTPRTEMP
READ EMPLOYEE 98
GOTO END IF EOF, DONE.
DNUM COMP DEPTNO 97
GOTO LOOP IF DEPT NUMBERS SAME, LOOP.
END  TAG
SETON LR
```

Figure 52 (Part 2 of 2). Reading an Indexed File Randomly by Key Field and Sequentially by Key Field

**Reading Randomly by Address Output (Addrout) File**

An address output (addrout) file is a record address file produced by a sort program. A record address file is an input file that tells the program the records that are to be read from a DISK file and the order in which to read them. An address output file contains the relative record numbers of the records in a DISK file. The advantages of an address output file are that:

- The space required for the address output file is much less than the space required for a sorted sequential file.
- The sort runs much faster.
- The original file is unchanged.

You can have only one address output file in a program. When an RPG program uses an address output file, it reads the relative record numbers consecutively from
the address output file. Then, using the relative record number, the program randomly reads the DISK file to process the corresponding record. In this way, the program can process an indexed DISK file in a new sequence without actually sorting the records and creating a new file. Also, once the file description and extension specifications are coded for the DISK file and for its associated address output file, you can code the DISK file as an ordinary indexed file. No input specifications are required for the address output file.

If you want to read records randomly by an address output file, code the file description specifications for the indexed file as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P (primary) or S (secondary). For an explanation of how primary and secondary records are read, see Chapter 12, “Primary/Secondary/Multifile Processing.”

Column 18 must contain A, D, or blank:
- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records in the file are in descending sequence.
- Blank indicates that the program does not check the sequence of the records in the file.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Column 28 must contain R to indicate that the file is processed randomly by an address output file.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain I to indicate that relative record numbers from the address output file are used to process the file.

Column 32 must contain I to indicate that this file is an indexed file.
Columns 35 through 38 must contain the record position in which the key field begins, if there is only one field in the key. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.

For the address output file, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain R to indicate that the file is a record address file.

Column 17 must contain E to indicate that all records from the file must be processed before the program can end, or a blank to indicate that the program can end before all records from the file are processed.

Column 19 must contain F or blank.

Columns 20 through 23 must contain the block length or blanks.

Column 27 must contain 3 because each record in an address output file is a relative record number, which is always three positions long.

Column 30 must contain 3 because relative record numbers in address output files are always three positions long.

Column 31 must contain I to indicate that relative record numbers are used in processing.

Column 32 must contain I to indicate that this file is an indexed file.

Column 39 must contain E to indicate that the file is further described on extension specifications.

Columns 40 through 43 must contain DISK.

Columns 71 and 72 can contain an external indicator, U1 through U8.
Two entries are required on the extension specifications:

<table>
<thead>
<tr>
<th>Line</th>
<th>From Filename</th>
<th>Table or Array Name</th>
<th>Number of Entries Per Record</th>
<th>Number of Entries Per Table or Array</th>
<th>Length of Entry</th>
<th>Length of Entry (Alternating Format)</th>
<th>Table or Array Name</th>
<th>Length of Entry</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 11 through 18 must contain the name of the address output file. This name must be the same one coded on the file description specifications for the address output file.

Columns 19 through 26 must contain the name of the indexed file to be processed by the address output file. The name must be the same one coded on the file description specifications for the indexed file.

Updating an Indexed File

Updating records in an indexed file involves reading a record, changing some data in the record, and writing the record back to its original location in the file. If you try to update a record that was not the last record read, error message RPG9043 [Program name] tried to update a record before reading it from file [file name], is displayed. The fields to be updated must be described on both the input and the output specifications.

When you update records in an indexed file, the file can be processed in any of the following ways:

- Sequentially by key field
- Sequentially within key-field limits
- Randomly by key field
- Randomly and/or sequentially by key field
- Randomly by address output (addrout) file.

Deleting Records from an Indexed File

Updating a file can include deleting records from the file. To allow records to be deleted from the file, the OCL FILE statement coded when the file was created must use the DFILE-YES parameter. For information about the FILE statement, see the Programming: System Reference for the System/36 Environment. If you try to delete a record from a file that does not allow deletions, error message RPG9067 Invalid operation attempted to file [file name], is displayed.
To delete a record, you first read the record (either randomly by key field or sequentially by key field). Then, with DEL coded in the output specifications, you write the record back to the same file. Code entries in the unshaded columns of the output specifications as shown below:

Columns 7 through 14 must contain the name of the output file.

Column 15 must contain D, T, or E, to indicate that the record is to be deleted at detail, total, or exception output time of the program cycle.

Columns 16 through 18 must contain DEL to indicate that the record is to be deleted.

Columns 23 through 31 can contain output indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

Records are not physically removed from a file when they are deleted. Instead, deleted records are filled with hexadecimal FFs. That is, all the bits for every character in the deleted record are set on.

**Updating Sequentially by Key Field**

When you want to update most of the records in a file, process the file sequentially by key field. Code entries in the unshaded columns of the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

The other entries are the same as those for reading an indexed file sequentially by key.
Updating Sequentially within Key-Field Limits

You can update records in an indexed file sequentially within key-field limits. Code entries in the unshaded columns of the file description specifications as shown below:

Column 15 must contain U to indicate that this file is an update file.

The other entries are the same as those for reading an indexed file sequentially within key-field limits.

Updating Randomly by Key Field

You can update an indexed file randomly by key field if it is a chained file (that is, if it has C in column 16 of the file description specifications). Code entries in the unshaded columns of the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

The other entries are the same as those for reading an indexed file randomly by key field.

Example of Updating an Indexed File Randomly by Key Field

Figure 53 on page 159 shows sample program OE400R, which updates an indexed file randomly by key. The program updates the amount due and the date-of-order fields in the customer master file (CUSTMAST) from the customer order file (CUSTORDS). The program reads the customer number (CUSNO) and the amount due for a new order (EXAMT) from the customer order file, CUSTORDS. It also reads the customer’s current amount due from the customer master file (CUSTMAST). On the calculation specifications, the amount due for each item ordered (EXAMT) is added, and the total for the current order is stored in result field TOTDUE. OE400R uses the customer number (CUSNO) to chain to the cor-
responding record in the customer master file, CUSTMAST. The total amount due for the current order (TOTDUE) is added to the customer’s previous amount due (AMDUE), and the result is stored again in AMDUE. On the output specifications, when the program writes to the customer master file (CUSTMAST), the amount-due field (AMDUE) is updated for the customer record. UPDATE is used to update the field that indicates the most recent date that the record was updated.

**Note:** To run this program, you must create the file CUSTORDS by running the program OE140R. You must also code the following procedure:

```plaintext
// LOAD OE400R
// FILE NAME-CUSTMAST
// FILE NAME-CUSTORDS
//RUN
```

```
FilenameIPEAFBlenRlenLK1AI0vKlocEDevice+........Exit++......A....U+.*
FCUSTORDSIP F 256 128 DISK
FCUSTMASTUC F 256 256R 8AI 2 DISK
```

```
FilenameSqNORiPos1NCCPos2NCCPos3NCC..............................*
I*
I* 01, 02, and 03 are Record-identifying indicators.
I* Indicator L1, a control-level indicator turns on
I* when the customer number changes in the CUSTORDS file.
I*
ICUSTORDSNS 01 2 CC 3 CU
I..........................PFromTo++DField+L1M1FrPoNeEq...*
I 4 11 CUSNO L1
I NS 02 2 CC 3 CS
I 4 11 CUSNO L1
I NS 03 2 CI 3 CT
I 4 11 CUSNO L1
I 34 422EXAMT
ICUSTMASTNS
I P 131 1352AMDUE
```

*Figure 53 (Part 1 of 2). Sample Program OE400R (Updating an Indexed File Randomly by Key Field)*
When a line-item record is read from the CUSTORDS file, add the total amount due for this customer order.

```
C 03 ADD EXAMT TOTDUE 92
```

When a control break occurs (the customer number changes), read the previous customer's record from the CUSTMAST file. Add the amount due from this order to the amount already due. Indicator 20 turns on when a customer number is not found in the CUSTMAST file.

```
CL1 CUSNO CHAINCUSTMAST 20
CL1N20 ADD TOTDUE AMDUE
CL1 Z-ADD*ZERO TOTDUE
```

Figure 53 (Part 2 of 2). Sample Program OE400R (Updating an Indexed File Randomly by Key Field)
**Updating Randomly and/or Sequentially by Key Field**

You can update records in an indexed file randomly by key field, sequentially by key field, or both randomly and sequentially by key field if the file is defined as a full-procedural file (F in column 16). Code entries in the unshaded columns of the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

The other entries are the same as those for reading an indexed file randomly by key field and/or sequentially by key field.

**Updating Randomly by Address Output (Addrout) File**

To update records randomly by an address output file, code entries in the unshaded columns of the file description specifications as shown below:

Column 15 must contain U to indicate that the file is an update file.

The other entries are the same as those for reading an indexed file randomly by address output file.

**Adding Records to an Indexed File**

You can add records to an indexed file in either of two ways:

- Randomly by key field
- Sequentially by key field.
Adding Records Randomly by Key Field

You can add records randomly by key field to an indexed file with chaining. Chaining means comparing the key field of the record to be added with the key fields already in the index. The reason for this comparison is to make sure that the record to be added is not a duplicate of a record already in the file. Chaining allows you to design your program so that, if a duplicate key field is found, your program can handle it appropriately without requiring the person using the display station to decide how to respond to an error message. If the program has a logic error that would allow a record with a duplicate key field to be added to the file, or if another program tries to add a record with a duplicate key field during the time between the CHAIN operation and the output operation in this program, the system ensures that the duplicate record is not added. However, a record with a duplicate key field can be added if the DUPKEYS-YES parameter was specified on the OCL FILE statement when the file was created.

To add records randomly by key field, entries are required in the file description and output specifications.

Code entries in the unshaded columns of the file description specification as shown below:

Columns 7 through 14 must contain the file name.

Column 15 must contain I, U, or O to indicate that the file is an input, update, or output file.

Column 16 must contain C or F to indicate that the file is a chained or full-procedural file. If column 16 contains C, you must code a CHAIN operation in the calculation specifications in order to read the file. If it contains F, you must code a READ, READE, READP, or CHAIN operation in the calculation specifications in order to read the file.
Column 19 must contain F or blank to indicate that every record in the file has the same length.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the length of each record.

Column 28 must contain R to indicate that the file is processed randomly by key if this is a chained file (C in column 16), or a blank if this is a full-procedural file (F in column 16). If this is a full-procedural file, column 28 can contain L to indicate that the file is processed within key-field limits by using the SETLL operation.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain A or P. A indicates that the key field is in alphameric or zoned-decimal format. P indicates that the key field is in packed-decimal format.

Column 32 must contain I to indicate that the file is an indexed file.

Columns 35 through 38 must contain the starting position of the key field, if the key has only one field. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK. Column 16 must contain F if columns 35 through 38 contain EXTK.

Columns 40 through 43 must contain DISK.

Column 66 must contain A to indicate that records are added to the file.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Entries are also required in the unshaded columns of the output specifications shown below:

Columns 7 through 14 must contain the name of the file to which the records are added.

Column 15 must contain D, T, or E to indicate if the record is to be added at detail, total, or exception output time of the RPG program cycle.

Columns 16 through 18 must contain ADD to indicate that the fields coded on the following lines form the record to be added to the file named in columns 7 through 14.

Columns 23 through 31 can contain output indicators.
Columns 32 through 37 can contain an EXCPT name if column 15 of the output specifications contains E.

**Example of Adding Records Randomly by Key Field**

Suppose you want to add new inventory items to the indexed inventory file created in the “Example of Creating an Indexed File” on page 137. The new records are not in sequence. Key fields in the new records can be lower than, between, or higher than key fields in the records already in the file. Input and output records will be in the same format as the records used to create the file.

Figure 54 shows how to code the file description, input, calculation, and output specifications to all records added randomly by key field.

```
1... 2... 3... 4... 5... 6... 7...
FILENAME IPEAFBLENRLKE1AI0V1LOCDEVICE+......EXIT++......A....U+.*
FADDFILE IPE F 96 96 DISK
FMATER UC F 192 96R06AI 2 DISK A
1... 2... 3... 4... 5... 6... 7...
FILENAME SquorPiPos1NCCPos2NCCPos3NCC
IADDFILE AA ChaiMASTER 99
I
2 7 ITEMNO
I
1 96 RECORD
IMASTER BA
```

*Figure 54 (Part 1 of 2). Adding Records to an Indexed File Randomly by Key Field*

```
1... 2... 3... 4... 5... 6... 7...
CC1N01N02N03Factor1+++OpdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*  
C* If 99 is on, the item is not in the file, so add the record.
C* If 99 is off, the item has a duplicate in the file, so do
C* not add the record.
C*  
C 01 ITEMNO CHAINMASTER 99
1... 2... 3... 4... 5... 6... 7...
OName++++DFBASbSaN01N02N03Excptn....................................*
0*
0* New records are added to the file.
0*
OMASTER DADD 01 99
0.................N01N02N03Field+YBEnd+PConstant/editword++++++++++++*
0 RECORD 96
```

*Figure 54 (Part 2 of 2). Adding Records to an Indexed File Randomly by Key Field*
Adding Records Sequentially by Key Field

Physically, all records added to an indexed file are placed at the end of the file. However, depending on the value of the key field, a record added sequentially by key can be processed as if it were added in either of two places: between records already in the file or at the end of the file. A key field added between existing key fields must have a value that is lower than the key field in the record currently being processed and higher than the key field in the last record processed. A key field added at the end of the index must have a value that is higher than the highest key field in any record already in the file.

Adding records sequentially can be faster than adding records randomly with chaining if the records to be added are already sorted into ascending order by key field. The reason is that you can use a large block containing many records when you add records sequentially.

To add records sequentially by key field, entries are required in the file description and output specifications.

Code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I, U, or O to indicate that the file is an input, update, or output file.

Column 16 can contain P, S, D, or F:

- If you code P (primary) or S (secondary), the file is read as part of the RPG program cycle. For an explanation of how primary and secondary files are read, see Chapter 12, “Primary/Secondary/Multifile Processing” on page 341.
- If you code D (demand) or F (full-procedural), you must code a READ operation in the calculation specifications in order to read the file. For information about the READ operation code, see “READ (Read)” on page 807.

Column 17 can contain E if column 16 contains P or S. E indicates that the program must process every record in the file before the program ends. Blank indicates that the program can end before all of the records in the file are processed.

Column 18 can contain A, D, or blank if column 16 contains P or S:
- A indicates that the program checks that the records in the file are in ascending sequence.
- D indicates that the program checks that the records are in descending sequence.
- Blank indicates that the program does not check the record sequence.

Column 19 must contain F or blank to indicate that all records in the file must be the same length.

Columns 20 through 23 must contain the block length or blanks.

Columns 24 through 27 must contain the record length.

Columns 29 and 30 must contain the total length of the key.

Column 31 must contain A or P. A indicates that the key field is in alphanemic or zoned-decimal format. P indicates that the key field is in packed-decimal format.

Column 32 must contain I to indicate that the file is an indexed file.

Columns 35 through 38 must contain the record position in which the key field begins, if the key has only one field. If the key has noncontiguous fields, columns 35 through 38 must contain EXTK. Column 16 must contain a P, S, or F, if columns 35 through 38 contain EXTK.

Columns 40 through 43 must contain DISK.

Column 66 must contain A to indicate that records are added to the file described on this line.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Entries are also required on the output specifications:

| Columns 7 through 14 must contain the name of the file to which the records are added. |
| Column 15 must contain D, T, or E to indicate that the records are to be added at detail, total, or exception output time of the RPG program cycle. |
| Columns 16 through 18 must contain ADD to indicate that the fields coded on the following lines form the record to be added to the file named in columns 7 through 14. |
| Columns 23 through 31 can contain output indicators. |
Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

**Example of Adding Records Sequentially by Key Field**

Suppose you want to add new inventory items to the indexed inventory file created in “Example of Creating an Indexed File” on page 137. You want to add records from a transaction file that contains both new items and new shipments received of existing items. The transaction file is sorted into ascending sequence by key field (item number), and the records in the transaction file are in the same format as the records already in the inventory file.

Figure 55 on page 168 shows how to code the specifications to update the inventory file and to add new item records to the file.
ADDING RECORDS TO AN INDEXED FILE

Figure 55 (Part 1 of 2). Adding Records Sequentially by Key Field
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C*
C* Update inventory master record.
C*
C  04 MR    Z-ADDCOST    UCOST
C  04 MR    SUB QTY     ONORD
C  04 MR    ADD QTY     ONHAND
C  04 MR    MOVE DATE   LDATE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn..............................................*
O*
O* Add new item to inventory master file.
O*
OMASTER DADD 01NMR
O..................N01N02N03Field+YBEnd+PConstant/editword++++++++++...*
O RECORD  96
O  D  04 MR
O*
O* Update existing record in inventory master file.
O*
O UCOST  28
O ONHAND 41
O ONORD  46
O LDATE  62

Figure 55 (Part 2 of 2). Adding Records Sequentially by Key Field
ADDING RECORDS TO AN INDEXED FILE
Chapter 7. Using a WORKSTN File

A WORKSTN file allows a user to interact with an RPG program at a display station. That is, while the program is running, information is displayed on the screen and the user can enter data at the keyboard. Several users at different display stations can interact with the same program at the same time.

A program can use only one WORKSTN file. If a program uses a WORKSTN file, it cannot use a KEYBORD, CRT, or CONSOLE file.

You can also use a WORKSTN file to communicate with other systems through the Intersystem Communications Function of the Operating System/400 (OS/400-ICF), hereafter, referred to as ICF. For information about ICF, see ICF Programmer’s Guide.

Note: Throughout this chapter, the term device means either a display station or an ICF session.

This chapter contains six sections, organized as follows:

• The first section contains a simple example of a program that uses a WORKSTN file.
• The second section explains the two steps in all programs that use a WORKSTN file (creating the displays and coding the RPG specifications).
• The third section presents some additional ways that programs commonly use a WORKSTN file.
• The fourth section covers the differences between programs that use only one display station and those that use more than one display station.
• The fifth section discusses some advanced topics relating to WORKSTN files.
• The sixth section contains sample programs.

EXAMPLE OF USING A WORKSTN FILE

Suppose you want to create a program that allows a user to display accounts receivable information about some customers. You want the program to display the name and address of the customer, the current balance, the credit limit, the amount due in 30, 60, and 90 days, and the date of the last payment.

From the point of view of the user, this program involves three steps:

1. Seeing a display that prompts the user to enter the customer number
2. Entering the customer number in response to the prompt
3. Seeing the accounts receivable information for the customer chosen.
From the programmer’s point of view, the program involves two basic steps:

1. Creating the two displays
2. Coding the specifications for the program.

The following sections describe the two displays that the program requires, and the specifications that you must code for the program.

The Displays

The first display, which prompts the user to enter a customer number, looks like the one shown below:

```
Customer Inquiry

Please enter customer number.
Customer number
Press the Enter key to see accounts receivable information.

Press F3 to return to the main menu.
```

Figure 56. Display AR230RD1 Prompts the User to Enter a Customer Number
The second display, which shows the accounts receivable information for the customer chosen, looks like this:

<table>
<thead>
<tr>
<th>Customer Inquiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer number</td>
</tr>
<tr>
<td>Customer name</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>State and zip</td>
</tr>
<tr>
<td>Current amount due</td>
</tr>
<tr>
<td>Amount due over 30 days</td>
</tr>
<tr>
<td>Amount due over 60 days</td>
</tr>
<tr>
<td>Amount due over 90 days</td>
</tr>
<tr>
<td>Credit limit</td>
</tr>
<tr>
<td>Last amount paid</td>
</tr>
<tr>
<td>Last date paid</td>
</tr>
</tbody>
</table>

Press the Enter key to continue.

Figure 57. Display AR230RD2 Shows Accounts Receivable Information

For information on how to use the Screen Design Aid (SDA) utility to create displays see *SDA User's Guide and Reference*.

**Coding the RPG Specifications**

This program requires file description, input, calculation, and output specifications. The coding required for each of these specifications is described in this section.
File Description Specifications

This program requires the following file description specifications:

```plaintext
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRlenLKIAIvKlocEDevice+........Exit++.......A....U+.*
F*
F* The combined WORKSTN file, CUSTNMBR, is a primary file.
F* This is the main file that the program reads. All records
F* in the file have the same length. All records contain
F* 256 positions.
F*
F* The input DISK file, CUSTMAST, is an indexed file to be
F* processed randomly by key field. The key field, CUSNO, is
F* eight positions long, starting in position 2.
F* CUSTMAST is defined as a Chained input file. All records in
F* the file contain 256 positions.
F*
FCUSTNMBRCFP F 256 WORKSTN
FCUSTMASTIF F 256 256R 8AI 2 DISK
```

*Figure 58. File Description Specifications*
Input Specifications

The following input specifications describe the two files that provide input:

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.........................................................
I*
I* The WORKSTN file, CUSTNMBR, contains three types of records,
I* identified by blank( ), A, and B (columns 24 to 27).
I* These record-IDs are the IDs of user-created displays.
I*
I* Record-type blank ( ) turns on record-identifying indicator 02.
I*
ICUSTNMBRNS 02 1 C
I*
I* Record-type A turns on record-identifying indicator 03.
I* The CUSNO field is in positions 2 through 9 of the record
I* type identified by record-identifying indicator 03.
I*
I    NS 03 1 CA
I 2 9 CUSNO
I*
I* Record-type B turns on record-identifying indicator 04.
I*
I    NS 04 1 CB

Figure 59 (Part 1 of 2). Input Specifications
I*
I.................................................PFromTo++DFIELD+L1M1FrPoEq...*
I*
I* The DISK input file, CUSTMAST, also has a field called CUSNO in
I* positions 2 through 9 of each record. CUSTMAST also contains
I* accounts receivable information. The P in column 43, indicates
I* that the data in these fields is in packed-decimal format.
I*
ICUSTMASTNS  10
I            2     9 CUSNO
I           10    34 CUSNM
I            35    59 CUSA1
I           60    84 CUSA2
I           85   109 CUSA3
I          110  111 STATE
I         P  112 1160ZIPCD
I         P  131 1352AMDUE
I         P  124 1270CRLIM
I         P  155 15920VR30
I         P  160 16420VR60
I         P  140 1442LSTAP
I         P  165 16920VR90
I         P  136 1390DLTPM

Figure 59 (Part 2 of 2). Input Specifications
Calculation Specifications

This program requires only two calculation specifications:

```
**.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..**
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C* Set off error indicator 99 before the program reads a record from
C* the customer master file, CUSTMAST.
C*
C  SETOF  99
C*
C* If a record contains an A in position 1, indicator 03 is on.
C* The CHAIN operation compares the customer number entered
C* on the first display with the key field, CUSNO, in the CUSTMAST file.
C* If the customer number is not found, an error message is printed.
C* Indicator 99 is also used on the output specifications to
C* condition error messages.
C*
C  03  CUSNO  CHAINCUSTMAST  99
```

Figure 60. Calculation Specifications
Output Specifications

The following output specifications describe the two displays:

*.. 1 ...+.. 2 ...+.. 3 ...+.. 4 ...+.. 5 ...+.. 6 ...+.. 7 ..*
OName+++DFBASbsaN01N02N03Excptn...................................................*
0*
0* CUSTNMBR is the output file. The first format AR23ORD1 is a
0* detail record.
0* If a customer number is not found, that is, if the
0* record-identifying indicator 02, 99, or 04 is on, and
0* if function key 3 is not pressed, write a detail
0* record.
0*
0   CUSTNMBRD  02NKC
0   OR  99NKC
0   OR  04NKC
0.................................N01N02N03Field+YBEnd+PConstant/editword+++++++++++...*
0                     K8 'AR23ORD1'
0                      CUSNO  8
0                        99  'CUSTOMER NUMBER DOES NOT'
0                        99  ' EXIST.'
0                        99  'ENTER A DIFFERENT CUSTOM'
0                        99  'ER NUMBER.'
0*
0* If a customer number is found, that is, if the record-identifying
0* 03 is on, indicator 99 is not on, and function key 3
0* is not pressed, write the second format AR23ORD2 at detail
0* output time.
0* The edit code, L in column 38 displays a minus sign after any
0* negative balance.
0*
0                 D   03N99NKC
0                       K8 'AR23ORD2'
0                        CUSNO  8
0                        CUSNM  33
0                        CUSA1  58
0                        CUSA2  83
0                        CUSA3 108

Figure 61 (Part 1 of 2). Output Specifications
Ending the Program

A program that uses a WORKSTN file, as this example does, can end in any of several ways:

- One way to end the program is simply to turn on the last-record indicator (LR).
- Another way is to have the user press a function key. In this example, function key 3 is used. Both display formats are conditioned so that they are not written after function key 3 is pressed. On the next input operation, the WORKSTN file reaches end of file.

  - If the WORKSTN file is a primary file, as in this example, the program automatically turns on the last-record indicator at input time of the next program cycle, and the program goes to end of job.
  - If the WORKSTN file is a demand file, and if an indicator is coded in columns 58 and 59 of the calculation specification containing the READ operation for the file, the indicator turns on.

For more information about reaching the end of a WORKSTN file, see “Reaching End of File for a MRT Program” on page 208.

This concludes the simple example of using a WORKSTN file. Although the example does not show how to change the information displayed, WORKSTN files do allow you to interact with the program. The other sample programs at the end of this chapter show how to use a WORKSTN file to update data.
CREATING THE DISPLAY FORMATS

STEPS IN USING A WORKSTN FILE

There are two general steps in using a WORKSTN file:

1. Create the format of each display from which your program will read input and to which your program will write output.
2. Code the necessary file description, input, calculation, and output specifications for your program.

The reason for creating the displays first is that the format of each display (that is, the location and characteristics of each field on the display) determines the coding required for the input and output specifications for the WORKSTN file.

Creating the Display Formats

Creating a display format includes designing the format, entering the specifications for the format, and compiling the specifications for the format. For a complete explanation of how to create display formats, see SDA User’s Guide and Reference.

There are three ways to create a display format:

- Use the Screen Design Aid (SDA) utility to design the format and to enter and compile the specifications for the format. For information about using SDA, see SDA User’s Guide and Reference.

- Use the display format layout sheet and the display format S and D specifications to design the format. Use the source entry utility (SEU) to enter the specifications for the format.

  You can also use the CL command CRTS36DSPF or the System/36 FORMAT procedure to create a display file using your S and D specifications.

- In the System/36 Environment you can use the $MAINT utility if you wish, and then use the $SFGR utility to compile the specifications for the format.

CODING THE RPG SPECIFICATIONS

SDA is the easier and recommended way to create a display format. SDA offers two advantages:

- It allows you to design formats right on the screen, so you do not need to fill out the display format S and D specifications.
- It allows you to choose an option that creates file description and input specifications for the WORKSTN file, so you do not need to code those RPG specifications for that file.

Coding the RPG Specifications

File Description Specifications

To use a WORKSTN file, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain C to indicate that the file is a combined (input and output) file.

Column 16 must contain P (primary) or D (demand) to indicate how the program uses the file:

- If column 16 contains P, the WORKSTN file is automatically read during input time of the RPG program cycle. Record-identifying indicators are automatically set off at input time of the program cycle. If the WORKSTN file is a primary file, no secondary files are allowed.
- If column 16 contains D, you must code a READ operation in the calculation specifications in order to read the file. Record-identifying indicators are not set off when the READ operation occurs, so it is possible to have more than one record-identifying indicator on if the WORKSTN file is a demand file.

Column 19 must contain F or blank to indicate that all the records in the file have the same length.

Columns 20 through 23 must be blank. The block length equals the record length.

Columns 24 through 27 must contain the length of the longest record. This number is equal to the highest end position coded on the input or output specifications. The maximum record length is 9999.
Columns 40 through 46 must contain the device name WORKSTN.

Columns 71 and 72 can contain an external indicator (U1 through U8).

**Continuation-Line Options**

The file description specifications can also include one or more continuation lines. Continuation lines are coded on the lines immediately following the file description specification for a WORKSTN file. Continuation lines are used to provide additional information about the file.

Column 53 must contain K to identify this as a continuation line.

Columns 54 through 58 must contain the continuation-line option. Valid entries for the continuation-line option are NUM, SAVDS, IND, SLN, FMTS, ID, INFSR, INFDS, and CFILE.

Columns 60 through 65 (columns 60 through 67 if the option is FMTS or CFILE) must contain the value for the continuation-line option.

Figure 62 shows sample values coded for each option.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+........Exit++......A....U+.*
```

```plaintext
FWSINPUT CP F WORKSTN
F KNUM  4
F KSAVDS SAVE
F KIND 06
F KSLN LINENO
F KFMTS FORMATS
F KID WSID
F KINFDS EXCDS
F KINFDS ERRSR
F KCFILE COMMFILE
```

*Figure 62. Continuation-Line Options for a WORKSTN File*

Continuation-line options are explained in the following paragraphs.

**NUM**

The entry coded in columns 60 through 65 for the NUM option indicates the maximum number of devices that can use this file at the same time. The maximum number that can be specified is 251. If no number is coded, 1 is assumed. If a number is coded, it must be at least as large as the sum of the number of acquired devices plus whatever you specified in the MRTMAX option when compiling from the CRTS36RPG command, the CRTS36RPT command, or whatever you enter in response to the prompt *Maximum number of requesting display stations* on the RPGC, or AUTOC procedure.

**Note:** If the value for NUM is greater than 1, use caution when updating a file (see “Updating Disk Files in a MRT Program” on page 206).
SAVDS
The name coded in columns 60 through 65 for the SAVDS option identifies the data structure that is saved and restored for each device. This data structure allows you to save the contents of a field that is unique to each display station. Therefore, it allows you to save your place in the program while another requester is using the program. For example, it can contain a field that is used to accumulate the number of records read or to store a field that is not used until later cycles, such as a credit limit. The data structure cannot be a display station local data area, a compile-time array, or a prerrun-time array. If SAVDS is not coded, no data areas are exchanged.

IND
The entry coded in columns 60 through 65 for the IND option indicates the number of indicators, beginning with 01, that are saved and restored by the display station. IND allows you to save the state of an indicator that is unique to the display station. Therefore, it allows you to save your place in the program while another requester is using the program. For example, you can use separate error indicators or security clearance indicators for each display station. If IND is not coded or if NUM equals 1, no indicators are saved and restored.

Indicators may need resetting in the program; they are not always reset by RPG in time to be useful to the programmer.

The following types of fields and indicators do not need to be placed in SAVDS and IND:

- Work fields that are used during one cycle (between input operations for the WORKSTN file), but can then be destroyed
- Job fields that are used by all display stations but are not destroyed

For SAVDS and IND, only one copy of the data structure and indicators is available at a time. The indicators and data structure that are available are those associated with the device from which the last input was read. The data structure and indicators that are available change each time the program does an input operation (either by the RPG program cycle for a primary file or by the READ operation code for a demand file). On an input operation, the program writes the present copy of the data structure and indicators in the program to a save area for the device from which the previous input was read. Then, for the device from which the current input is being read, the program writes the data structure and indicators from the save area associated with the device to the program SAVDS and IND areas. After the first input operation for each device, all the restored indicators will be off and all the fields in the SAVDS data structure will be blank.

SLN
The entry coded in columns 60 through 65 for the SLN option identifies a two-digit numeric field whose value determines the line on the screen at which the display is to begin if a variable starting-line number was specified in SDA or in the display format S specifications. If a variable starting-line number is not specified, all displays having a variable starting-line number begin on line 01.
FMTS
The name coded in columns 60 through 67 for the FMTS option identifies the
display file containing the display formats. If a name is not entered, the compiler
assumes that the name of the display file is the same as the program name with
FM added to the end of the name. The constant "NONE" coded in columns 60
through 64 indicates that the only formats in this program are system-supplied ICF
formats, or user-defined communication formats.

ID
The name coded in columns 60 through 65 for the ID option identifies a two-
character alphanumerical field that contains the identification of the device that supplied
the record currently being processed in this file. You do not have to code this field
as an input or result field. The ID field is updated whenever a record is read from
the WORKSTN file. Therefore, it always contains the identification of the device
from which the last record was read (unless your program moves a different identifi-
cation into the ID field). If the NUM option has a value of more than 1, you can
direct output to various devices by changing the value in the ID field to the identifi-
cation of another device in the file.

Display station identifiers are in the form AX, where A is an alphabetic character (A
through Z, or one of the special characters #, @, or $) and X is any character. If
the OCL WORKSTN statement exists for the display station, the value of ID is iden-
tical to the value of the SYMID parameter on the WORKSTN statement as long as
the program is not placed in the job queue or evoked.

ICF session identifiers can be in either of two formats: NN or NA, where N is
numeric (0 through 9) and A is alphabetic (A through Z, #, @, or $). If the format is
NA, an OCL SESSION statement must be specified with a SYMID parameter
whose value is also in the NA format.

INFSR
The name coded in columns 60 through 65 for the INFSR option identifies the
WORKSTN exception/error-processing subroutine. For more information about this
subroutine, see "Handling Exceptions and Errors" on page 191.

INFDS
The name coded in columns 60 through 65 for the INFDS option identifies the
WORKSTN file information data structure. For more information about this data
structure, see "Handling Exceptions and Errors" on page 191.

CFILE
The name coded in columns 60 through 67 for the CFILE option identifies an ICF
communication file. This file associates a WORKSTN file with an ICF communi-
cation file defined by the user. For more information about creating ICF communica-
tion files, see Communications: Intersystem Communications Function
Programmer's Guide.
## Input Specifications

Code entries in the unshaded columns of the input specifications as shown below:

### In the first line:
- Columns 7 through 14 must contain the name of the WORKSTN file unless the preceding input specifications are for the same file.
- Columns 14 and 15 can contain OR, or columns 14 through 16 can contain AND, to indicate a relationship between record-identifying indicators or record types on consecutive lines.
- Columns 15 and 16 must contain a numeric or alphabetic entry. A numeric entry indicates that the program checks the sequence of input records. An alphabetic entry indicates that the program does not check the sequence of input records.
- Column 17 can contain 1 or N if columns 15 and 16 contain a numeric entry. 1 indicates that only one record of this type can be present in the sequenced group. N indicates that one or more records of this type can be present in the sequenced group.
- Column 18 can contain blank or O. Blank indicates that the record type must be present if columns 15 and 16 contain a numeric entry. O indicates that the record type is optional if columns 15 and 16 contain a numeric entry.
- Columns 19 and 20 can contain a record-identifying indicator.
- Columns 21 through 41 can contain record identification codes.

### In the second line of the input specifications:
- Column 43 can contain blank, P, or B:
  - Blank indicates that the field is in zoned-decimal format or is alphameric.
  - P indicates that the field is in packed-decimal format. Use the P entry only if the input is from an ICF session.
  - B indicates that the field is in binary format. Use the B entry only if the input is from an ICF session.
- Columns 44 through 51 must contain the location of the field in the input record. These entries do not refer to the location of the field as it is displayed. The input fields are placed in the input record in the order in which they are described in SDA (top to bottom, left to right) or in the order in which they are described in the display format specifications. However, you can use the line number and horizontal position columns on the display format specifications to change the order in which the fields appear on the display. Figure 63 on page 186 shows the relationship between the display format specifications and the RPG input specifications.
CODING THE RPG SPECIFICATIONS

- Column 52 can contain a digit to indicate the number of decimal positions in a numeric field named in columns 53 through 58.
- Columns 53 through 58 must contain the name of a field, array, or array element in the input record.
- Columns 63 and 64 can contain a field-record-relation indicator.
- Columns 65 through 70 can contain field indicators.

**Figure 63. Relationship between the Display Format Specifications and the RPG Input Specifications**
Each record, including the blank record at the first read to a device, should be identified on the input specifications. For displays, specify a nondisplayed, protected output/input field as the record code on each display format.

The first input record read from a device is blank except in the following cases:

- The program reads a format that was displayed by a procedure (PDATA-YES was specified on the OCL PROMPT statement in the procedure). For more information on creating procedures, see Making Procedures in the System Reference for the System/36 Environment.
- The program reads a format that was displayed by a different program (see “Reading Data From a Display Shown by a Previous Program” on page 201).
- The program is a single requesting terminal (SRT) program and writes a display before reading for the first time.
- The procedure has a PDATA-YES attribute and data was specified on the procedure invocation.

### Output Specifications

Code entries in the unshaded columns of the output specifications as shown below:

![Output Specifications Table](image)

**On the first line:**

- Columns 7 through 14 must contain the name of the WORKSTN file unless the preceding output specifications are for the same file.
- Columns 14 and 15 can contain OR, or columns 14 through 16 can contain AND, to indicate a relationship between output indicators on consecutive lines.
- Column 15 must contain H (heading), D (detail), T (total), or E (exception) to indicate the type of record written.
- Column 16 can contain R to indicate that the device is released from the program after output to that device occurs. If OR is coded in column 14 and 15, column 16 must contain an R for each OR line.
- Columns 23 through 31 can contain output indicators other than the first-page (1P) indicator. For information about output indicators, see Chapter 13, “Using Indicators.”
- Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

**On the second line:**

- Columns 40 through 43 must contain K and the number of characters in the name of the display format.
• Columns 45 through 54 must contain the name of the display format, enclosed in apostrophes. One and only one format name is required for each output record for a WORKSTN file.

On the third and following lines:

• Columns 23 through 31 can contain output indicators other than the first-page (1P) indicator.

• Columns 32 through 37 can contain the name of an output field. The fields must be coded on the output specifications in the order in which they are described on the display format S and D specifications.

• Column 38 can contain an edit code. For more information about edit codes, see Chapter 16, “Editing Numeric Fields.”

• Column 39 can contain B or blank. B indicates that the field named in columns 32 through 37 is set to blanks or zero after it is written.

• Columns 40 through 43 must contain the end position of the field in the output record. The end position does not refer to the end position of the field as it appears on the display. Use SDA or the output from the $SFGR utility as a guide when coding the end position of the field (see Figure 64 on page 189).

• Column 44 can contain blank, P, or B:
  – Blank indicates that the field is in zoned-decimal or alphameric format.
  – P indicates that the field is in packed-decimal format. Use the P entry only if the output is to an ICF session.
  – B indicates that the field is in binary format. Use the B entry only if the output is to an ICF session.

• Columns 45 through 70 can contain an edit word or a constant. For information about edit words, see Chapter 16, “Editing Numeric Fields.”
Common Processing Variations

RPG programs that use a WORKSTN file commonly include one or more of the following processing variations:

- Using function keys
- Handling exceptions and errors
- Reading data from a display shown by a previous program.
Using Function Keys

There are 24 function keys. Each one corresponds to a separate function-key indicator:

<table>
<thead>
<tr>
<th>Function-Key Indicator</th>
<th>Corresponding Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>KA</td>
<td>1</td>
</tr>
<tr>
<td>KB</td>
<td>2</td>
</tr>
<tr>
<td>KC</td>
<td>3</td>
</tr>
<tr>
<td>KD</td>
<td>4</td>
</tr>
<tr>
<td>KE</td>
<td>5</td>
</tr>
<tr>
<td>KF</td>
<td>6</td>
</tr>
<tr>
<td>KG</td>
<td>7</td>
</tr>
<tr>
<td>KH</td>
<td>8</td>
</tr>
<tr>
<td>KI</td>
<td>9</td>
</tr>
<tr>
<td>KJ</td>
<td>10</td>
</tr>
<tr>
<td>KK</td>
<td>11</td>
</tr>
<tr>
<td>KL</td>
<td>12</td>
</tr>
<tr>
<td>KM</td>
<td>13</td>
</tr>
<tr>
<td>KN</td>
<td>14</td>
</tr>
<tr>
<td>KP</td>
<td>15</td>
</tr>
<tr>
<td>KQ</td>
<td>16</td>
</tr>
<tr>
<td>KR</td>
<td>17</td>
</tr>
<tr>
<td>KS</td>
<td>18</td>
</tr>
<tr>
<td>KT</td>
<td>19</td>
</tr>
<tr>
<td>KU</td>
<td>20</td>
</tr>
<tr>
<td>KV</td>
<td>21</td>
</tr>
<tr>
<td>KW</td>
<td>22</td>
</tr>
<tr>
<td>KX</td>
<td>23</td>
</tr>
<tr>
<td>KY</td>
<td>24</td>
</tr>
</tbody>
</table>

Note: The keyboard keys used to set these indicators may vary, depending on the type of keyboard that you have.

You can use SDA or column 28 and columns 64 through 79 of the display format S specifications to allow the use of a function key in a program (see SDA User's Guide and Reference for more information). You can use any of the 24 function keys with a WORKSTN file. If a user presses a function key that can be used in that program, the corresponding function-key indicator turns on. You can then use the function-key indicator to condition calculation and output operations.

For example, you can specify that the user press function key 2 (rather than the Enter key) when the last item for an invoice has been typed at the display station. You can then use function-key indicator KB in the program to condition calculation operations and output operations, such as presenting the next display.

If the user presses a function key that is not allowed by the format, error message CPF5024 Function key not allowed, is displayed. The user can press the Error Reset key and then press the correct function key.

For a discussion of how to determine if a function key was pressed, see “Coding the INFDS Data Structure” on page 194.

Note: Each time an input operation occurs from a WORKSTN file, all function-key indicators are reset, unless an exception or error occurs during the input operation.
To document the use of the function keys for the user, you can use the template assignment form on the *New User’s Guide*.

### Handling Exceptions and Errors

For a display station, the term **exception** means input from a command key (Print, Roll Up, Roll Down, Clear, Help, or Home) to the program. This input is an exception because no data is read into the program, so the program cannot go through normal processing such as record identification.

To allow the use of command keys, you must do three things:

- You must define the command keys from which you will allow input. You define them with either SDA or column 27 and columns 64 through 79 of the display screen format S specifications when you create the display for your program. No specific command is automatically associated with any command key, so you can define the actions you want for each one. For information about how to define command keys, see *SDA User’s Guide and Reference*.

- Your program must include the file information data structure (INFDS), which contains an indication of the exception or error. If the program does not contain the INFDS, it cannot tell if one of the command keys was pressed.

- Your program must also include either the exception/error-processing subroutine (INFSR) or a resulting indicator in columns 56 and 57 of the calculation specification for a READ operation. If neither the INFSR subroutine nor resulting indicators are specified, a message stating that the function key is not allowed will appear on your screen. Control will return to the program when the user presses a correct key.

- When a System/36-Compatible RPG II program, which is capable of processing command keys opens a display file that has command keys enabled for processing, the enabled command keys process as expected. If this program opens the same display file after it calls another RPG II program that is not capable of processing command keys, command key processing for the display file will be disabled. When this program returns to the calling program, the calling program will now have its command keys disabled until a different format is written to the screen.

The term **error** means an error that occurs during an input or output operation (a program cycle input/output operation or an ACQ, REL, NEXT, READ, or EXCPT operation).

If an exception or error occurs while your program is processing a WORKSTN file, you can use the INFDS and either the INFSR or resulting indicators in columns 56 and 57 of the calculation specification for an ACQ, REL, NEXT, POST, or READ operation to control the program logic.

First, your program can check the information in the INFDS. The INFDS contains an identification of the exception or error that occurred and an identification of the operation for which it occurred. The INFDS also contains status information on normal conditions (not exceptions or errors) such as if a function key was pressed or if end of file has occurred. The information in the INFDS is updated for each ACQ, REL, NEXT, POST, READ, or EXCPT operation or for each input or output operation in the program cycle.
Then, using that information in the INFDS, you can determine those exceptions or errors that you want to handle in the INFSR subroutine in your program and those that you want RPG to handle. Control automatically passes to the INFSR subroutine, if specified, under the following conditions:

- If an exception or error occurs at input time of the program cycle for a primary file, at exception output time, or at normal output time
- If an exception or error occurs on an ACQ, REL, NEXT, POST, or READ operation that does not have an indicator coded in columns 56 and 57.

In addition, the INFSR subroutine can be called directly from detail or total calculations by the EXSR operation.

The indicator specified in columns 56 and 57 for an ACQ, REL, NEXT, POST, or READ operation turns on if an exception or error occurs on that operation. Control then passes to the next operation that can be processed in the program. In order to use the INFSR subroutine, the next operation must be an EXSR operation that calls the INFSR subroutine. Control does not automatically pass to the INFSR subroutine if the EXSR operation is not specified.

The relationship between INFDS, INFSR, and indicators in columns 56 and 57 is shown in Figure 65 on page 193. These exception/error-handling techniques are optional and can be used individually or in any combination. However, if command keys are allowed for the program, the INFDS data structure and either the INFSR subroutine or an error indicator on the READ operation must be specified. You can choose the technique that best suits your own program.

On a System/36, an RPG II program that processes a WORKSTN exception/error-processing subroutine (INFSR) and ends because the ENDSR operation for that subroutine specifies *CANCL, gives the appearance of ending normally. (No error message is issued.)

A System/36-Compatible RPG II program on the AS/400 issues a RPG9901 error message when cancelled the same way. That error causes a SYS7375 message to be issued, (which requires a reply), when the RPG II program is called from the System/36 environment command entry display or from an OCL procedure.

To avoid receiving the RPG9901 message, code the following operations immediately before the ENDSR operation in the subroutine.

```
SETON LR
RETRN
```
HANDLING EXCEPTIONS AND ERRORS

Update
*STATUS in INFDS

Exception/error

Yes

Error indicator specified in cols 56 & 57

No

Continue

Set on indicator

Continue

INFSR specified

No

INFSR

specified

Run INSFR subroutine

Yes

Factor 2 blank on ENDSR

No

RPG error handling (program stops). If INFSR was called by EXSR, returns the next sequential instruction.

Go to point in RPG cycle specified by factor 2 entry on ENDSR

*GETIN (beginning of new cycle)

*DETC (detail calculations)

*CANCL (RPG9991 ERROR)

MESSAGE)

1 For the exact point in the cycle that is specified by these keywords, see ENDSR operation section in Chapter 7.

Figure 65. Handling Exceptions and Errors in a WORKSTN File
HANDLING EXCEPTIONS AND ERRORS

Coding the INFDS Data Structure

Note: Overwriting the INFDS data structure may cause unexpected results in subsequent error handling, and is therefore not recommended.

File Description Specifications

Three entries are required on the continuation line of the file description specifications for the WORKSTN file:

Column 53 must contain K to indicate that this is a continuation line.

Columns 54 through 58 must contain INFDS.

Columns 60 through 65 must contain the name of the data structure to be used as the INFDS.

Input Specifications

The following input specifications are required for the INFDS:

On the first line of the input specifications:

- Columns 7 through 12 must contain the name of the data structure. The name must be the same as the name coded in columns 60 through 65 of the file description specifications continuation line with the INFDS keyword.
- Columns 19 and 20 must contain DS to indicate data structure.
On the second and following lines:

- Columns 44 through 51 must contain a keyword that identifies the location of the subfields containing the status information. The valid keywords are *STATUS, *OPCODE, *RECORD, *SIZE, *MODE, *INP, and *OUT. The keywords are not labels and cannot be used to refer to the subfields.
- Columns 53 through 58 must contain a subfield name. You can use the subfield names to refer to the subfields.

Figure 66 shows all the subfield keywords and subfield names.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC........................................*  
I*  
I* The keywords and predefined From and To locations in columns 44  
I* through 51 define the location and size of the subfields in the  
I* INFDS data structure, which contain the status information.  
I* Field names must be assigned in columns 53 through 58 so that the  
I* subfields can be referred to in a program.  
I*  
IEXCPDS    DS  
I..........................PFromTo++DField+L1M1FrPoNeEq...*
I              *STATUS   STATUS  
I              *OPCODE   OPCODE  
I              *RECORD   FMTNM  
I              *SIZE     SIZE  
I              *MODE     MODE  
I              *INP      INP  
I              *OUT      OUT  
I            23  26 RCODE  
```

Figure 66. Subfield Keywords for the INFDS Data Structure
**STATUS Keyword**: The name in columns 53 through 58 for the *STATUS keyword identifies a five-digit numeric subfield with zero decimal positions within the INFDS data structure. This subfield contains a code that identifies the exception or error that occurred. The codes are as follows:

<table>
<thead>
<tr>
<th>Exception/Error Conditions</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command Keys</strong></td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td>01121</td>
</tr>
<tr>
<td>Roll Up</td>
<td>01122</td>
</tr>
<tr>
<td>Roll Down</td>
<td>01123</td>
</tr>
<tr>
<td>Clear</td>
<td>01124</td>
</tr>
<tr>
<td>Help</td>
<td>01125</td>
</tr>
<tr>
<td>Home</td>
<td>01126</td>
</tr>
<tr>
<td><strong>Error Status Codes</strong></td>
<td></td>
</tr>
<tr>
<td>Input was rejected because the buffer is too small</td>
<td>01201</td>
</tr>
<tr>
<td>Permanent I/O error occurred</td>
<td>01251</td>
</tr>
<tr>
<td>Maximum number of requesting and/or acquired devices</td>
<td>01261</td>
</tr>
<tr>
<td>Device is busy</td>
<td>01271</td>
</tr>
<tr>
<td>Display station was externally released from the MRT program. See 2400 return code on page 198.</td>
<td>01275</td>
</tr>
<tr>
<td>Input was rejected, device is not available, or device was not found</td>
<td>01281</td>
</tr>
<tr>
<td>Attempt to acquire a device already owned</td>
<td>01285</td>
</tr>
<tr>
<td>Other input/output errors</td>
<td>01299</td>
</tr>
<tr>
<td>Change of direction was received with no data</td>
<td>01311</td>
</tr>
<tr>
<td>Request for change of direction was received</td>
<td>01321</td>
</tr>
<tr>
<td>Time interval ended</td>
<td>01331</td>
</tr>
</tbody>
</table>

If an exception or error occurs, RPG bypasses the move field logic, no fields are changed, no record-identifying indicators are turned on, and the function-key indicators are not reset.

You also have access to the following successful status codes that are placed in *STATUS after any input/output operation:

<table>
<thead>
<tr>
<th>Normal Condition</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>No exception/error</td>
<td>00000</td>
</tr>
<tr>
<td>Any of the 24 function keys</td>
<td>00002</td>
</tr>
<tr>
<td>End-of-file on a read (input)</td>
<td>00011</td>
</tr>
</tbody>
</table>

Any code in *STATUS greater than 99 is considered an exception or error, and the resulting error indicator, if specified, turns on. If no resulting error indicator is speci-
fied on an ACQ, REL, NEXT, POST, or READ operation or if the program cycle is at input time for a primary file, at exception output time, or at normal output time, control automatically passes to the INFSR subroutine.

**OPCODE Keyword:** The name coded in columns 53 through 58 for the *OPCODE keyword identifies a five-character alphameric subfield within the INFDS data structure. This subfield contains a value that identifies the operation that was being performed when the exception or error occurred. The value inserted in the *OPCODE subfield is READ, ACQ, REL, NEXT, POST, or WRITE (for output operations). A value is inserted in the *OPCODE subfield when a value greater than 99 is placed in *STATUS.

**RECORD Keyword:** The name coded in columns 53 through 58 for the *RECORD keyword identifies an eight-character alphameric subfield within the INFDS data structure. This subfield contains the format name if *OPCODE contains WRITE. If *OPCODE does not contain WRITE, *RECORD is blank.

**SIZE Keyword:** The name coded in columns 53 through 58 for the *SIZE keyword identifies a 4-digit numeric subfield within the INFDS data structure. The digits in this subfield indicate the size of the display. The subfield contains either 1920 (24 rows x 80 columns = 1920 characters), or, if you are using a 3180 model 2 workstation, 3564 (27 rows x 132 columns = 3564 characters). The subfield is reset each time the POST operation occurs.

**MODE Keyword:** The name coded in columns 53 through 58 for the *MODE keyword identifies a two-digit numeric field that indicates that this is a double-byte character session.

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>This is a double-byte character session.</td>
</tr>
<tr>
<td>00</td>
<td>This is not a double-byte character session.</td>
</tr>
</tbody>
</table>

**INP Keyword:** The name coded in columns 53 through 58 for the *INP keyword identifies a two-digit numeric field that indicates if the DBCS or the alphameric/Katakana keyboard is being used with this display station.

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>The DBCS keyboard is being used.</td>
</tr>
<tr>
<td>00</td>
<td>The alphameric/Katakana keyboard is being used.</td>
</tr>
</tbody>
</table>

**OUT Keyword:** The name coded in columns 53 through 58 for the *OUT keyword identifies a two-digit numeric field that indicates if this display station's screen is capable of displaying DBCS characters.

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>The screen can display DBCS characters.</td>
</tr>
<tr>
<td>00</td>
<td>The screen cannot display DBCS characters, or the display is not output-capable.</td>
</tr>
</tbody>
</table>

**Positions 23-26:** Positions 23 through 26 of the INFDS data structure contain the four-character return code for WORKSTN files. These positions are filled in for all WORKSTN files. This subfield is similar to *STATUS except that *STATUS values are the same for RPG on all IBM systems. Positions 23 through 26 must be coded
handling exceptions and errors

as the beginning and ending positions of the return-code subfield on the input specifications (see Figure 66 on page 195). Refer to Concepts and Programmer’s Guide for the System/36 Environment for information on return codes for communications. Table 4 shows the RPG return codes for display stations.

Note: This subfield is not updated for a *STATUS value of 01261 because RPG does not call the system.

The subfield is referred to by the name coded in columns 53 through 58 of the input specifications.

Table 4. WORKSTN Return Codes

<table>
<thead>
<tr>
<th>Major Return Code</th>
<th>Minor Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Operation was successful</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>Program successfully acquired a new requester</td>
</tr>
<tr>
<td>02</td>
<td>00</td>
<td>Controlled job cancel is pending</td>
</tr>
<tr>
<td>08</td>
<td>00</td>
<td>Program attempted to acquire a display station that was already acquired; no error</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>Input operation was attempted but no input was available (end of file)</td>
</tr>
<tr>
<td>24</td>
<td>00</td>
<td>Display station was released from the MRT program because:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The user selected option 2 after pressing the System Request Key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The interactive session was ended, for example, by issuing the ENDJOB CL command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A permanent I/O error was encountered, for example, by powering off the display station or dropping the remote line.</td>
</tr>
<tr>
<td>28</td>
<td>00</td>
<td>Operation was rejected because the program previously released the single requester</td>
</tr>
<tr>
<td>32</td>
<td>00</td>
<td>Acquire failed because the user is unauthorized</td>
</tr>
<tr>
<td>38</td>
<td>00</td>
<td>Attempt to acquire the display station failed</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>Permanent device error occurred</td>
</tr>
</tbody>
</table>

For more information about major and minor Return Codes, see the Concepts and Programmer’s Guide for the System/36 Environment.

Coding the INFSR Subroutine

The INFSR subroutine can process any function normally allowed in calculations, including exits to other calculation subroutines and input/output operations. The INFSR subroutine returns control to the point specified by the optional factor 2 entry for the ENDSR operation.
File Description Specifications

Three entries are required on the continuation line of the file description specifications for the WORKSTN file:

- **Column 53** must contain K to indicate that this is a continuation line.
- **Columns 54 through 58** must contain INFSR.
- **Columns 60 through 65** must contain the name of the calculation subroutine that is processed if a WORKSTN exception or error occurs on an ACQ, REL, NEXT, POST, or READ operation for which no error indicator is coded in columns 56 and 57, or of program cycle input/output operations.

Calculation Specifications

The INFSR subroutine requires the following entries in the calculation specifications:

**BEGSR Operation:** Factor 1 of the BEGSR operation must contain the name of the exception/error-processing subroutine. This must be the same as the name coded in columns 60 through 65 of the file description specification continuation line.

**ENDSR Operation:** Factor 2 of the ENDSR operation can contain a literal, an array element, or a field name that identifies the point to which the INFSR returns control. INFSR can be accessed using EXSR if factor 2 contains a blank or any of the allowed literals: *GETIN, *DETC, or *CANCL.
If factor 2 contains blanks and if the INFSR subroutine was called directly by the EXSR operation, control returns to the operation following the EXSR statement.

If factor 2 contains blanks and if the subroutine was called indirectly because of an exception (that is, a command key was pressed), the subroutine is run, and a message is displayed on the user’s screen stating that the function key is not allowed. If the subroutine was called indirectly because of an error, the subroutine is run, and control is passed to the RPG error-handling routine, which causes the program to halt and prompts the user at the display station to choose the appropriate option.

If factor 2 contains a literal, the literal must be one of the following keywords. The keyword must be enclosed in apostrophes.

- **GETIN**: Control returns to the beginning of a new cycle.
- **DETC**: Control returns to the beginning of detail calculations within the same cycle.
- **CANCL**: The RPG9901 error message is issued, which causes a SYS7375 message.
- **A literal value of blanks is the same as no entry.**

If an exception or error occurs on an operation that attempts to read data from a file and the exception/error-handling subroutine receives control, you must ensure that an output operation to the WORKSTN file occurs before another READ occurs. Two consecutive READ operations cannot be processed to the WORKSTN file. For example, if the WORKSTN file is a primary file and the exception/error subroutine ENDSR statement specifies a return point of *GETIN, an output operation must be done to the file before the ENDSR operation occurs. The *GETIN routine will attempt to read from a WORKSTN primary file.

If factor 2 is an array element or field name, it must identify a six-character alphabetic field that contains one of the keywords *GETIN, *DETC, or *CANCL or blanks that define the return point from the subroutine. By specifying the return point in a field, you can use the subroutine to process all types of exceptions and errors that occur in the WORKSTN file.

If a field name or array element is specified in factor 2, the field or array element is set to blanks upon each exit from the subroutine. Therefore, you can control the return point of the INFSR within the program by placing the return point in the field that best fits the particular exception or error that occurred. If factor 2 is blank and if the subroutine was called indirectly because of an exception, the subroutine runs, and a message stating that the function key pressed is not allowed will be displayed on the user’s screen. If factor 2 is blank and if the INFSR subroutine was called indirectly because of an error, the subroutine runs, and control passes to the RPG error handling routine. If factor 2 is blank and if the INFSR subroutine was called directly by the EXSR operation, control returns to the calculation immediately following the EXSR operation.
Reading Data From a Display Shown by a Previous Program

When one program in a procedure uses a normal output operation to show a display and then goes to end of job or releases the display station, the next program in the procedure can read that display. The user at the display station can enter data into the display while the second program is initiating. When the user presses the Enter key, the data entered into the display is sent to the second program. This technique is called a **read under format**.

There are two ways to do a read under format:

- One way is to use the OCL PROMPT statement to display the format. The PROMPT statement is in the procedure, so no other program is involved. For information about the PROMPT statement, see the *System Reference for the System/36 Environment*.

- The other way is to display the format in one program, then end that program, and load and run a second program, which reads the format. This method does not involve any OCL statements to display the format.

Using One or More Devices

A WORKSTN file allows users to interact with the program from one or more display stations at the same time. A program that can process requests from only one display station or ICF session is called a Single Requester Terminal (SRT) program. A program that can process requests from more than one display station or ICF session at the same time, using a single copy of the program, is called a Multiple Requester Terminal (MRT) program.

A program is a SRT or a MRT depending on the number that you enter for the MRTMAX parameter of the CRTS36RPG or CRTS36RPT procedures, or in the System/36 Environment in response to the prompt **Maximum number of requesting display stations** on the RPGC, or AUTOC procedure when you compile the program. Entering a parameter value of 0 or blank specifies a SRT program, whereas entering a parameter value of 1 to 99 specifies a MRT program.

For information about these procedures, see Chapter 3, “Compiling an RPG II Program” and Chapter 26, “The Auto Report Feature.” For a complete description of SRT and MRT programs, see the *Concepts and Programmer’s Guide for the System/36 Environment*.

Using a SRT Program

Although a SRT program allows only one requester, it is possible for more than one requesting display station or ICF session to use a SRT program at the same time if each display station or ICF session uses a separate copy of the program.

Using a MRT Program

Each requester of a MRT program uses the same copy of the program. The first requester loads and starts the program. If the WORKSTN file is a primary file, each succeeding requester begins to use the program at the beginning of an input cycle. If the WORKSTN file is a demand file, each succeeding requester begins to use the program when the READ operation for the WORKSTN file occurs. If the
program is handling the maximum number of requesters, OS/400 places the next requester of the program in a queue. When the program releases one of its requesters, the program can process the queued request.

If the program is called by more than one requester, the first requester:

- Initiates the program
- Provides the external indicators (U1 through U8)
- Provides the display station local data area for the data structure defined by a U in column 18 of the input specifications.

Each requester can access any display station local data area and external indicators attached to the program by using SUBR20 and SUBR21. For information about SUBR20, see “Setting and Restoring External Indicators (SUBR20)” on page 204. For information about SUBR21, see “Reading and Writing the Local Data Area for a Display Station (SUBR21)” on page 205.

Program error messages go to the message queue QSYSOPR.

Requesters can leave the program without suspending the program or other devices.

If the Sys Req key is pressed while a MRT program is running, the processing of information is suspended at the display station where the Sys Req key was pressed. The program continues to process information from other requesters. However, if the program must write a display to the suspended requester, the entire program is suspended (all requesters are suspended) until the requester is no longer suspended and the writing is complete.

File Description Specifications

A MRT program can include one or more continuation lines for the WORKSTN file description specifications. For more information, see “Continuation-Line Options” on page 182.

Calculation Specifications

Two operations that are commonly used in MRT programs are NEXT and REL.

NEXT Operation

The NEXT operation forces the next input to the program to come from the device specified in factor 1. Factor 1 must contain the name of a two-character field that contains the device identification or a two-character alphameric literal that is the device identification. Factor 2 must contain the name of the WORKSTN file.

If NEXT is coded more than once between input operations, only the last operation occurs. If you code an indicator in columns 56 and 57, it turns on if an exception or error occurs on the NEXT operation.

If columns 56 and 57 do not contain an indicator and an exception or error occurs, the program halts unless the INFSR subroutine is specified. If the INFSR subroutine is specified, the subroutine automatically receives control when an exception or error occurs. For more information on the INFSR subroutine, see “Handling Exceptions and Errors” on page 191.
**REL Operation**

The REL operation releases the device specified in factor 1 from the program. Factor 2 must contain the name of the WORKSTN file.

You can release either a requester or an acquired device with the REL operation code. The program releases the specified device when the REL operation occurs during calculations unless the device is the requester of a SRT program. If the device specified in factor 1 is the requester of a SRT program, the device is released at end of job, not when the REL operation occurs in calculations, but the device is no longer available to the program except to log messages. If the device is a display station, the display station is no longer available to the program.

If you code an indicator in columns 56 and 57, it turns on if an exception or error occurs on the REL operation. If columns 56 and 57 do not contain an indicator and an exception or error occurs, the program halts unless the INFSR subroutine is specified.

If the INFSR subroutine is specified, the subroutine automatically receives control when an exception or error occurs. For more information on the INFSR subroutine, see “Handling Exceptions and Errors” on page 191.

**Output Specifications**

You can release a device from a MRT or SRT program by coding R in column 16 of the output specifications. The device is released when the output specification occurs during the output operations. If a format name is coded in columns 46 through 53 of the next output specification, the display appears or the ICF operation runs, and then the device is released.

**Acquiring One or More Devices by the Program**

Both SRT programs and MRT programs can acquire one or more display stations or ICF sessions while the program is running. The program acquires other display stations or ICF sessions by using the OCL WORKSTN statement or the ACQ operation. For information about the WORKSTN statement, see the *System Reference for the System/36 Environment*.

A SRT program that acquires any display stations or ICF sessions must include a NUM option in the continuation line of the file description specifications for the WORKSTN file.

An acquired device cannot supply external indicators U1 through U8, and SUBR20 cannot be used to read or to write them.

An acquired device does not provide the display station local data area for the data structure defined by a U in column 18 of the input specifications, and SUBR21 cannot be used to read or to write a local data area for an acquired device.

Program error messages go to the requester. If the requester is an ICF session, program error messages go to the messages queue QSYSOPR.

In order for a program to acquire a display station, the display station must be powered on and have the SIGN ON display up.
REQUESTING THE PROGRAM

ACQ Operation

The ACQ operation acquires the device specified in factor 1 for the program. Factor 2 must contain the name of the WORKSTN file.

If the device is available, it is acquired by the program. If it is not available or was already acquired by the program, the indicator coded in columns 56 and 57 is set on.

If no indicator is coded in columns 56 and 57 but the program contains the INFSR (WORKSTN exception/error-processing) subroutine, the INFSR subroutine automatically receives control when an exception or error occurs on the ACQ operation.

If no indicator is specified and the program does not contain the INFSR subroutine, the program halts when an exception or error condition occurs. Then you can continue the job or try the ACQ operation again. No input or output operation occurs when you use the ACQ operation.

Requesting the Program by One or More Display Stations/Remote Systems

Each requester of a MRT program uses the same copy of the program. The first requester provides the external indicators (U1 through U8) and the display station local data area for the data structure defined by a U in column 18 of the input specifications. Other requesters can access any external indicators and display station local data area attached to the program by using SUBR20 and SUBR21.

Setting and Restoring External Indicators (SUBR20)

The IBM-written subroutine SUBR20 allows you to set and to restore the external indicators (U1 through U8) for each requesting display station when more than one display station requests the same program. To call SUBR20, code the EXIT SUBR20 operation, followed by exactly three RLABL operations (see the example below):

```
* 1+ 2+ 3+ 4+ 5+ 6+ 7+ 8+
CC1NO1N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++
C EXIT SUBR20
C RLABL OP 1
C RLABL TNAME 2
C RLABL RCODE 1
```

OP is a one-character field that contains I or O to indicate if the external indicators are to be input to or output from the program for this display station. To enter the appropriate character in the OP field, you can use a MOVE operation before calling the subroutine.

TNAME is a two-character field that contains the identification of the display station. This field is normally the field whose name is coded in columns 60 through 65 for the ID option on the WORKSTN file description continuation line. If TNAME is not the field whose name was coded in columns 60 through 65 on the WORKSTN file description continuation line, you can use a MOVE operation to enter the appropriate display station identification in the TNAME field before calling the subroutine.
REQUESTING THE PROGRAM

RCODE is a one-character field that contains one of the following return codes:

0 = Successful
1 = Unsuccessful (the display station is not attached to the program)
2 = Unsuccessful (the display station is not a requester).

The external indicators for the requester of a SRT program are automatically available to the program without the use of SUBR20 and are written out at end of job. The external indicators for the first requester of a MRT program are available without the use of SUBR20, but they are not automatically written out at end of job.

Reading and Writing the Local Data Area for a Display Station (SUBR21)

The IBM-written subroutine SUBR21 allows you to read and write the local data area for each display station when more than one display station requests the same WORKSTN file. To call SUBR21, code the EXIT SUBR21 operation followed by exactly four RLABL operations (see the example below):

```
C EXIT SUBR21
C RLABL OP 1
C RLABL TNAME 2
C RLABL RCODE 1
C RLABL AREA
```

OP is a one-character field that contains I or O to indicate if the display station local data area is input to or output from the program for this display station.

TNAME is a two-character field that contains the identification of the display station.

RCODE is a one-character field that contains one of the following return codes:

0 = Successful
1 = Unsuccessful (the display station is not attached to the program)
2 = Unsuccessful (the display station is not a requester).

AREA is a field or data structure from which the local data area for the display station is read or to which it is written. AREA can be up to 512 characters long. If AREA is between 257 and 512 in length, then AREA must be the name of a data structure. Position 1 of the local data area is always placed in position 1 of this field. If you use AREA to pass options to OCL statements, use caution when you use the characters question mark (?) and slash (/).

If a single display station is used, the program writes the information from the data structure to the local data area (LDA). To print, write, or display the contents of the LDA, use the contents as data passed to the desired output device. See Figure 81 on page 243 for an example of information transfer between programs.

The local data area for the requester of a SRT program or for the first requester of a MRT program can be referred to in your program if you define a data structure.
with a U in column 18 of the input specifications. For a MRT program, the local
data area is not automatically written out at the end of the job. (For more informa-
tion see the section “Local Data Area for a Display Station” on page 436 in
Chapter 15, “Using Data Structures.”)

Specifying and Compiling a MRT Program

Compiling the Program

When compiling a MRT program, make sure that the number you specify for the
MRTMAX parameter is equal to the maximum number of requesting devices for
that program. If you are using the CRTS36RPG or CRTS36RPT command dis-
plays on the AS/400 system, type the number in the entry field beside the prompt
Maximum MRT devices. If you use the RPGC or AUTOC procedure displays within
the System/36 Environment, the corresponding prompt is: Maximum number of
requesting display stations.

For the NEP parameter, you should specify that it is a never-ending program if it
will be used often or if initializing the program is time-consuming. The program is
initialized once and remains in main storage until the MRT job ends. Specify that it
is not a never-ending program if the program is seldom used, if initialization time is
negligible, or if DISP-SHR is not specified on the OCL FILE statement for the files
used by your program and needed often by other programs. NONEP programs
finish when their devices are no longer attached to them. If you are using the
CRTS36RPG or CRTS36RPT command displays on the AS/400 system, type *NO
or *YES in the entry field beside the prompt Never-ending program. If you use the
RPGC or AUTOC procedure displays within the System/36 Environment, respond
NONEP or NEP to the same prompt.

Specifying a MRT

Use the AS/400 CHGS36PRCA or EDTS36PRCA command to change the desig-
nation of a procedure to MRT.

Updating Disk Files in a MRT Program

Possible Errors

Use care when updating DISK files in a MRT program. If a file is shared by two or
more display stations in a program and if the record being processed is not
updated before the next record is read, the following errors can occur:

1. An update can be lost. For example, suppose a record is read from file X and
displayed at display station 1, then the same record is read from file X and
displayed at display station 2. The update that ran at one display station might
be destroyed by an update that ran at the other display station. If this condition
occurs and if DISP-SHR was specified on the OCL FILE statement for file X, an
error message is displayed and the second update is not run. If this condition
occurs and if DISP-SHR was not specified, the second update overlays the first
update.

2. An update run by another program sharing the file can be lost. For example,
suppose a record is read from file X and is displayed at display station 1, then
another record is read from file X and displayed at display station 2. The second read operation from file X causes OS/400 to free the first record. Another program sharing file X can then update the first record. If display station 1 then reads the record again and updates the original field values, the updates made by the other program may be lost.

3. The wrong record can be updated. For example, suppose a record is read from file X and displayed at display station 1, then a different record is read from file X and displayed at display station 2. If display station 1 tries to update the first record but the program does not reread that record, the program tries to update the last record read from file X. If this condition occurs during an attempt to update an indexed file, an error message is displayed and the requested update is not run. Otherwise, the wrong record is updated.

**Avoiding These Errors**

You can avoid the preceding error conditions by using one of the following techniques:

1. Before you update a record, read the record again and check that none of the fields being updated have been changed after the record was displayed for updating. If any of the fields were changed, you might want to display the field again for updating or, if possible, update the record by using the field values currently in the record.

2. Within the program, define an array for each DISK file. The array should contain one element for each display station. When a user enters a relative record number or the key field of an updated record, the program should check the array to ensure that no other display station is updating that record. If no other display station is updating the requested record, the program should place the specified relative record number or key field into the array element corresponding to the display station. The program can then read the record and display it at the display station. If another display station is updating the record, you can display a message saying so. After the user at the display station enters the updates, the program must read the record again and use the information entered at the display station to update the record. The program should then blank out the array element corresponding to the display station.

3. There is another way to solve the problem of more than one display station in a MRT program updating the same record. This method also applies if another program is updating the same file and causes updates for the MRT program to fail. That method is to define each file on a separate OCL FILE statement for each display station using the MRT program. Each of the FILE statements should specify a different name but the same label, and each FILE statement should specify DISP-SHR. This method locks or protects each record for each display station and each program when other display stations or programs try to access it.

The third technique outlined applies to the first two possible errors only.
DuPliCaTe ChArAcTeR VAleVe

Reaching End of File for a MRT Program

Primary File

If a MRT program is defined as a never-ending program, if all devices have been released or no input-capable records are in the WORKSTN file, and if the program tries to read another record from the WORKSTN file, end of file does not occur and RPG does not set on the LR indicator. However, if you set on the LR indicator, the program goes to end of job.

Note: A MRT program should not set on the LR indicator until end of file is reached for the WORKSTN file. If the LR indicator is set before end of file is reached, undesirable results occur for requesters that are signing on or that are still signed on.

Demand File

If the program is defined as a never-ending program and if all devices have been released or no input-capable records are in the WORKSTN file, the end-of-file indicator on the READ operation is not set on until another READ operation occurs. However, if you set on the LR indicator based on some condition other than the end-of-file indicator on the READ operation, the program goes to end of job.

Advanced Topics

Advanced topics include:

- Processing the duplicate character value
- Using message identification codes
- Overriding fields in a display format
- Using the POST operation.

Processing the Duplicate Character Value

If you specify Enable Dup for a field in a display format (by using SDA or a Y entry in column 34 of the display format D specifications), the user at the display station can press the Dup key to indicate to the program to duplicate the contents of the field from the field in the previous record. When the Dup key is pressed, the field, from the position of the cursor to the end of the field, is filled with the duplicate character value (hexadecimal 1C), which is displayed as the character . The Dup key does not duplicate any characters; therefore, you must process the duplicate character values in your program.

If you want the user at the display station either to duplicate the entire field or to type in the entire field, you need to test only one character in the field to determine if the Dup key was pressed. For example, you can test the last character in an alphanemic field for the duplicate character value by using the TESTB operation code. If the last character in the field is not a duplicate character value, move the contents of the test field to the processing field (see Figure 67 on page 209).

You can also write your program to allow the user to change the first part of a field and to duplicate the latter part of the field. For example, if the user changes the first four characters in a ten-character field and then presses the Dup key, positions 5 through 10 of the field will contain the duplicate character value (hexadecimal...
In your program, you then have to test each character in the field to determine where the first duplicate character occurs, and replace the appropriate positions with the data to be duplicated.

For a list of the hexadecimal value of each character, see Chapter 17, “Changing the Hexadecimal Value of Characters.”

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFIilenameSqNORiPos1NCCPos2NCCPos3NCC

INPUT
I........................................PFromTo++DField+L1M1FrPoNeEq....
I 1 6 TSTFLD
I 6 6 TEST

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1NO1N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C TEST'345' TEST 99 ALL ON?
C 99 TEST'01267' TEST 99 ALL OFF?
C N99 MOVE TSTFLD OLDFLD

Figure 67. Testing for a Duplicate Character Value

Using Message Identification Codes

When a message identification code (MIC) is displayed for a WORKSTN file, the length of the message must be entered in the field length column of the display format specifications, and the constant type column must contain an M. The name of a six-character field or a six-character constant must then be coded on the output specifications. The contents of the field or the constant must be in the form xxxxyy, where xxxx is the MIC number and yy is the two-character message member identifier. For a complete description of the message member identifier, see the manual SDA User’s Guide and Reference.

Overriding Fields in a Display Format

An override operation allows you to override fields in a display format when you redisplay the same format. You can specify an override operation when you design the format with SDA or with the display format S specification (by coding an indicator in columns 33 and 34). An override operation occurs if the indicator is on when the format is displayed. A normal output operation is run if the indicator is off when the format is displayed (see Figure 68 on page 210).

During an override operation (the indicator in columns 33 and 34 is on), the following occurs:

- A field is unchanged if you code an indicator in columns 23 and 24 of the D specification for that field and that indicator is off. If data was typed into the field, that data is unchanged. Any field that had Y, N, or blank coded in columns 23 and 24 is also unchanged.
- A field is displayed if you code an indicator in columns 23 and 24 of the D specification for that field and that indicator is on. Any data that was typed into
the field is lost. Output information is displayed from the same locations in the output record area as for a normal display.

- For all fields, the use of indicator-controlled characteristics such as highlight or reverse image is determined by the state of that indicator. All field characteristics that are not controlled by indicators are unchanged.

For example, you may want to override fields in a display if a user types incorrect data into a field. To override fields, code an indicator in columns 33 and 34 of the S specification to allow the format to be overridden. If the user types incorrect data into a field, you can then set on the indicator in columns 33 and 34 and display the format again. If the indicator coded in columns 23 and 24 of the D specification for the field is off, the incorrect data is unchanged. If the indicator is on, data from the program is displayed. You can also use indicators for field characteristics such as highlight and reverse image and can set these indicators on when the override indicator is set on.

For more information about overriding fields in a display format, see the manual SDA User's Guide and Reference.

<table>
<thead>
<tr>
<th>Indicator in Columns 33 and 34 of the S Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
</tr>
<tr>
<td>OFF</td>
</tr>
<tr>
<td>ON</td>
</tr>
</tbody>
</table>

*Figure 68. Effect of Indicators on Output Data during an Override Operation*

**Using the POST Operation on Display Stations**

The POST operation allows you to retrieve status information from the subfield named in columns 53 to 58 of the input specifications for the *SIZE, *MODE, *INP, and *OUT keywords in the INFDS data structure. After a POST operation is done, the *SIZE subfield will contain one of the following:

- 1920, to indicate that the specified display station has a 1920-character display screen (24 rows x 80 columns)
- 3564, if you are using a 3180 model 2 work station, to indicate that the specified display station has a 3564-character display screen capability (27 rows x 132 columns).

After a POST operation is done, the *MODE subfield will contain either 10, to indicate that DBCS support was requested, or 00, to show that it was not requested.
HOW WORKSTN FILES ARE PROCESSED

Following a POST operation, the *INP subfield will contain either 10, to indicate that
the DBCS keyboard is being used, or 00, to show that the alphabetic/Katakana
keyboard is being used.

Following a POST operation, the *OUT subfield will contain either 10, to indicate
that the display station’s screen is capable of displaying DBCS characters, or 00, to
show that the screen cannot display DBCS characters, or that the display is not
output capable.

Factor 1 must contain a variable or an alphabetic literal that identifies the display
station whose status is being requested. The result field must contain the name of
the INFDS data structure in which this information is posted.

Columns 56 and 57 can contain an indicator that turns on if an error occurs on the
POST operation. An error occurs if the specified display station is not attached to
the WORKSTN file for which the INFDS data structure is specified.

If columns 56 and 57 do not contain an indicator but the program contains the
INFSR subroutine, the subroutine automatically receives control when an error
occurs.

If columns 56 and 57 do not contain an indicator and the program does not contain
the INFSR subroutine, the program halts when an exception or error occurs.

How WORKSTN Files Are Processed

Figure 69 on page 212 shows how the RPG program cycle processes WORKSTN
input files. All steps in the cycle except steps 1, 3, 11, 12, 13, and 15 are the
same as those for the regular RPG program cycle.
1P output to a WORKSTN file is not allowed.

WORKSTN input processing can include:

- Saving the common IND/SAVDS area in the IND/SAVDS area for the device from which the last input record was read (if specified).
- Getting the display station record. If the display station is new to the file, the record may be blank. Only the last input-capable display format can be read into the program.
• Restoring the IND/SAVDS area of the device from which the last input record was read (if specified).

• Inserting a value into the ID field (if specified).

For a detailed explanation of processing WORKSTN input files, see Figure 70 on page 214 (an expansion of step 3).

Data keyed at a display station is returned (input) to the RPG program for processing when the operator presses a command key or the Enter key. The operator can also cause the data to return to the program by pressing the Field Exit, Field +, or Field - key if the last input field in the format is defined as an auto record advance field (column 36 of the D specification).

3 All function-key indicators are turned off; then the appropriate one, if any, is turned on.

Note: If an exception/error occurs on the READ, the function-key indicators are not reset.

4 If the READ operation code is used, it combines steps 3, 5, 11, 12, 13, and 14. If the EXCPT operation code is specified, it uses the ID field to direct output to the display station whose ID is contained in the field.
Step 3 of Figure 69 on page 212, which is the WORKSTN input processing step, is expanded and shown in Figure 70. The explanation of the steps shown in Figure 70 follows the diagram.

One RPG cycle is used to start each acquired device. If no input or output operation to the device has previously been specified, RPG acquires the device if necessary and creates a blank record to satisfy the first read.

All requesters of the program except the first enter the program at this step.
End of file occurs for a WORKSTN file at the time of the read if:

- All devices have been released.
- Input is not allowed from any of the attached display stations because:
  - A new format has not been displayed at the display station since data was last keyed.
  - Suppress input-yes is specified in column 35 of the S specification for the format currently displayed and no input-capable formats are concurrently displayed at the display station.
- The program is a NEP if all devices have been released, and if the job has been stopped with an ENDJOB command.

If this is the first input for this device, the indicators specified in the IND field are off and the SAVDS field is blank.

Steps 3-5 through 3-9 occur only for acquired devices.

**Step 3-1.** RPG determines if the IND and/or SAVDS continuation line option is coded on the file description specifications for this file. If neither option is coded, RPG goes to step 3-3.

**Step 3-2.** If the IND and/or SAVDS option is coded, RPG moves the common IND/SAVDS area to the IND/SAVDS area for the device from which the last input record was read.

**Step 3-3.** RPG determines if this is the first cycle for the first requester of the program. If it is, RPG goes to step 3-10. All requesters of the program except the first enter the program at step 3-10.

**Step 3-4.** If the device is not a requester, RPG determines if all devices in the internal device table have been started. If all are started, the program goes to step 3-10.

A device is started when it has been acquired and a successful input or output operation has occurred. If a device is acquired by the ACQ operation, the device is not considered started unless output is sent to the device in the same cycle.

**Step 3-5.** If not all devices have been started, RPG locates a device that has not been started.

**Step 3-6.** If the device located is a display station, RPG determines if it has been acquired.

**Step 3-7.** If the display station has not been acquired, RPG calls the OS/400 to acquire the display station.

**Step 3-8.** RPG determines if the acquire was successful. If it was not successful, RPG goes back to step 3-5.

**Step 3-9.** If the device is acquired, RPG creates a first-time blank record to satisfy the first read to the device. RPG then goes to step 3-11.
Step 3-10. RPG reads in the record from the device. Remember that all requesters of the program except the first enter the program at this point.

Step 3-11. RPG determines if WORKSTN input is available. If it is not, the program has reached the end of the file.

Step 3-12. RPG determines if the IND and/or SAVDS continuation line option is coded on the file description specifications for this file. If an option is not coded, RPG goes to step 3-14.

Step 3-13. If the IND and/or SAVDS option is coded, RPG moves the IND/SAVDS area for the device that satisfied the read to the common IND/SAVDS area.

Step 3-14. RPG inserts the device ID of the device that satisfied the read into the ID field if the ID continuation line option is specified on the file description specifications.

After the WORKSTN input file processing, RPG goes to step 4 as shown in Figure 69 on page 212.

User-Defined Communication Formats

The RPG compiler allows the use of user-defined ICF communication formats or system-supplied formats ("$FORMATS"), when doing input or output to a communications file through an RPG WORKSTN file.

You may already have formats that have been created on a System/36, using the Interactive Data Definition Utility (IDDU). You can convert your IDDU data dictionaries to ICF files in a library with the same name as the System/36 data dictionary, by using the IBM AS/400 system: System/36 Migration Aid, licensed program 5727-MG1, and the IBM AS/400 system: OS/400 Migration Aid, licensed program 5763-SS1. For more information about converting your data dictionaries to ICF files, see IBM AS/400: System/36 to AS/400 Migration Aid User's Guide and Reference, SC09-1166.

You can also create ICF files by using Data Description Specifications (DDS) and the CL command CRTICFF (Create Intersystem Communications Function File). You use DDS to define the formats, and then use CRTICFF to create the file associated with the formats. For more information about creating an ICF file, see ICF Programmer's Guide.

You must indicate at compile time that your program uses user-defined communication formats. Use one of the System/36 Environment procedures, RPGC or AUTOC, or one of the AS/400 system commands, CRTS36RPG or CRTS36RPT, to indicate this. Each of these procedures and commands has a prompt for specifying the library name that contains the ICF files.

In order to associate the ICF communication file with a WORKSTN file, you must use the CFILE continuation line option. CFILE must be coded in columns 54 through 59 of the file description specifications, and the name of the communications file must be specified in the leftmost of columns 60 through 67.
If your WORKSTN file includes both Screen Format Generator (SFGR) screen formats and user-defined communication formats, you should also specify the ID continuation option on the file description specifications. You can use this specification in your program to control the type of device used.

On output specifications, user-defined communication formats are specified in the same way as SFGR format names.

User-defined communication formats can have the same names as SFGR screen formats, even in the same program; however, it is best to use unique format names in the program.

Example of Using User-Defined Communication Formats

The following example shows how to associate a user-defined communication format file with a WORKSTN file, in order to use the confirm function of Advanced Program-to-Program Communications (APPC).

In this example, the DDS for the communication format looks like this:

```
*..1+...2+...3+...4+...5+...6+...7+...
AAN01N02N03T.Name++++RLen++TDpBLinPosFunctions++++++++++++++++++++
A INDARA
A R FORM1
A CONFIRM
A ALWMRT
A BUFFER 256
A R EVOKE
A EVOKE(RMTLIB/RMTPGM)
A SECURITY(2 'ABC123' 3 'CLARK')
A SYNLVL(*CONFIRM)
```
The DDS for the screen format looks like this:

```
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...
AAN01N02N03T.Name+++++RLen++TdpBLinPosFunctions+++++++++++++++++++++
A     INDARA
A     R SCRNI
A     FIELD A 19 10 3 21EDTCDE(1)
A     FIELD B 20 10 5 28EDTCDE(1)
A     FIELD C 20 30 7 28EDTCDE(1)
A     TEXT1 12 10 5
A     TEXT2 9 10 18
```

The following code associates the user-defined communication format file with a WORKSTN file:

```
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...
H...OLExeD..CDYI....S..........I...1.F.H...........T................
H     0641
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice++++.Exit+++.A....U+.*
FWSFILE CD 256 WORKSTN
F     KNUM 3
F     KFMTS SCREEN D and E
F     KID WSID F
F     KINFDS INFODS G
F     KCFILE COMM H and I
FPRTFILE O F 132 132 PRINTER
*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC................................
IWASFILE NS 11 79 CO
I     70 750QTY1
IINFODS DS
I     *STATUS STATUS
I     23 24 MAJCOD
I     25 26 MINCOD
```
Chapter 7. Using a WORKSTN File

219
USER-DEFINED COMMUNICATION FORMATS

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*  
0..................NO1NO2NO3Field+YBEnd+PConstant/editword++++++++++...* 

OWSFILE K E EVOK  
0 K5 'EVOKE' 
0 E COMM  
0 K5 'FORM1' 
0 FIELDA 19 
0 FIELDB 49 
0 FIELDC 79 
0 E DISPL  
0 K5 'SCRN1' 
0 FIELDA 19 
0 FIELDB 39 
0 FIELDC 59 
0 71 'THIS WILL BE' 
0 80 'DISPLAYED' 

OPRTFILE D 3 0419 LR  
0 45 'THIS PROG TRYOUT' 
0 60 'HAS' 
0 95 70 'FAILED' 
0 11N95 70 'PASSED' 
0 95 MAJCOD 75 
0 95 MINCOD 77

A CONFIRM specifies that the output operation can not end until a response is received from the remote program.  
B ALWWRT specifies that the remote program can begin sending after the current output operation is confirmed.  
C You must use the EVOKE keyword with SYNLVL(*CONFIRM) when you use the CONFIRM keyword.  
D FMTS indicates that screen formats are also included in the WORKSTN file.  
E SCREEN is the display file that holds all screen formats used for input and output with a display screen.  
F The ID continuation-line option must be specified when both user-defined communication and screen formats are included in a WORKSTN file.  
G The CFILE continuation-line option associates the WORKSTN file with a user-defined OS/400-ICF communication file.  
H COMM is the user-defined OS/400-ICF format file that holds all communication formats used during a communications session.  
I INFODS is that data structure that holds the return code information.  
J FIELDA, FIELD B, and FIELDC are numeric fields transmitted with the user-defined communication formats and displayed using the screen formats.  
K The 'WC' session ID makes RPG look for the user-defined communication format in the OS/400-ICF file COMM.  
L The '1S' session ID makes RPG look for the screen format in the display file SCREEN.
**AR230R (Inquiring into an Accounts Receivable File)**

Figure 73 on page 224 shows sample program AR230R, an accounts receivable inquiry program. The program displays the customer number, name, address, amount due, credit limit, amount due more than 30, 60, and 90 days, and the date of the last payment for the customer requested. The program reads the customer number (CUSNO) from the customer master file, CUSTMAST. The requested customer number is displayed. This record, however, cannot be updated. You can press F3 to end the program.

The explanations for the Indicator Definitions are shown below:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>Record ID for the blank read for the WORKSTN file</td>
</tr>
<tr>
<td>03</td>
<td>Record ID for format AR230RD1</td>
</tr>
<tr>
<td>04</td>
<td>Record ID for format AR230RD2</td>
</tr>
<tr>
<td>10</td>
<td>Record ID for CUSTMAST file</td>
</tr>
<tr>
<td>99</td>
<td>Error indicator (Turns on if the customer number does not exist in the CUSTMAST file)</td>
</tr>
<tr>
<td>KC</td>
<td>F3 ends the program.</td>
</tr>
</tbody>
</table>

**Running this program:** To run the program AR230R, code the following procedure:

```
// LOAD AR230R
// FILE NAME=CUSTMAST
// RUN
```

This program uses two displays, AR230RD1 and AR230RD2. The manual *SDA User’s Guide and Reference* explains how to create displays using SDA.
The first display, which prompts the user to enter a customer number, looks like this:

```
Customer Inquiry

Please enter customer number.
Customer number

Press the Enter key to see accounts receivable information.
```

Press F3 to return to the main menu.

*Figure 71. Display AR230RD1—Prompts the User to Enter a Customer Number*

If the display read into the program contains an A in the first position of the record, the CUSNO field from the CUSTNMBR record is used to chain to the CUSTMAST file. If the customer number entered is equal to a customer number in the CUSNO field in the CUSTMAST file, the second display is shown. In all other cases, the first display is shown again with the error message *Customer number does not exist. Enter a different customer number.*
The second display looks like this:

```
<table>
<thead>
<tr>
<th>Customer Inquiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer number</td>
</tr>
<tr>
<td>Customer name</td>
</tr>
<tr>
<td>Address</td>
</tr>
<tr>
<td>State and zip</td>
</tr>
</tbody>
</table>

Current amount due  ******* Credit limit  *******
Amount due over 30 days  *******
Amount due over 60 days  ******* Last amount paid  *******
Amount due over 90 days  ******* Last date paid  *******
```

Press the Enter key to continue.

Figure 72. Display AR230RD2–Shows Accounts Receivable Information

It shows the customer number, name, address, amount due, credit limit, amount due more than 30, 60, and 90 days, and the date of last payment. The user cannot update this information in this program.
AR230R (INQUIRING INTO AN ACCOUNTS RECEivable FILE)

*.. 1 ..+.. 2 ..+.. 3 ..+.. 4 ..+.. 5 ..+.. 6 ..+.. 7 ..*
H...OLExeD..CDYI....S...........I...1.F.H...........T................*
H

*.. 1 ..+.. 2 ..+.. 3 ..+.. 4 ..+.. 5 ..+.. 6 ..+.. 7 ..*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+.Exit++.A....U+.*
FCUSTNMBCP F 256 WORKSTN
FCUSTMASTIC F 256R 8AI 2 DISK
*.. 1 ..+.. 2 ..+.. 3 ..+.. 4 ..+.. 5 ..+.. 6 ..+.. 7 ..*
IFilenameSgNORiPos1NCCPos2NCCPos3NCC.......................
ICUSTNMBRNS  02  1 C
I   NS   03  1 CA
I..................................................PFromTo++DField+L1M1FrPoNeEq...*
I
I   NS   04  1 CB
ICUSTMASTNS  10
I
I   2   9 CUSNO
I
I   10  34 CUSNM
I
I   35  59 CUSA1
I
I   60  84 CUSA2
I
I   85 109 CUSA3
I
I   110 111 STATE
I
I   P 112 1160ZIPCD
I
I   P 131 1352AMDEU
I
I   P 124 1270CRLIM
I
I
I   P 155 15920VR30
I
I
I   P 160 16420VR60
I
I
I   P 140 1442LSTAP
I
I
I   P 165 16920VR90
I
I
I   P 136 1390DLTPM

Figure 73 (Part 1 of 2). Sample Program AR230R (Inquiring Into An Accounts Receivable File)
AR230R (INQUIRING INTO AN ACCOUNTS RECEIVABLE FILE)

Figure 73 (Part 2 of 2). Sample Program AR230R (Inquiring Into An Accounts Receivable File)
Figure 78 on page 231 shows sample program AR330R, which maintains a customer master file. This program can add new customer records, change existing customer records, and remove customer records. This is a MRT program that allows two requesting display stations. The first display looks like this:

![Maintain Customer Master File](image)

It prompts the user to enter a customer number and an A, C, or R to indicate to add, change, or remove a record from the customer master file CUSTMAST.

If the user enters an invalid number, one of several error messages is displayed.
If the customer number is valid, the program shows one of three displays that allows the user to add, change, or remove a record from the customer master file, depending on whether the user entered A, C, or R on the first display.

If the user types **A = Add a new customer record** for the *Choose an option* prompt in Figure 74 on page 226 and presses Enter, the display looks like Figure 75.

<table>
<thead>
<tr>
<th>Add a Customer Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date *******</td>
</tr>
<tr>
<td>Customer number . . . . . . . *******</td>
</tr>
<tr>
<td>Customer name . . . . . . . .</td>
</tr>
<tr>
<td>Customer address . . . . . . .</td>
</tr>
<tr>
<td>State and Zip code . . . . . . .</td>
</tr>
<tr>
<td>Phone number . . . . . . . . .</td>
</tr>
<tr>
<td>Customer type . . . . . . . . .</td>
</tr>
<tr>
<td>Credit limit . . . . . . . . .</td>
</tr>
<tr>
<td>Salesman number . . . . . . . .</td>
</tr>
</tbody>
</table>

F7 = End program  F3 = Do not add this customer record

*Figure 75. Display AR330RD2—Display to Add a Customer Record*
If the user types `C = Change a customer record` for the `Choose an option` prompt in Figure 74 on page 226 and presses Enter, the display looks like Figure 76.

![Display AR330RD3 - Display to Change a Customer Record](image)

*Figure 76. Display AR330RD3-Display to Change a Customer Record*
If the user types $D = \text{Delete a customer record}$ for the \textit{Choose an option} prompt in Figure 74 on page 226 and presses Enter, the display looks like Figure 77.

<table>
<thead>
<tr>
<th>Remove a Customer Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date *********</td>
</tr>
<tr>
<td>Customer number *********</td>
</tr>
<tr>
<td>Customer name ***********</td>
</tr>
<tr>
<td>Customer address *********</td>
</tr>
<tr>
<td>State and Zip code ***********</td>
</tr>
<tr>
<td>Phone number *** ***********</td>
</tr>
<tr>
<td>Customer type * ***********</td>
</tr>
<tr>
<td>Credit limit ***********</td>
</tr>
<tr>
<td>Salesman number ***********</td>
</tr>
<tr>
<td>Press enter to remove this customer record.</td>
</tr>
</tbody>
</table>

F7 = End program  F3 = Do not add this customer record

\textit{Figure 77. Display AR330RD4–Display to Remove a Customer Record}

You can press F7 from any display to end this program. You can return to the AR330RD1 display by pressing F3 from either the AR330RD2, AR330RD3, or AR330RD4 displays, and not add, change, or delete the customer record.

\textbf{Note:} All customer records are added to, changed in, or removed from the file CUSTMAST.

The following is an explanation of Indicator Definitions used in the program:

- \textbf{01} Record ID for the blank read for the WORKSTN file.
- \textbf{02} Record ID for format AR330RD1.
- \textbf{03} Record ID for format AR330RD2.
- \textbf{04} Record ID for format AR330RD3.
- \textbf{05} Record ID for format AR330RD4.
- \textbf{08} First-time switch for code needed only for the first cycle.
- \textbf{10} Turns on if option A is chosen on format AR330RD1 and is used to display format AR330RD2.
- \textbf{20} Turns on if option C is chosen on format AR330RD1 and is used to display format AR330RD3.
- \textbf{30} Turns on if option R is chosen on format AR330RD1 and is used to display format AR330RD4.
- \textbf{50} Turns on if the customer record is not found in CUSTMAST.
- \textbf{60} Record ID for records read from CUSTMAST.
- \textbf{90} Error (The CUSNO field read from format AR330RD1 is blank.)
AR330R (MAINTAINING A CUSTOMER MASTER FILE)

91 Error (The option chosen on AR330RD1 was not A, C, or R.)
92 Error (The add option was chosen, but customer number is already in the CUSTMAST file.)
93 Error (The change option was chosen, but customer number is not in the CUSTMAST file.)
94 Error (The remove option was chosen, but customer number is not in the CUSTMAST file.)
95 Error (When the add option was chosen, the customer number was not in the CUSTMAST file. But after format AR330RD2 was displayed, another person added the customer record. The record is not added now.)
96 Error (When the change option was chosen, the customer number was in the CUSTMAST file. But after format AR330RD3 was displayed, another person deleted the customer record. The record is not changed now.)
97 Error (When the remove option was chosen, the customer number was in the CUSTMAST file. But after format AR330RD4 was displayed, another person deleted the customer record. The record is not removed now.)
99 Error indicator for formats (Turns on any time one of the error indicators 90-97 turns on.)

KD Press F4 to display format AR330RD1 again after format AR330RD2, AR330RD3, or AR330RD4 is displayed.

KG Press F7 to end the program.

Running This Program: To run AR330R, code the following procedure:

// LOAD AR330R
// FILE NAME=CUSTMAST
// RUN
Figure 78 (Part 1 of 10). Sample Program AR330R (Maintaining a Customer Master File)
Figure 78 (Part 2 of 10). Sample Program AR330R (Maintaining a Customer Master File)
*.. 1...+... 2...+... 3...+... 4...+... 5...+... 6...+... 7 ..*
IFIlenameSsqNORiPos1NCCPos2NCCPos3NCC
I* 1*
I* The save data structure is used to save the customer number
I* for each display.
I*
I* ISAVERDS        DS
I*..............................PFromTo+++DField+L1M1FrPoNeEq...*
I*  1    8  CUSNO
I* 1*
I* This data structure is used to initialize all the fields not used
I* by this program when a record is added to file CUSTMAST.
I*
I* DS
I*  11   16'zerodot'DLTPM
I*  11   16'zerodot'DLTOR
I*  17   230RRFS
I*  17   230RLST
I*  24   79  BUFFR

Figure 78 (Part 3 of 10). Sample Program AR330R (Maintaining a Customer Master File)
**AR330R (MAINTAINING A CUSTOMER MASTER FILE)**

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcedFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* Initialize the fields used for adding a new customer record.
C*
C  N08 Z-ADD*ZERO AMDUE
C  N08 Z-ADD*ZERO DLTPM
C  N08 SETON 08
C*
C* These indicators turn on during each cycle as they are needed.
C*
C  SETOF 102030
C  SETOF 909192
C  SETOF 939495
C  SETOF 969799
C*
C* If function key 3 is pressed, skip calculations and
display format AR330RD1. If function key 7 is pressed,
skip calculations and release this display station.
C*
C  KC
COR KG GOTO ENDCAL
C*
C* If format AR330RD1 was read, indicator 02 is on. If the CUSNO
field is blank, display a message to ask for a nonblank customer
number.
C*
C 02 CUSNO COMP *BLANK 90
C 02 90 SETON 99
```

*Figure 78 (Part 4 of 10). Sample Program AR330R (Maintaining a Customer Master File)*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C* If format AR330RD1 was read, indicator 02 is on. If the option
C* chosen is not A, C, or R, indicators 90 and 99 turn on to display
C* an error message.
C*
C 02N99 OPTION COMP 'A' 10
C 02N99 OPTION COMP 'C' 20
C 02N99 OPTION COMP 'R' 30
C 02N99N10
CANN20N30 SETON 9199
C*
C* If a valid option was chosen on format AR330RD1 and the CUSNO
C* field is not blank, check for the following errors:
C* Option A - Display error message if customer number is found.
C* Option C - Display error message if customer number is
C* not found.
C* Option R - Display error message if customer number is
C* not found.
C*
C 02N99 CUSNO CHAINCUSTMAST 50
C 02 10N50 SETON 9299
C 02 20 50 SETON 9399
C 02 30 50 SETON 9499
C*
C* If format AR330RD2 is read, add the customer record to the file.
C* If the customer record is found when the add is attempted, display
C* an error message.
C*
C 03 CUSNO CHAINCUSTMAST 50
C 03 50 EXCPTADD
C 03 50 SETON 01
C 03N50 SETON 9599

Figure 78 (Part 5 of 10). Sample Program AR330R (Maintaining a Customer Master File)
*.. 1  ... 2  ... 3  ... 4  ... 5  ... 6  ... 7  ..*
CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments+++++++
C*
C* If format AR330RD3 is read, change the customer record. If the
C* customer's record is not found when the change is attempted,
C* display an error message.
C*
C  04  CUSNO  CHAINCUSTMAST  50
C  04N50  EXCPUPDATE
C  04N50  SETON    01
C  04  50  SETON    9699
C*
C* If format AR330RD4 is read, remove the customer record. If the
C* customer's record is not found when the delete is attempted,
C* display an error message.
C*
C  05  CUSNO  CHAINCUSTMAST  50
C  05N50  EXCPDELETE
C  05N50  SETON    01
C  05  50  SETON    9799
C  ENDCAL  TAG

Figure 78 (Part 6 of 10). Sample Program AR330R (Maintaining a Customer Master File)
AR330R (MAINTAINING A CUSTOMER MASTER FILE)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn........................................*

OWORKSTN D    01
0       OR    99
0       OR    KC
0..........................N01N02N03Field+YBEnd+PConstant/editword+...........*

0                     K8 'AR330RD1'
0
0                     UPDATE Y    8
0                     99 CUSNO    16
0                     99 OPTION    17
0
0                      99N91  36 'THE CUSTOMER NUMBER'
0                     90       60 ' CANNOT BE BLANK. ENTER'
0                     90       79 ' A CUSTOMER NUMBER.'
0                     91       41 'CHOOSE OPTION A, C, OR R'
0                     91       42 '

0                     92       60 ' YOU ARE TRYING TO ADD I'
0                     92       82 'S ALREADY IN THE FILE.'
0                     93       60 ' YOU ARE TRYING TO CHANG'
0                     93       81 'E IS NOT IN THE FILE.'
0                     94       60 ' YOU ARE TRYING TO REMOV'
0                     94       81 'E IS NOT IN THE FILE.'
0                     95       60 ' WAS ADDED BY SOMEONE EL'
0                     95       84 'SE AFTER YOU CHOSE THE A'
0                     95       94 'DD OPTION.'
0                     96       60 ' WAS REMOVED BY SOMEONE '
0                     96       84 'AFTER YOU CHOSE THE CHAN'
0                     96       94 'GE OPTION.'
0                     97       60 ' WAS REMOVED BY SOMEONE '
0                     97       84 'AFTER YOU CHOSE THE REMO'
0                     97       94 'VE OPTION.'

Figure 78 (Part 7 of 10). Sample Program AR330R (Maintaining a Customer Master File)
Figure 78 (Part 8 of 10). Sample Program AR330R (Maintaining a Customer Master File)
Release the display station at which function key 7 was pressed.

Change the customer record.

```
OCUSTMASTE     UPDATE
  0            CUSNMW  34
  0            CUSA1W  59
  0            CUSA2W  84
  0            CUSA3W  109
  0            STATEW  111
  0            ZIPCDW  116P
  0            AREACW  118P
  0            PHONEW  122P
  0            CUSTPW  123
  0            CRLIMW  127P
  0            SLSNOW  130P
  0            UPDATE  192P

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...* OName++++DFBASbSaN01N02N03Excptn....................................* 

Add the customer record. All fields in this record are written to initialize them properly.

```

```
EADD ADD
0..................N01N02N03Field+YBEnd+PConstant/editword+++++++++++...*
  0            ARCOD    1
  0            CUSNO    9
  0            CUSNMW   34
  0            CUSA1W   59
  0            CUSA2W   84
  0            CUSA3W   109
  0            STATEW   111
  0            ZIPCDW   116P
  0            AREACW   118P
```

Figure 78 (Part 9 of 10). Sample Program AR330R (Maintaining a Customer Master File)
Figure 78 (Part 10 of 10). Sample Program AR330R (Maintaining a Customer Master File)

0 PHONEW 122P
0 CUSTPW 123
0 CRLIMW 127P
0 SLSNOW 130P
0 AMDUE 135P
0 DLTPM 139P
0 LSTAP 144P
0 PRBAL 149P
0 CHGTD 154P
0 OVR30 159P
0 OVR60 164P
0 OVR90 169P
0 CRDTD 174P
0 ADJTD 179P
0 SLSLY 184P
0 DTLOR 188P
0 UDATE 192P
0 RRFST 196P
0 RRLST 200P
0 BUFFR 256

0* Delete the customer record.
0* EDEL DELETE

Figure 78 (Part 10 of 10). Sample Program AR330R (Maintaining a Customer Master File)
AR935R (Requesting a Printout of Accounts Receivable)

Figure 81 shows sample program AR935R, which allows the user to choose the type of aged trial balance report printed by sample program AR936R, shown in Chapter 8, “Using a PRINTER File.”

The first display looks like this:

```
Print aged trial balance report.
Choose report option . . . 1,2,3,4 *
  1 = All customers.
  2 = Customers with balances.
  3 = Customers with balances over a certain amount.
  4 = Customers with overdue balances.

F7 = End program
```

Figure 79. Display AR935RD1—Display to Select a Printed Report

You can choose the type of report you want by selecting one of the options below:

1 = All customers.
2 = Customers with balances.
3 = Customers with balances over a certain amount.
4 = Customers with overdue balances.
If the user enters 3 on the first display, the second display is shown. The second display looks like this:

```
Print report for balances over what dollar amount? ................................... $ .00
```

F7 = End program  F3 = Choose a different report option

*Figure 80. Display AR935RD2—Display to Select a Minimum Balance*

The information that the user enters is written to the display station local data area (the data structure coded on the input specifications). The information in this local data area is read by program AR936R, which prints the report requested.

The following information is written in the local data area:

<table>
<thead>
<tr>
<th>Position</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Report option</td>
</tr>
<tr>
<td>2-8</td>
<td>OVBAL if option 3 is chosen</td>
</tr>
<tr>
<td>9</td>
<td>C if F7 is pressed. This tells the procedure to run AR936R.</td>
</tr>
</tbody>
</table>

The following is an explanation of the Indicator Definitions used in the program:

- **01** Record ID for format AR935D1.
- **02** Record ID for format AR935D2.
- **03** Checks for a valid option.
- **04** Checks for a valid option.
- **05** Conditions first-time processing.
- **30** Turns on if option 3 is chosen and displays format AR935RD2.
- **99** Turns on if the option chosen was not 1, 2, 3, or 4 and displays an error message.
- **KC** Press F3 to display format AR935RD1 again when format AR935RD2 has been displayed.
- **KG** Press F7 to cancel the report request and end the program.
LR

Turns on when the last record is processed. The program goes to end of job.

**Running This Program:** AR935R must be run before AR936R. To run both AR935R and AR936R, code the following procedure:

```plaintext
// LOAD AR935R
// WORKSTN UNIT-?WS?,RESTORE-YES
// RUN
* If F7 is presses in AR935R, a nonblank character
* is placed in position 9 of the local data
* area and AR936R is not run.
// IFF ?L'9,1'?/ GOTO NOPRT
// LOAD AR936R
// FILE NAME-CUSTMAST
// RUN
// TAG NOPRT
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
H...OExeD..CDYI....S.........I...1.F.H.........T...................
H 14
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRienLK1AI0vKlocEDevice+.......Exit++......A....U+.*
FWORKSTN CP  256 WORKSTN
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.............................*
IWORKSTN NS  01  1 CA
I..................................................PFromTo+DField+L1M1FrPoNeEq...*
I 2  2 OPTION
I NS  02  1 CB
I 2  800VBAL
I NS  05
I*  
I* This is the local data area information passed from AR935R to AR936R.
I*  
I UDS
I 1  1 OPTION
I 2  800VBAL
I 9  9 F7
I 1  9 CLEAR
```

*Figure 81 (Part 1 of 3). Sample Program AR935R (Requesting a Printout of Accounts Receivable)*
Figure 81 (Part 2 of 3). Sample Program AR935R (Requesting a Printout of Accounts Receivable)
OE140R (Entering Orders from Customers)

Figure 84 on page 248 shows sample program OE140R, which allows a user to enter customer orders at a display station. The user enters a customer number and an order number on the first display:

Order Entry

Enter the customer number. . . .
Enter the order number . . . .

F7 = End program

Figure 82. Display OE140RD1—Order Entry Display
On the second display, the user enters the item number and the quantity ordered. When this information is entered, the program displays the item number, the quantity ordered, a description of the item, the unit cost, and the total cost for the quantity of that item ordered. Up to four items are displayed on the same display:

<table>
<thead>
<tr>
<th>Customer Number</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>---------------</td>
</tr>
<tr>
<td>Address</td>
<td>---------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item No.</th>
<th>Qty</th>
<th>Description</th>
<th>Price</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enter Item Number. . . ********
Enter Quantity . . . ********

F2 = End of Order  Roll = Roll through line items

Figure 83. Display OE140RD2–Order Entry Information Display

The *STATUS keyword for the file information data structure INFDS and the exception/error-processing subroutine INFSR allow the user to use the Roll Up and Roll Down keys to display additional orders and make corrections. Press F2 to write the order to the CUSTORDS file. If the array entry is blank (no items ordered), no order is written to the CUSTORDS file. Press F7 on the display OE140R to end the program.

The following records are written to the CUSTORDS file.

- Customer record
- Ship-to record
- Line-item record (one for each line item entered).

The following is an explanation of the Indicator Definitions:

- 01 Record ID for the blank read for the WORKSTN file.
- 02 Record ID for format OE140RD1.
- 03 Record ID for format OE140RD2.
- 04 Indicator is turned on by the information subroutine, passes control to the beginning of detail calculations, and skips to the end of detail calculations.
- 11 Indicator turns off when no line items have been entered and is used to display array line one.
Indicator turns off when only one item has been entered and is used to display array line two.

Indicator turns off when only two items have been entered and is used to display array line three.

Indicator turns off when only three items have been entered and is used to display array line four.

Indicator is used in subroutines UPINDEX and DNINDEX to determine when 1 is to be added to or subtracted from the array indexes A, B, C, and D. The indexes are not changed if I is less than 4.

Indicator is used in subroutines ROLLUP and ROLLDOWN to indicate that a roll cannot be done because the top or bottom of an array has been reached.

Indicator turns on when the Roll Up key is pressed.

Indicator turns on when the Roll Down key is pressed.

Indicator is used in subroutine ADDORD to determine when all line items are written to the file CUSTORDS.

Indicator is used in subroutine ADDORD to write the customer record and the ship-to record to the file CUSTORDS for each order.

Error (The customer number entered on format OE140RD1 was blank. It cannot be blank.)

Error (The order number entered on format OE140RD1 was blank. It cannot be blank.)

Error (The customer number entered on format OE140RD1 was not found in the file CUSTMAST.)

Error (The item number entered on format OE140RD2 was blank. It cannot be blank.)

Error (The quantity entered on format OE140RD2 was zero. It cannot be zero.)

Error (The item number entered on format OE140RD2 was not found in the file ITEMMSTR.)

Error (The operator tried to enter more than 98 items for one order on format OE140RD2.)

Error indicator for formats (Turns on any time one of the error indicators 90-96 turns on.)

F2 writes the order placed in file CUSTORDS.

F7 ends the program.

Roll Up is used to roll forward through the line items.

Roll Down is used to roll backwards through the line items.
Running This Program: Before the user runs this program, the file CUSTORDS must be created. The user must also code the following procedure:

```
// LOAD OE140R
// FILE NAME-CUSTMAST
// FILE NAME-ITEMMSTR
// FILE NAME-CUSTORDS
// RUN
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
H 064
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+........Exit++.......A.....U+.*
FCUSTMASTIC F 256 256R 8AI 2 DISK
FITEMMSTRIC F 128 128R 8AI 2 DISK
FCUSTORDSO F 128 128 DISK A
FWORKSTN CP 500 WORKSTN
F KINFDS DSINF
F KINFSR SUBINF
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++ArrnamEntParrLenPDSAltnamLenPDSComments+++++++++
E ANBR 99 8 ITEM-NUMBER ARRAY
E AQTY 99 6 0 QUANTITY ORDERED
E*ARRAY
E ADSC 99 30 ITEM DESCRIPTION
E*ARRAY
E APRI 99 9 2 ITEM-PRICE ARRAY
E AEXA 99 9 2 TOTAL-PRICE ARRAY
```

Figure 84 (Part 1 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

Figure 84 (Part 2 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcodeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C* Initialize error indicators to off.
C*
C  SETOF                929596
C  SETOF                99
C*
C* If function key 2 was pressed on format OE14ORD2,
C* add the items to file CUSTORDS.
C*
C  KB  EXSR  ADDORD
C* If function key 7 was pressed, go to the end of
C* calculations and turn on LR. Indicator 04 is turned on by the
C* information subroutine to indicate that a roll key had been pressed
C* and that detail output should be performed.
C* If function key 2 had been pressed, skip over detail
C* calculations.
C*
C  04
COR  KB
COR  KG            GOTO ENDCAL
C* If format OE14ORD1 was read, indicator 02 is on. If field CUSNO or
C* ORDNO is blank or zero, display an error message. Note: Only one
C* error message can be displayed at a time.
C*
C  02  90
COR  02  91    SETON                99
C  02  90    SETOF                91

Figure 84 (Part 3 of 11). Sample Program OE140R (Entering Orders from Customers)
C* If CUSNO and ORDNO were not blank or zero, check to see if the
customer number is in the CUSTMAST file. If it is not found, display
C* an error message.
C*
C  02N99  CHAINCUSTMAST  92
C  02  92  SETON  99
C*
C* If the customer number was valid, initialize the array index fields
C* to display format OE140RD2.
C*
C  02N99  Z-ADD1  A  20
C  02N99  Z-ADD2  B  20
C  02N99  Z-ADD3  C  20
C  02N99  Z-ADD4  D  20
C  02N99  Z-ADD0  I  20
C  02N99  Z-ADD1  J  20
C  02N99  SETOF  111213
C  02N99  SETOF  14

.. 1  ...+... 2  ...+... 3  ...+... 4  ...+... 5  ...+... 6  ...+... 7  ...
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++  
C* If format OE140RD2 was read, indicator 03 is on. If field ITNBR
C* or QTYOR is blank or zero, display an error message. Note: Only
C* one error message can be displayed at a time.
C*
C  03  93
COR  03  94  SETON  99
C  03  93  SETOF  94
C*
C* Check to make sure the maximum number of lines has not been entered.
C* If 98 orders have been entered, the operator can enter no more. An
C* error message is displayed.
C*
C  03N99  I  COMP  98  96
C  03  96  SETON  99

Figure 84 (Part 4 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

C*
C* If ITNBR and QTYOR were not blank or zero, check to see if the
C* item number is in the ITEMMSTR file. If it is not found, display
C* an error message.
C*
C 03N99 ITNBR CHAINITEMMSTR 95
C 03 95 SETON 99
C*
C* If the item number was valid, move the item number, the quantity
C* ordered, the description, and the price (obtained from file ITEMMSTR),
C* and the total cost for the item (product of the quantity and
C* the price) into arrays for display and for later output to CUSTORDS.
C*
C 03N99 ADD 1 I
C 03N99 ADD 1 J
C 03N99 MOVE ITNBR ANBR,I
C 03N99 Z-ADDQTYOR AQTY,I
C 03N99 MOVE ITDSC ADSC,I
C 03N99 Z-ADDPRICE APRI,I
C 03N99 PRICE MULT QTYOR AEXA,I
C 03N99 MOVE ANBR,J ITNBR
C 03N99 Z-ADDAQTY,J QTYOR

Figure 84 (Part 5 of 11). Sample Program OE140R (Entering Orders from Customers)
C* End of detail calculations.
C*
C  ENDCAL   TAG
C    KG     SETON     LR
C    SETOF   04
C*
C* This is the information subroutine. Control comes directly to here
C* when one of the roll keys is pressed. It allows the operator to press
C* the roll keys and roll through the items already entered.
C*
CSR  SUBINF  BEGSR
CSR  STATUS  COMP 01122     60
CSR  STATUS  COMP 01123     61
CSR  60     EXSR ROLLUP
CSR  61     EXSR ROLLDN
CSR  SETON  0304
CSR  ENDSR'"DETC'"
C* When the Roll Up function key is pressed, this subroutine moves the
C* content of the arrays (which is two higher than what is displayed)
C* into the fields ITNBR and QTYOR so the operator can change that item
C* or get back to the top of the list.
C*
CSR  ROLLUP  BEGSR
CSR    I       COMP 98     25
CSRN25  ADD 1       I
CSRN25  ADD 1       J
CSRN25  MOVE ANBR,J ITNBR
CSRN25  Z-ADDAQTY,J QTYOR
CSRN25  EXSR UPINDEX
CSRN25  EXSR SETIND
CSR  ENDSR

Figure 84 (Part 6 of 11). Sample Program OE140R (Entering Orders from Customers)

Chapter 7. Using a WORKSTN File  253
C*
C* When the Roll Down function key is pressed, this subroutine moves
C* the bottom array element displayed to fields ITNBR and QTYOR so the
C* operator can change those fields.
C*
```
CSR ROLLDN BEGSR
CSR I COMP *ZERO 25
CSRN25 MOVE ANBR,I ITNBR
CSRN25 Z-ADDAQTY,I QTYOR
CSRN25 SUB 1 I
CSRN25 SUB 1 J
CSRN25 EXSR DNINDX
CSRN25 EXSR SETIND
CSR ENDSR
```

C* This subroutine adds 1 to the indexes used to display the array data
C* on the screen if more than four line items have been entered.
C*
```
CSR UPINDX BEGSR
CSR I COMP 4 20
CSR 20 ADD 1 A
CSR 20 ADD 1 B
CSR 20 ADD 1 C
CSR 20 ADD 1 D
CSR ENDSR
```

Figure 84 (Part 7 of 11). Sample Program OE140R (Entering Orders from Customers)
C*
C* This subroutine subtracts 1 from the indexes used to display the array
C* data on the screen when rolling down and when the item moved into
C* ITNBR is greater than the fourth element in the arrays.
C*

```
CSR DNINDEX BEGSR
CSR I COMP 4  20  20
CSR 20 SUB 1 A
CSR 20 SUB 1 B
CSR 20 SUB 1 C
CSR 20 SUB 1 D
CSR ENDSR
```

C*
C* This subroutine turns on indicator 11 if one or more line items have
C* been entered, indicator 12 if two or more line items have been
C* entered indicator 13 if three or more line items have been entered,
C* indicator 14 if four or more line items have been entered.
C*

```
CSR SETINDEX BEGSR
CSR I COMP 4  14  14
CSR I COMP 3  13  13
CSR I COMP 2  12  12
CSR I COMP 1  11  11
CSR ENDSR
```

Figure 84 (Part 8 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC11N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++++
C*
C* This subroutine puts the order entered in file CUSTORDS. It writes
C* the customer order record, then the ship-to record. Next it writes
C* one line-item record for each line item entered. If no line items
C* have been entered, no records are written.
C*
CSR ADDORD BEGSR
CSR SETOF 7071
CSR Z-ADD1 I
CSR LOOP TAG
CSR ANBR,I COMP *BLANK 70
CSR 70 GOTO END
CSRN71 EXCPTCUSHP
CSRN71 SETON 71
CSR EXCPTLINEIT
CSR ADD 1 I
CSR GOTO LOOP
CSR END TAG
CSR MOVE *BLANK ANBR
CSR MOVE *ZERO AQTY
CSR MOVE *BLANK ADSC
CSR MOVE *ZERO APRI
CSR MOVE *ZERO AEXA
CSR ENDSR

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn....................................
WORKSTN D 01
0 OR 02 99
0 OR KB
0 OR KH

Figure 84 (Part 9 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

0..............N01N02N03Field+YBEnd+PConstant/editword+++++++++++.*
0                      KB 'OE14ORD1'
0                      99  CUSNO     8
0                      99  ORDNO    14
0                      99N91  33 'THE CUSTOMER NUMBER'
0                      91   33 '   THE ORDER NUMBER'
0                      99N92  50 'CANNOT BE BLANK.'
0                      92   48 'WAS NOT FOUND.'

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn....................................................
0                      D 02N99NKG
0                      OR 03NKBHK
0..............N01N02N03Field+YBEnd+PConstant/editword+++++++++++.*
0                      KB 'OE14ORD2'
0                      CUSNO     8
0                      ORDNO    14
0                      NAMAD    116
0                      ZIPCD    125
0                      11 ANBR,A 133
0                      11 AQTY,A 139
0                      11 ADSC,A 169
0                      11 APRI,A 180 '$'
0                      11 AEXA,A 191 '$'
0                      12 ANBR,B 199
0                      12 AQTY,B 205
0                      12 ADSC,B 235
0                      12 APRI,B 246 '$'
0                      12 AEXA,B 257 '$'
0                      13 ANBR,C 265
0                      13 AQTY,C 271
0                      13 ADSC,C 301
0                      13 APRI,C 312 '$'
0                      13 AEXA,C 323 '$'

Figure 84 (Part 10 of 11). Sample Program OE140R (Entering Orders from Customers)
OE140R (ENTERING ORDERS FROM CUSTOMERS)

0 14 ANBR,D 331
0 14 AQTY,D 337
0 14 ADSC,D 367
0 14 APRI,D 378 '$'
0 14 AEXA,D 389 '$'
0 ITNBR 397
0 QTYOR 403
0 93 418 'THE ITEM NUMBER'
0 93 435 'CANNOT BE BLANK.'
0 94 427 'THE QUANTITY CANNOT BE Z'
0 94 431 'ERO.'
0 95 418 'THE ITEM NUMBER'
0 95 433 ' WAS NOT FOUND.'
0 96 427 'TOO MANY LINE ITEMS HAVE'
0 96 441 ' BEEN ENTERED.'

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excpn..............................................*
OCUSTORDSEADD CUSSHP
0..................N01N02N03Field+YBEnd+PConstant/editword++++++++++++...*
0 3 ' CU'
0 CUSNO 11
0 3 ' CS'
0 CUSNO 11
0 LINEIT
0 3 ' IT'
0 CUSNO 11
0 I 19
0 ANBR,I 27
0 AQTY,I 33
0 AEXA,I 42

Figure 84 (Part 11 of 11). Sample Program OE140R (Entering Orders from Customers)
Chapter 8. Using a PRINTER File

A PRINTER file provides output to a printer. An RPG II program can use a maximum of eight PRINTER files; each PRINTER file must have a separate name. Use the OCL PRINTER statement to assign a PRINTER file to a particular printer. For information about the PRINTER statement, see the System Reference for the System/36 Environment.

To use a PRINTER file, code entries on the file description, line counter, and output specifications.

File Description Specifications

Code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>Filename</th>
<th>File Type</th>
<th>Mode of Processing</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>External Record Name</th>
<th>Record Length</th>
<th>Block Length</th>
<th>Overflow Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>P</td>
<td>F</td>
<td></td>
<td></td>
<td>PRINTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the PRINTER file.

Column 15 must contain O to indicate that the file is an output file.

Column 19 must contain F or blank to indicate that all the records in the file have the same length.

Columns 20 through 23 must contain the block length or blanks. If you enter the block length, it must equal the record length coded in columns 24 through 27. If you leave the entry blank, the program assumes that the block length equals the record length.

Columns 24 through 27 must contain the length of the largest record in the file. The record length can be 1 through 132 or 1 through 225, depending on the number of print positions your printer has.

Columns 33 and 34 can contain an overflow indicator (OA through OG or OV) or blank. For more information about overflow indicators, see “Overflow Indicators” on page 265.

Column 39 must contain L if the file is further described on the line counter specifications.

Columns 40 through 46 must contain the device name PRINTER.
Columns 71 and 72 can contain U1 through U8 to indicate that the file is conditioned by an external indicator. For more information about external indicators, see Chapter 13, “Using Indicators.”

**Line Counter Specifications**

Code entries in the unshaded columns on the line counter specifications as shown below:

<table>
<thead>
<tr>
<th>L</th>
<th>FL</th>
<th>OL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FL</td>
<td>OL</td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the PRINTER file. This name must be the same as the name coded in columns 7 through 14 of the file description specifications.

Columns 15 through 17 must contain the number of lines per page. The number of lines can be 1 through 112.

Columns 18 and 19 must contain FL to indicate that the entry in columns 15 through 17 is the form length.

Columns 20 through 22 must contain the line number that is the overflow line. This entry can be 2 through 112, but it must be less than or equal to the number coded in columns 15 through 17. If the entry in columns 20 through 22 equals the entry in columns 15 through 17, overflow does not occur. If you leave these columns blank, the overflow line is six lines from the bottom of the page (line 60 if you have 66 lines per page). You cannot override the entry in columns 20 through 22 by using the LINES option of the OCL PRINTER statement.

For more information about overflow, see “Handling Overflow” on page 265.

Columns 23 and 24 must contain OL to indicate that the entry in columns 20 through 22 is the overflow line.
Output Specifications

File- and record-identification entries in columns 7 through 37 describe the output file, the records in the file, the values that control the spacing and skipping by the printer, and the indicators that condition the output.

Field-description entries in columns 23 through 70 describe the position and format of data on the output record. These entries must begin one line below the file- and record-identification entries.

File- and Record-Identification Entries

Columns 7 through 14 must contain the name of the PRINTER file. This name must be the same as the name given to the file in columns 7 through 14 of the file description specifications.

Columns 14 through 16 can contain AND if more than three indicators are needed to condition an output operation. Columns 14 and 15 can contain OR if an output operation is conditioned by any one of two or more output indicators or sets of output indicators. For more information about AND and OR lines, see "AND and OR Lines" on page 263.

Column 15 must contain H (heading), D (detail), T (total), or E (exception) to indicate the type of record written.

Column 16 must contain F if you use the fetch overflow routine. For more information, see "Fetch Overflow Routine" on page 269.

Column 17 can contain 0 through 3 to indicate the number of lines to be spaced before a line is printed. Spacing means advancing the form in the printer a specified number of lines. For more information about spacing, see "Spacing and Skipping" on page 269.

Column 18 can contain 0 through 3 to indicate the number of lines to be spaced after a line is printed.

Columns 19 and 20 can contain 01 through 99, A0 through A9, or B0 through B2 to indicate the line number that the printer should skip to before printing. A0 through A9 means 100 through 109. B0 through B2 means 110 through 112. Skipping
means advancing the page in the printer to a specified line. For more information about skipping, see “Spacing and Skipping” on page 269.

Columns 21 and 22 can contain 01 through 99, A0 through A9, or B0 through B2 to indicate the line number that the printer should skip to after printing. A0 through A9 means 100 through 109. B0 through B2 means 110 through 112.

Columns 24 and 25, columns 27 and 28, and columns 30 and 31 can contain output indicators to specify the conditions under which a line is written. If these indicators are on, the output operation occurs.

Figure 85 shows how output indicators condition the printing of an entire line or of a single field.

If no output indicators are specified, the line is written every time that record is checked for output. If no output indicators are specified on a heading or detail line, that line is also written at the beginning of the program cycle.

For more information about indicators, see Chapter 13, “Using Indicators.”

Column 23, 26, or 29 can contain N to indicate that the output operation occurs only if the indicator coded in columns 24 and 25, 27 and 28, or 30 and 31 is not on. An N plus an indicator is called a negative indicator. No output line should be conditioned by negative indicators only; at least one of the output indicators should be positive. If a heading or detail line is conditioned by negative indicators only, the line is written at the beginning of the program cycle when the first-page lines (those conditioned by the 1P indicator) are written.

Columns 32 through 37 can contain an EXCPT name if column 15 contains E.

```
*...1....+...2....+...3....+...4....+...5....+...6....+...7..*  
OName++++DFBASbSaNO1N02N03Excptn........................................  
OPRINT  D 1  44  
0................NO1N02N03Field+YBEnd+PConstant/editword++++++++++...*  
0  INVOIC  10  
0  AMOUNT  18  
0  CUSTR  65  
0  SALSMN  85  
```

Figure 85. Using Output Indicators to Condition the Printing of an Entire Line or of a Single Field

One indicator is used to condition an entire line of printing. When 44 is on, the fields named INVOIC, AMOUNT, CUSTR, and SALSMN are all printed.
A control-level indicator is used to condition when one field should be printed. When indicator 44 is on, fields INVOIC, AMOUNT, and CUSTR are always printed. However, SALSMN is printed for the first record of a new control group only if 44 and L1 are on.

Field-Description Entries

Columns 23 through 31 can contain output indicators to specify the conditions under which a field is written.

Columns 32 through 37 can contain one of the following to specify each field that is written:

- Any field name or data structure name that was used in this program
- The reserved words PAGE, PAGE1 through PAGE7, *PLACE, UDATE, UDAY, UMONTH, or UYEAR
- An array name, an array element, or a table name

For more information, see “Columns 32-37 (Field Name)” on page 612 in Chapter 25, “Output Specifications.”

Column 38 can contain an edit code. For information about edit codes, see Chapter 16, “Editing Numeric Fields.”

Column 39 can contain B to indicate that the field named in columns 32 through 37 is reset to blanks or zeros after the output operation is complete. Leave these columns blank if the field is not reset.

Columns 40 through 43 can contain the end position of a field or constant in the output record. This entry cannot exceed the record length, which is specified in columns 24 through 27 of the file description specifications for this file.

Columns 45 through 70 can contain a constant or edit word. For information about edit words, see Chapter 16, “Editing Numeric Fields.”

AND and OR Lines

Use an AND line if you need more than three indicators to condition an output operation. Code the word AND in columns 14 through 16 of each additional line. The conditions for all indicators in an AND relationship must be satisfied in order for the output operation to occur. You can use any number of AND lines for an output operation.
OUTPUT SPECIFICATIONS

Use an OR line if you want an output operation to occur if any one of two or more conditions is satisfied. Code the word OR in columns 14 and 15 of each additional line. You can use OR lines for an output operation.

If you use a combination of AND and OR lines for an output operation, you can use any number of AND lines and OR lines.

AND and OR lines can condition entire output lines, but they cannot condition individual fields (see Figure 87). However, you can use more than three indicators to condition an output field by using the SETON operation code in calculations. For example, suppose you use indicators 10, 12, 14, 16, and 18 to condition an output field named PAY. In the calculation specifications you can set on indicator 20 if indicators 10, 12, and 14 are on, and then in the output specifications you can use indicators 16, 18, and 20 to condition the output field PAY.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
OName++++DFBASbSaN01N02N03Excptn.................................................*
0*
0* The detail line is printed if either of two sets of conditions is met. If 21, 40, 01, and 16 are all on, the line is printed; if 0* 21 and 40 are on and 01 and 16 are off, the line is also printed. 0*
0* OTRSACTN D 21 40 01
 0  AND  16
 0  OR   21 40N01
 0  AND  N16
0..................N01N02N03Field+YBEnd+PConstant/editword++++++++++++...*
0                  NAME  15
0                  ACCTNO  25
0                  ADDR   60
0                  BALNC  70

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
OName++++DFBASbSaN01N02N03Excptn.................................................*
0* OTRSACTN D 21 40 01
 0  AND  16
 0  OR   21 40N01
 0  AND  N16
0..................N01N02N03Field+YBEnd+PConstant/editword++++++++++++...*
0                  NAME  15
0                  ACCTNO  25
0                  ADDR   60
0*
0* A maximum of three indicators can be used to condition a field.
0*
0                  MR L1 02BALNC     70

Figure 87. Output Indicators in AND and OR Lines
If you use a control-level indicator (L0 through L9) in an OR relationship with the last-record (LR) indicator, the output operation might occur twice when the last-record indicator is on. One operation occurs when the last record is processed, and the other occurs at detail or total time. Figure 88 on page 265 shows how to use control-level indicators and the last-record indicator correctly in an OR relationship.

*.. 1 ...+.. 2 ...+.. 3 ...+.. 4 ...+.. 5 ...+.. 6 ...+.. 7 ..*
OName++++DFBASbSaN01N02N3Excptn...............................
00UT  D LINLR
0  OR LR

Figure 88. Correct Use of Control-Level Indicators and the Last-Record Indicator in an OR Relationship

**Handling Overflow**

Overflow is the condition that exists when a printer reaches the last line printed on a form. There are three ways to handle overflow:

- Automatic overflow
- Overflow indicators
- Fetch overflow routine.

**Automatic Overflow**

If columns 33 and 34 of the file description specifications are blank and you do not use the fetch overflow routine, the program automatically advances the forms when overflow occurs and continues printing on the line one inch from the top of the new page (line 06 if you have 66 lines per page and six lines per inch).

The following steps occur during automatic overflow:

1. All remaining detail lines in that program cycle are printed if a printer operation spaced or skipped to the overflow area.
2. All remaining total lines in that program cycle are printed.
3. The printer skips to the line one inch from the top of the new page. Therefore, detail lines begin on that line (normally line 06) for all pages after the first.

If you use line counter specifications, overflow occurs at the line coded in columns 20 through 22 of those specifications. If you do not use line counter specifications, overflow occurs six lines before the line number coded as the LINES option of the OCL PRINTER statement.

**Overflow Indicators**

You can use OA through OG or OV as an overflow indicator in columns 24 and 25, 27 and 28, or 30 and 31 of the output specifications. An overflow indicator conditions the lines in the PRINTER file that will print when overflow occurs. No more than one overflow indicator can be assigned to each PRINTER file in a program, and no overflow indicator can be assigned to more than one PRINTER file in a program. To use an overflow indicator in the output specifications, you must also assign the same overflow indicator to the PRINTER file in columns 33 and 34 of the file description specifications.
The RPG program cycle allows the overflow indicator to turn on at three different times: at total time, at detail time, and at calculation time if exception output is used. However, the only time that the program checks to see if the overflow indicator is on is right after all total records are printed, unless the fetch overflow routine was specified by an F in column 16.

When the overflow indicator turns on, the following steps occur:

1. Detail lines are printed (if that part of the program cycle is not already complete).
2. Total lines are printed (if conditions are met).
3. Total lines conditioned by the overflow indicator are printed.
4. Heading lines and detail lines conditioned by the overflow indicator are printed.
5. The overflow indicator turns off.

**Coding Overflow Indicators**

When you code overflow indicators in the output specifications, consider the following:

- Spacing past the overflow line turns on the overflow indicator.
- Skipping past the overflow line to any line on the same page turns the overflow indicator on.
- Skipping past the overflow line to any line on a new page does *not* turn the overflow indicator on.
- A skip to a new page coded on a line not conditioned by an overflow indicator turns the overflow indicator off before the forms advance to a new page.
- Control-level indicators can be used with an overflow indicator so that each page contains information from only one control group (see Figure 89 on page 267).
- You can code an overflow indicator on AND or OR lines. However, only one overflow indicator can be associated with one group of output indicators.
- If you use an overflow indicator in an AND relationship with a record-identifying indicator, you may get unusual results because the record type might not be the one read when overflow occurred. In that case, the record-identifying indicator would not be on, so all lines conditioned by both overflow and record-identifying indicators would not print.
- An overflow indicator can be specified on the record-identification line of the output specifications only for a heading, detail, or total record (column 15 contains H, D, or T).
- An overflow indicator cannot be specified on the record-identification line for an exception record (E in column 15). However, an overflow indicator can condition fields within the exception record.
- You can turn overflow indicators on and off by using the SETON and SETOF operation codes.
HANDLING OVERFLOW

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn...........................................* 0*

* Headings are printed on every page: first page, every overflow
* page, and each new page started when indicator L2 is on.
* 0*

* Headings are printed at the top of a new page (skip to 6) only
* when an overflow occurs, that is when indicator OA is on and
* indicator L2 is not on.
* 0*

OPRINT H 306 OANL2
0*

* When a new control group begins, that is when only the indicator
* L2 is on, headings are printed on a new page. In this way, no
* duplicate headings occur, because indicators L2 and OA are not
* on at the same time.
* 0*

After the program reads the first record, headings are printed
* on the first page because the first record always turns on
* indicator L2 (it causes a control break) if control fields
* are specified on the record.
0*

0 OR L2
0.......................N01N02N03Field+YBEnd+PConstant/editword...............

0 8 'DATE'
0 18 'ACCOUNT'
0 28 'N A M E'
0 46 'BALANCE'

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn...........................................* 0*

* If you want to print certain fields on every page: a skip to 06
* is done when either indicator OA is on (an overflow conditon
* occurs), or indicator L2 is on (a control break occurs).
* 0*

* The NL2 indicator prevents the line from printing and skipping
* twice in the same cycle.
0*

OPRINT D 306 OANL2
0 OR L2
0.......................N01N02N03Field+YBEnd+PConstant/editword...............

0 ACCT 8

Figure 89. Using Control-Level Indicators with an Overflow Indicator

Chapter 8. Using a PRINTER File 267
Figure 90 on page 268 shows the setting of overflow indicators when overflow is handled by overflow indicators and when it is handled by the fetch overflow routine. Both normal output and exception output are shown for each case. The solid lines show when the overflow indicator is on. The dashed lines show connections between the end of one program cycle and the beginning of the next.

**Figure 90. The Setting of Overflow Indicators during Overflow Handled by Overflow Indicators and by the Fetch Overflow Routine**

<table>
<thead>
<tr>
<th>Program Cycle</th>
<th>Using Overflow Indicators</th>
<th>Using Fetch Overflow Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read a record</td>
<td>Normal Output</td>
<td>Normal Output</td>
</tr>
<tr>
<td></td>
<td>Overflow</td>
<td>Overflow</td>
</tr>
<tr>
<td></td>
<td>Detail Output</td>
<td>Total Output</td>
</tr>
<tr>
<td>Process all calculations conditioned by control-level indicators (columns 7-8 of calculation specifications)</td>
<td>Normal Output</td>
<td>Overflow</td>
</tr>
<tr>
<td></td>
<td>Detail Calc.</td>
<td>Total Calc.</td>
</tr>
<tr>
<td>Total output</td>
<td>Normal Output</td>
<td>Normal Output</td>
</tr>
<tr>
<td>Overflow output</td>
<td>Overflow</td>
<td>Overflow</td>
</tr>
<tr>
<td>T = Total</td>
<td>Detail Output</td>
<td>Total Output</td>
</tr>
<tr>
<td>H = Heading</td>
<td>Overflow</td>
<td>Overflow</td>
</tr>
<tr>
<td>D = Detail</td>
<td>Detail Calc.</td>
<td>Total Calc.</td>
</tr>
<tr>
<td>Process all calculations not conditioned by control-level indicators (columns 7-8)</td>
<td>Normal Output</td>
<td>Overflow</td>
</tr>
<tr>
<td>Heading and detail output</td>
<td>Normal Output</td>
<td>Overflow</td>
</tr>
<tr>
<td>Set off control-level indicators</td>
<td>Normal Output</td>
<td>Overflow</td>
</tr>
</tbody>
</table>

---

...
Fetch Overflow Routine

The fetch overflow routine allows you to change the overflow logic of the RPG program cycle. You can advance forms when total, detail, or exception records are printed instead of waiting for the usual time in the program cycle.

Use the fetch overflow routine if printing a particular line would cause overflow and if not enough space is left on the page to print the remaining detail, exception, or total output lines. The fetch overflow routine can prevent printing over the page perforation and can ensure use of as much of the page as possible. To determine when to use the fetch overflow routine, study all possible overflow situations. By counting lines and spaces, you can calculate what happens if overflow occurs on each detail and total line.

To use the fetch overflow routine, code F in column 16 of the output specifications. Each time the program finds the F in column 16, it tests to determine if the overflow indicator assigned to the PRINTER file is on. If the overflow indicator is on, the fetch overflow routine occurs in the following sequence:

1. All total lines conditioned by the overflow indicator are printed.
2. 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 the overflow indicator.
3. Heading lines and detail lines conditioned by the overflow indicator are printed.
4. The line containing the F in column 16 is printed.
5. Any detail, exception, and total lines left to be printed for that output cycle are printed.

The fetch overflow routine does not automatically cause forms to advance; forms advance only if columns 19 and 20 or columns 21 and 22 of the output specifications for the overflow-conditioned line contain a two-digit entry that is less than the line number that the printer is currently on.

Column 16 of each OR line must contain an F if the fetch overflow routine is used for each record in an OR relationship. The fetch overflow routine cannot be used when an overflow indicator is coded in columns 23 through 31 of the same specification line. If this occurs, the overflow routine is not called.

Spacing and Skipping

Spacing means advancing the form in the printer a specified number of lines. Skipping means advancing the form in the printer to a specified line. Spacing and skipping can be specified both before and after a line is printed. If both spacing and skipping are specified on the same line, they occur in this order:

1. Skip before
2. Space before
3. Skip after
4. Space after.

With spacing, the maximum number of blank lines that can occur between two lines of print is five. If six spaces are specified (three after the preceding print line and
three before the current print line), the printer spaces six lines and begins printing
on the sixth line.

Spacing or skipping to the overflow line or past the overflow line turns the overflow
indicator on. However, skipping past the overflow line to a line on the next page
does not turn the overflow indicator on. Therefore, if you want to turn the overflow
indicator on when you skip to the next page, use a SETON operation to turn on the
overflow indicator to condition overflow operations.

Skipping is usually done when a new page is needed. A skip to a lower line
number means advancing to a new page. Skipping can also be specified when
more than five blank lines are required between two lines of print. The entry for
skipping must be a two-digit number that indicates the number of the next line
printed. The skip entry must not be a higher number than the form length coded in
columns 15 through 17 of the line counter specifications. If you code a skip to the
line number that the forms are already positioned on, the forms do not move.

If columns 17 through 22 of the output specifications are blank, single spacing
occurs after each line is printed. Separate spacing and skipping entries can be
coded for each record in an OR relationship. If no spacing or skipping entries are
coded for an OR line, spacing and skipping are done according to the specifications
for the line before that OR line. No spacing or skipping can be specified on AND
lines.

Sample Program

Figure 91 on page 272 shows sample program AR936R, which prints a trial
balance report. The program reads the customer master file (CUSTMAST) and
prints a report depending on the option that the user selects. The report options
are 1, 2, 3, and 4. If the user selects 1 = All customers, the program prints the
balance from each customer. If the user selects 2 = All customers with balances,
the program prints only those customers whose balance is not zero. If the user
selects 3 = All customers with balances over the amount entered in field OVBAL,
the program prints only those customers whose balances exceed the amount speci-
fied by the user in the OVBAL field. If the user selects 4 = All customers with
overdue balances, the program prints only those customers whose balance is
overdue.

The following is an explanation of the indicator definitions used in this sample
program:

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Record-identifying indicator for the CUSTMAST file.</td>
</tr>
<tr>
<td>02</td>
<td>Indicator conditions first-time processing.</td>
</tr>
<tr>
<td>03</td>
<td>Indicator conditions first-time headings.</td>
</tr>
<tr>
<td>05</td>
<td>Indicator conditions printing and record count.</td>
</tr>
<tr>
<td>10</td>
<td>Indicator turns on if option 1 is chosen.</td>
</tr>
<tr>
<td>20</td>
<td>Indicator turns on if option 2 is chosen.</td>
</tr>
<tr>
<td>30</td>
<td>Indicator turns on if option 3 is chosen.</td>
</tr>
<tr>
<td>40</td>
<td>Indicator turns on if option 4 is chosen.</td>
</tr>
<tr>
<td>LR</td>
<td>Last record processed.</td>
</tr>
</tbody>
</table>
OA PRINTER file page overflow.

Running this program: AR935R must be run before AR936R. To run both AR935R and AR936R, code the following procedure:

```plaintext
// LOAD AR935R
// WORKSTN UNIT-?WS?,RESTORE-YES
// RUN
* If F7 is pressed in AR935R, a nonblank character
* is placed in position 9 of the local area and
* AR936R is not run.
// IFF?L'9'/?GOTO NOPRT
// LOAD AR936R
// FILE NAME-CUSTMAST
// RUN
// TAG NOPRT
```
Sample program AR935R, which maintains the accounts receivable information in the customer master file CUSTMAST, is shown in Chapter 7, “Using a WORKSTN File.” AR935R must be run before AR936R can be run.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
H...OLExeD..CDYI....S........I...1.F.H.........T.................*
H 014

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+.......Exit++......A....U+.*
FCUSTMASTIP  F 256 256  8AI  2 DISK
FPRT O  132 132 OA PRINTER

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.......................*
ICUSTMASTNS 01  1 C
I.................................PFromTo++DField+L1M1FrPoNeEq...*
I 2  9 CUSNO
I 10  34 CUSNM
I P 124 1270CRLIM
I P 131 1352AMDUE
I P 150 1542CHGTD
I P 155 15920VR30
I P 160 16420VR60
I P 165 16920VR90
I*
I* This is the local data area information passed to AR936R from AR935R.
I*
I UDS
I 1  1 OPTION
I 2 800VBAL

Figure 91 (Part 1 of 4). Sample Program AR936R (Printing Accounts Receivable)
SAMPLE PROGRAM

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CCIN01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C*
C* First-time logic to determine which report to print.
C*
C  02 SETON 03
C N02 OPTION COMP '1' 10
C N02 OPTION COMP '2' 20
C N02 OPTION COMP '3' 30
C N02 OPTION COMP '4' 40
C N02 SETON 02
C*
C* Calculate the total overdue payments for each customer.
C*
C Z-ADDOVR3 OVRTOT 92
C ADD OVR60 OVRTOT
C ADD OVR90 OVRTOT
C*
C* If option 1 is chosen, print a report of all customers.
C*
C  10 SETON 05
C*
C* If option 2 is chosen, print a report only of customers with a balance.
C*
C  20 AMDUE COMP *ZERO 0505
C*
C* If option 3 is chosen, print a report of customers with balances over a certain amount.
C*
C  30 AMDUE COMP OVBAL 05
C*
C* If option 4 is chosen, print a report of customers with overdue balances.
C*
C  40 OVRTOT COMP *ZERO 0505

Figure 91 (Part 2 of 4). Sample Program AR936R (Printing Accounts Receivable)
SAMPLE PROGRAM

C*
C* Count the number of records printed.
C*
C  05 ADD 1 CUSCNT 50

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn........................................
OPRT D 206 01N03
  OR 0A
0...............N01N02N03Field+YBEnd+PConstant/editword++++++++++*
0 24 'AGED TRIAL BALANCE REPORT'
0 25 'T'
0 UPDATE Y 34
0 10 54 'FOR ALL CUSTOMERS'
0 20 61 'FOR CUSTOMERS WITH A BAL'
0 20 65 'ANCE'
0 30 61 'FOR CUSTOMERS WITH A BAL'
0 30 70 'ANCE OVER'
0 30 OVBAL J 81 '$'
0 40 61 'FOR CUSTOMERS WITH OVERD'
0 40 72 'UE BALANCES'
0 100 'PAGE'
0 PAGE 105
0 D 2 01N03
0 OR 0A
0 8 'CUST NUM'
0 24 'CUSTOMER NAME'
0 46 'AMOUNT DUE'
0 57 'CHARGES'
0 70 'OVERDUE 30'
0 82 'OVERDUE 60'
0 94 'OVERDUE 90'
0 109 'TOT OVERDUE'
0 120 'CRED LIM'

Figure 91 (Part 3 of 4). Sample Program AR936R (Printing Accounts Receivable)
0     D  1      01 05
0             CUSNO     8
0             CUSNM     34
0            AMDUE J  46  '$'
0            CHGTD J  58  '$'
0            OVR30 J  70  '$'
0            OVR60 J  82  '$'
0            OVR90 J  94  '$'
0           OVRTOTJ 108  '$'
0            CRLIM J 120  '$'
0         T  2      LR
0             CUSCNT1   6
0                         31  'CUSTOMER RECORDS PRINTED'

Figure 91 (Part 4 of 4). Sample Program AR936R (Printing Accounts Receivable)
Chapter 9. Using a SPECIAL File

An RPG program can process files that use input and output devices not directly supported by RPG. To use such a file you must code the device name SPECIAL on the file description specifications and provide a subroutine to transfer data between the SPECIAL device and main storage. That subroutine can be SUBRO1, supplied by IBM, or a subroutine that you write yourself.

This chapter contains information on using SPECIAL files. This includes:
- File description specifications
- Restrictions for special files
- Using a subroutine for input and output.

File Description Specifications

To use a SPECIAL file, code the unshaded columns in line 02 on the file description specifications as shown below. If you use a continuation line, also code the unshaded columns in line 03:

Columns 7 through 14 must contain the name of the file.

Column 15 can contain I, O, U, or C to indicate that the file is an input, output, update, or combined file.

Column 16 can contain P, S, or D to indicate that the file is a primary, secondary, or demand file. Column 16 must be blank if column 15 contains O (alphabetic character).

Column 17 must contain E if the program must process every record from the file before the program can end. Leave column 17 blank if the program can end before it processes every record in the file. Column 17 applies only to input, update, or combined files used as primary or secondary files.

Column 18 can contain A, D, or blank. A indicates that the program checks that the records are in ascending sequence. D indicates that the program checks that the records are in descending sequence. Blank indicates that the program does not check the record sequence. Column 18 applies only to input, update, or combined files used as primary or secondary files.

Column 19 must contain F or blank to indicate that all records in the file have the same length.
Columns 20 through 23 are used to specify the block length, which has no relevance in RPG for the AS/400 system. Any value entered in these columns is ignored.

Columns 24 through 27, the record length, must contain a number from 1 through 4096.

Columns 28 through 31 must be blank because SPECIAL files can only be processed consecutively.

Column 32 is used to specify the number of input/output areas, which has no relevance in RPG for the AS/400 system. Any value entered in this column is ignored.

Columns 40 through 46 must contain the device name SPECIAL.

Columns 54 through 59 must contain the name of the subroutine that does the input and output operations between the SPECIAL device and main storage. You can use SUBR01 for this purpose. The subroutine name must be in the form SUBRxx, where x is any alphabetic character (numeric characters are reserved for subroutines supplied by IBM), or in the form SRYzzz, where y is any of the following 15 characters: B, C, D, F, G, H, I, L, M, O, P, R, S, T, or U, and z is any of the following 16 characters: A, B, C, D, F, G, H, I, L, M, O, P, R, S, T, or U.

Columns 71 and 72 can contain an external indicator, U1 through U8.

Continuation Line

Column 53 must contain K to indicate that this continuation line provides more information about the SPECIAL file coded on the preceding line. Only one continuation line can be used for each SPECIAL file.

Column 54 through 59 must contain the name of a table or array used by the subroutine that you wrote to do input and output for the file.

Restrictions for SPECIAL Files

You can use the following with SPECIAL files:
- FORCE operation in the calculation specifications
- READ operation in the calculation specifications
- File translation (column 43 of the control specification).

You cannot use the following with SPECIAL files:
- CHAIN operation in the calculation specifications
- Spacing and skipping (columns 17 through 22 of the output specification).

Using a Subroutine for Input and Output

Because RPG does not support a SPECIAL device directly, you must provide a subroutine to transfer data between the SPECIAL device and main storage. You can use SUBR01, supplied by IBM, or you can write your own subroutine.
Using IBM’s Subroutine, SUBR01

Subroutine SUBR01 reads records from the system source of input. (If you enter OCL statements from the display station keyboard, the display station is the source of input. If the OCL statements are in a procedure, the procedure is the source of input.) The records can be 120 or 512 characters long. RPG treats the records read by SUBR01 as data records. To use this subroutine, code SUBR01 in columns 54 through 59 of the file description specifications for a SPECIAL device:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P, S, or D to indicate that the file is a primary, secondary, or demand file.

Column 19 must contain F or blank to indicate that all records in the file have the same length.

Columns 20 through 23 (block length) and 24 through 27 (record length) must contain 120 or 512. If you do not specify a block length, the block length is assumed to equal the record length. If you enter the OCL statements to run the program from the keyboard, the records made available to the file must also be entered from the keyboard. If you call a procedure to run the program, the records to be made available to the file must follow the RUN statement in the procedure.

The last input record in the procedure should be followed by the OCL END statement. If the program uses only one display station and if there is no END statement in the procedure, the program treats the next OCL statements (entered from the keyboard) as input to the SPECIAL file in the program. If the program uses more than one display station, you must use a procedure for the data records, and an END statement must follow the last data record.

If a program is run from the input job queue, you must use a procedure to run the program. If the program that uses SUBR01 also uses a CONSOLE file, the OCL statements and that data records for SUBR01 must be contained in a procedure. Otherwise, undesirable results can occur.

See System Reference for the System/36 Environment for information on how to create a procedure.
Using Your Own Subroutine

You can write your own subroutine to specify an input/output device that is not directly supported by RPG. Enter SPECIAL in positions 40 to 46 of the file description specifications. On the same file description specification line, enter the name of your subroutine in positions 54 to 59.

RPG uses this user-written routine to open the file, to read and write records, and to close the file. It creates a parameter list for your user-written routine to use. The parameter list contains an option code parameter (option), a return status parameter (status), an error-found parameter (error), a record area parameter (area), a user-specified array or table (array), and a final parameter (last) that defines the attributes of all the preceding parameters.

This parameter list is accessed by RPG and by your user-written routine; it cannot be accessed by the RPG program that contains the SPECIAL file.

The parameters in this parameter list are as follows:

- **Option**: The option parameter is a 1-position character field, which indicates the action that your user-written routine is to do. RPG passes one of the following values to your user-written routine, depending on the operation being processed on the SPECIAL file (Open, Close, Read, Write, Delete, Update):

<table>
<thead>
<tr>
<th>Value Passed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>Open the file.</td>
</tr>
<tr>
<td>C</td>
<td>Close the file.</td>
</tr>
<tr>
<td>R</td>
<td>Read a record and place it into the area defined by the area parameter.</td>
</tr>
<tr>
<td>W</td>
<td>RPG has placed a record into the area defined by the area parameter; it is to be written out.</td>
</tr>
<tr>
<td>D</td>
<td>Delete the record.</td>
</tr>
<tr>
<td>U</td>
<td>The record is an update of the last record that was read.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal return. The requested action was done.</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>The requested action was not done; an error condition exists.</td>
</tr>
</tbody>
</table>

- **Error**: The error parameter is a five-digit zoned numeric field with zero decimal positions. If your user-written routine detects an error, the error parameter contains an indication or value representing the type of error. The value is placed...
into the first five positions of the file information data 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 program.

- **Array**: If the SPECIAL file is a primary file, you can specify an entry parameter list that contains both the RPG-created parameters and a user-specified array or table.

- **Last**: The last parameter passed generates a description for the four preceding parameters. The description contains the following information for each preceding parameter.
  - A 1-byte character field: ‘C’ When the parameter is alphabetic; ‘Z’ When the parameter is numeric.
  - A 4-byte zoned field: Length of the parameter in bytes.
  - A 2-byte zoned field: Number of decimal digits if the parameter is numeric.
  - A 4-byte zoned field: Number of elements if the parameter is an array.

You need to declare an array of length equal to the number of variables that are declared. Each element in the array must have the following structure:

```plaintext
CHAR       Type of element (C or Z)
ZND(4,0)   Length of parameter
ZND(2,0)   Number of decimal digits
ZND(4,0)   Number of array elements
```

The array itself will be based on the last parameter that is declared in the subroutine, because RPG passes the parameters to the subroutine by address.
USING A SUBROUTINE FOR INPUT AND OUTPUT
Chapter 10. Using a CONSOLE, KEYBORD, or CRT File

The device names WORKSTN, CONSOLE, KEYBORD, and CRT all refer to the same object: a display station, which consists of a display screen and a keyboard. However, WORKSTN, CONSOLE, KEYBORD, and CRT files differ in the ways in which they use a display station.

Whenever possible, use a WORKSTN file instead of a CONSOLE, KEYBORD, or CRT file. WORKSTN files offer many advantages. The reason that the AS/400 system allows the use of CONSOLE, KEYBORD, or CRT files is so that programs that used those files on earlier IBM systems can run on the AS/400 system without being rewritten.

As Chapter 7, "Using a WORKSTN File" explains, a WORKSTN file is a combined (both input and output) file. It allows you to specify those fields on the display that are input fields, those that are output fields, and those that are both input and output fields. You can also use a WORKSTN file in a program that allows one or more requesters.

By contrast, CONSOLE, KEYBORD, and CRT files can be used only in programs that allow only one requester (that is, single requester terminal (SRT) programs). In addition:

- A CONSOLE file can be used only as an input file, so you cannot display the records in a CONSOLE file. A CONSOLE file can be used as an input data file to provide data to a program that is running, or as a record address file to provide key fields for processing within key-field limits. A program can use only one CONSOLE file. This also applies to WORKSTN files. The CONSOLE file provides an easy way to create a simple data-entry program, because the program creates input prompts automatically.

- A KEYBORD file can be used as both an input and an output file when you use the KEY and SET operations. These operations allow you to display function keys, prompts and messages, and to respond by entering one field at a time.

- A CRT file can be used only as an output file to display information. You cannot change this information by entering data at the keyboard.

- You can specify only one of each of the following files: KEYBORD, CRT, CONSOLE, and WORKSTN. If you have specified a WORKSTN file, you cannot specify a KEYBORD, a CRT, or a CONSOLE file.

Using a CONSOLE File

To use a CONSOLE file in a program, code entries in the file description specifications and in the input specifications.
File Description Specifications

Code entries in the unshaded columns of the file description specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>Filetype</th>
<th>File Description</th>
<th>File Name</th>
<th>Mode of Processing</th>
<th>Length of Key Field or of Record Address Field</th>
<th>Extent Exit for DAM Storage Index</th>
<th>Number of Tracks for Cylinder Overflow</th>
<th>Number of Extents</th>
<th>Tape Rewind R/U/N</th>
<th>Number of Hours</th>
<th>Type of File Type</th>
<th>Organization or Additional Area</th>
<th>Block Length</th>
<th>Key Field Format</th>
<th>Key Field Length</th>
<th>Record Address Type</th>
<th>Extension Code E/L</th>
<th>I/X/D/T/R or 2</th>
<th>A/P/I/K</th>
<th>L/R</th>
<th>F/V/S/M/D/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>F</td>
<td>I</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 through 14 must contain the name of the CONSOLE file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 must contain P (primary), S (secondary), D (demand), or R (record address) to indicate how the program uses the file.

Column 17 can contain E if column 16 contains P, S, or R. It must be blank if column 16 contains D. E in column 17 indicates that the program must process every record from the file before the program can end. Blank indicates that the program can end before all of the records in the file have been processed. If this column is blank for every file, the program must process every record from every file before the program can end. To indicate that all the records for a CONSOLE file have been entered, the user at the display station presses function key 12. For more information about function keys, see “Allowing Function Keys To Be Pressed” on page 308.

Column 18 can contain A or D if column 16 contains P or S. It must be blank if column 16 contains D or R. A indicates that the program checks that the records in the file are in ascending sequence. D indicates that the program checks that the records are in descending sequence. Blank indicates that the record sequence is not checked.

Column 19 must contain F or blank to indicate that every record in the file has the same length.

Columns 20 through 23 contain the length of a block of records. The block length must be equal to the record length, entered in columns 24 through 27, or be blank.

Columns 24 through 27 contain the record length. The record length must be the same as the highest number coded in columns 48 through 51 on the input specifications (to the field location). This record length cannot be less than 2 or greater than 1518. For more information about calculating record length, see “Using Displays” on page 294. If the CONSOLE file is used as a record address file, determine the record length by multiplying the length of the record address field by 2. This record length cannot be less than 2 or greater than 58.
Columns 29 and 30 must be blank if column 16 contains P, S, or D. If column 16 contains R, columns 29 and 30 must contain the length of the key field of the indexed DISK file.

Column 31 is used only for record address files. Leave the column blank if the key fields in the record address file are the same as the key fields in the indexed DISK file. Enter A for an indexed DISK file with zoned-decimal key fields.

Column 39 must be blank if column 16 contains P, S, or D. If column 16 contains R for record address file, this column must contain E.

Columns 40 through 46 must contain CONSOLE as the device name.

Columns 71 and 72 can contain external indicators, U1 through U8.

Input Specifications

Input specifications are not required (and not allowed) for record address files. Therefore, if column 16 of the file description specifications contains R, do not code any input specifications for that file. However, if column 16 of the file description specifications contains P, S, or D, entries are required in the unshaded columns of the input specifications as shown below:

File and Record Specifications

Columns 7 through 14 must contain the name of the CONSOLE file. The name must be the same as the file name on the file description specifications.

Columns 14 through 16 must not contain the characters AND; however, columns 14 and 15 can contain the characters OR. These OR lines can be used to indicate a relationship between record-identifying indicators or record types. If columns 14 and 15 contain OR, the same number of record identification codes must be described on this specification line as are described on the preceding line.

Columns 15 and 16 can contain any two alphabetic characters if you do not want the program to check the sequence of input records. Code a numeric entry (01 through 99) in these columns to assign a sequence number to each record type in the file. The maximum number of record types that you can use for a CONSOLE file is 10.

Column 17 must be blank if columns 15 and 16 contain alphabetic entries. If columns 15 and 16 contain numeric entries, code 1 in column 17 if the record type can consist of only one record, or code N if the record type can consist of one or more records.
USING A CONSOLE FILE

Column 18 must be blank if columns 15 and 16 contain alphabetic entries. If columns 15 and 16 contain numeric entries, code O in column 18 if the record type is optional.

Columns 19 and 20 must contain a record-identifying indicator (01 to 10) to identify the function key that the person at the display station enters to select this record type. You cannot use the same indicator to identify more than one record type within the input specifications for one program.

Column 24 must contain 1 to indicate that the record identification code is in position 1.

Column 26 must contain C to indicate that the entire character is used as the record identification code.

Column 27 must contain the character that is used as the record identification code in position 1 of the record. In an output only area of the display, the program automatically inserts a one- or two-character record identification code into positions 1 and 2 of each new record that is prompted.

Columns 28 through 34 must be blank if a one-character record identification code is used. If a two-character record identification code is used, code these columns the same as columns 24 through 27, except that column 31 must contain 2 to indicate record position 2.

Field Specifications
Columns 44 through 47 must contain the record location in which the field begins.

Columns 48 through 51 must contain the record location in which the field ends. The maximum length for an alphameric field is 66 characters. The maximum length for a numeric field is 15 digits.

Subfields can be coded within the fields of a CONSOLE file record. The from and to field locations for subfields must not overlap the from and to field locations for another field. The program does not prompt for subfields, but it assigns values from the prompted field to subfields. You can use subfields in calculation and output specifications.

For example, in Figure 92 on page 287, the part number 01ROC43CP843987831 is entered in response to the prompt field PARTNO. LOCATN, WHSE, BIN, ASMTP, and NUMBER are subfields within the PARTNO field. The values for the subfields are taken from the PARTNO field.

Columns 53 through 58 must contain a descriptive field name (one to six alphameric characters) used as a prompt for this data. To enter data into a whole array for a CONSOLE file, define the whole array as a subfield within a field of the CONSOLE file record, or define each element of the array with an index and place this entry in columns 53 through 58.

Columns 59 and 60 can contain a control-level indicator (L1 through L9) if this is a primary or secondary file. A control-level indicator indicates that a control break occurs when the contents of a field change.

Columns 61 and 62 can contain a match-field value (M1 through M9) if this is a primary or secondary file. Otherwise, leave these columns blank.
Creating Display Formats for CONSOLE Files

There are a number of ways to create display formats for a CONSOLE file. They include:

- Using the RPGR procedure.
- Specifying GEN on RPGC or AUTOCC
- Specifying CONSOLE on CRTS36RPG or CRTS36RPT
- Using the AS/400 system command CRTS36RPGR.

Figure 92. Coding Subfields for a CONSOLE File
Using the RPGR Procedure

The RPGR procedure uses the input specifications to create source input to the Display Format Generator (the $SFGR utility) in the System/36 Environment. The display format generator compiles this source input and creates a program object containing the display formats for the program. For a complete description of display formats, see the SDA User’s Guide and Reference.

To call the RPGR procedure directly, type RPGR on the command line of the display. Then you can press Enter (in which case, the following display appears), or you can type RPGR and the parameters that you want to use with the RPGR procedure and then press Enter (in which case, the following display does not appear):

![RPGR Procedure Display](image)

Figure 93. The RPGR Display

Respond to each prompt by entering the appropriate information.

Name of source program containing CONSOLE files: Enter the name of your RPG source program.

Size of $SOURCE file in blocks: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

Save display format source member: Enter SAVE or NOSAVE.

SAVE  Save the source statements for the $SFGR utility in the library specified as the source input library. The name given to the saved source statements is the program name plus FM. (The program name is the name coded in columns 75 through 80 of the control specification.) For example, if the name of the program is PRNAME, the name of the display format source member and program object for that program is PRNAMEFM. The format name cannot be changed after compilation.

NOSAVE  Do not save the source statements for the $SFGR utility.
If you do not choose an option, SAVE is assumed.

Name of library containing source program: Enter the name of the library that contains the RPG source program. If you do not specify a library name, the current library is assumed.

Name of library to contain format load member: Enter the name of the library that will contain the program object created by the $SFGR utility. If you do not specify a library name, the current library is assumed.

Replace duplicate members: Enter REPLACE or NOREPLAC.
- REPLACE: Replace an existing library member with the newly compiled library member that has the same name.
- NOREPLAC: If another library member has the same name, display an error message.

If you do not choose an option, REPLACE is assumed.

Note: If you type the parameters for the RPGR procedure on the command line of your display instead of using the prompts, allow for the GEN parameter at this point. The only possible entry for this parameter is GEN. If you use the prompts, ignore this parameter. This parameter is included only for compatibility with the RPGR procedure on previous systems.

Output option for format listing: Enter PRINT or NOPRINT.
- PRINT: Print a copy of the display formats created by the $SFGR utility and a listing of the $SFGR source specifications.
- NOPRINT: Do not print a copy of the display formats created by the $SFGR utility or a listing of the $SFGR source specifications.

If you do not choose an option, PRINT is assumed.

You can use the Help key from the first RPGR Procedure display to see which function keys to use, and for additional information about parameters.

If more than one display format is created for the same program, the RPGR procedure adds FM to the program name to identify the entire set of display formats, and it adds the record-identifying indicator to the program name to identify each display format. Therefore, if the program PRNAME contains three record types (identified by indicators 01, 02, and 03), the RPGR procedure creates the following names:

- PRNAMEFM, the name of the entire set of display formats
- PRNAME01, the name of the first display format in the set
- PRNAME02, the name of the second display format in the set
- PRNAME03, the name of the third display format in the set.

If OR lines are used on the input specifications to identify the same record, only one format is associated with the record.

It is preferable to leave the program name parameter (PGM) as *CTLSPEC, which is the default. If you choose not to use *CTLSPEC, use a name of six characters or less and make sure it is the same name as that specified on the H-specification. If you specify a name longer than six characters, the Display Format Creation Program will truncate the name to six characters, add the suffix FM, and assign
that name to the display file. This could lead to your accidentally erasing files or using the wrong display files.

Creating Display Formats for CONSOLE Files with CRTS36RPGR

You can use the AS/400 system command CRTS36RPGR to create display formats. This command offers a number of advantages over the RPGR procedure. CRTS36RPGR allows you to create DDS specifications and to save them in a specific source file member that you can specify on the command.

There are two CRTS36RPGR displays. The first displays the parameter options.

![CRTS36RPGR Display, Showing Valid Options](image)

Figure 94. CRTS36RPGR Display, Showing Valid Options
If you press F11, you get the following display, which shows the valid parameter keywords.

![Create Console Display File (CRTS36RPGR)](image)

Figure 95. CRTS36RPGR Display, Showing Keywords

If you press F11 from this display, you return to the display shown in Figure 94 on page 290.

Enter the following information for each parameter.

**Source member (SRCMBR Parameter)**

Specifies the source member that contains the RPG II program specifications.

source-file-member-name: Enter the name of the member in which the source program is stored.

**Source file (SRCFILE Parameter)**

Specifies the file that contains the RPG II source member and the library that contains the file.

File

**QS36SRC**: Specify the default file name.

source-file-name: Enter the name of the source file containing the source member.

Library

*CURLIB*: The current library will be used. If you have not specified a current library, QGPL will be used.

library-name: Enter the name of the library in which the source file is stored.

**Output Library (OUTLIB Parameter)**

Specifies the library that contains the display format object member.

*CURLIB*: The current library will be used. If you have not specified a current library, QGPL will be used.
**USING A CONSOLE FILE**

$output$-library-name: Enter the name of the library in which the object member will be stored.

**File to Receive System/36 Formats (FMTSRCF Parameter)**
Specifies the file in which the member containing the S and D specifications will be stored and the library that will contain the file.

**File**
- **SRCFILE**: Use the file name specified in the source file parameter (SRCFILE).
- **NONE**: Specify that the S and D specifications will not be produced.

**S&D-source-file-name**: Enter the name of the file that will contain the S and D specifications.

**Library**
- **SRCLIB**: Use the same library that you specified in the source file library name (SRCFILE parameter).

**S&D-library-name**: Enter the name of the library that will contain the S and D source file.

**Member to Receive Formats (FMTMBR Parameter)**
Specifies the member that will contain the S and D specifications.

- **CRTDFT**: Use the specified source member name (SRCMBR) with FM appended to it.

**S&D-member-name**: Enter the name of the member that will contain the S and D specifications.

**File to Receive DDS (DDSSRCF Parameter)**
Specify the file in which the member containing the DDS specifications will be stored and the library that will contain the file.

**File**
- **SRCFILE**: Use the file name specified in the source file parameter (SRCFILE).
- **NONE**: Specify that the DDS specifications will not be produced.

**DDS-source-file-name**: Enter the name of the source file that will contain the DDS member.

**Library**
- **SRCLIB**: Use the same library that you specified in the source file library name (SRCFILE parameter).

**DDS-library-name**: Enter the name of the library that will contain the DDS source file.

**Member to Receive DDS (DDSMBR Parameter)**
Specify the member that will contain the DDS specifications.

- **CRTDFT**: Use the specified source member name (SRCMBR) with A appended to it.

**DDS-member-name**: Enter the name of the member that will contain the DDS specifications.
If you press F10, you get the following display, which shows the valid parameter options.

![Figure 96. CRTS36RPGR Display with Optional Parameters, Showing Valid Options](image1)

If you press F11 from this display you will get the following display, which includes two optional parameters.

![Figure 97. CRTS36RPGR Display With Optional Parameters, Showing Keywords](image2)

If you press F11 from this display you will return to the display shown in Figure 96.
Using a Console File

The optional parameters are described below.

**Generate CONSOLE Formats (GEN Parameter)**

Specifies that the display formats for the CONSOLE display file will be created.

- **YES**: Specify that a new CONSOLE display file will be created.
- **NO**: Specify that a new CONSOLE display file will not be created.

**Replace Existing File (REPLACE Parameter)**

Specifies that a new CONSOLE display file will be created without first deleting any existing file with the same name.

- **YES**: Specify that a new CONSOLE display file will be created and any existing CONSOLE display file of the same name in the specified library will be deleted.
- **NO**: Specify that a new CONSOLE display file will not be created if a member of the same name exists in the specified library.

**Using Displays**

Displays prompt the user at the display station to enter data. The user presses function key 12 to indicate end of file (that is, there is no more data to enter). For information about function keys, see “Allowing Function Keys To Be Pressed” on page 308.

**Display Formats**

The top line on the display contains control information that the user needs to identify the current record and to specify the next record type to be prompted (see Figure 98 on page 295). The remaining 23 lines are used for the formatted record. The maximum number of input fields that can be displayed is 80.

For each field defined, the system reserves 14 characters to contain the field name and its attributes. Therefore, the maximum record length is 1518 characters. (23 lines on the display format x 80 characters per line = 1840 characters on the display format. 23 lines x 14 characters reserved for each field (line) = 322 reserved characters. 1840 total characters - 322 reserved characters = 1518 characters available for the record.) The format actually created for a record depends on the size and number of fields in the record.

The screens shown in Figure 98 on page 295 to Figure 101 on page 297 show examples of different display formats. The possible display formats are:

- **One column format.** The compiler creates a one-column display format whenever the number of fields prompted for is less than 24. See Figure 98 on page 295.

- **Two column format.** The compiler creates a two-column display format whenever the number of fields is 24 through 46 (see Figure 99 on page 296). If any field is longer than 26 characters, the display format is changed to allow these fields.

- **Three column format.** The compiler creates a three-column display format whenever the number of fields is 47 through 69 (see Figure 100 on page 296). If any field is longer than 12 characters, the display format is changed to allow these fields.
• Four column format. The compiler creates a four-column display format whenever the number of fields is 70 through 80 (see Figure 101 on page 297). If any field is longer than six characters, the display format is changed to allow these fields.

If the format is changed so that the 4-column format cannot be used, error message RPG1024 Format for CONSOLE file does not fit on screen, is displayed. This message means that you must reduce the number of fields in the record or change the order of the fields. Remember that all fields for the record must fit on one display format.

```
M 1 1,2,3,4 1,2,3,4
ACCTNO A 5
DISCNT N 3.0
```
### Figure 99. Example of Two-Column Display Format Created for a CONSOLE File

<table>
<thead>
<tr>
<th>M</th>
<th>1</th>
<th>1,2,3,4</th>
<th>1,2,3,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTNO</td>
<td>A 5</td>
<td>PHONE</td>
<td>A 11</td>
</tr>
<tr>
<td>NAME</td>
<td>A 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR1</td>
<td>A 20</td>
<td>ADDR2</td>
<td>A 20</td>
</tr>
<tr>
<td>ADDR3</td>
<td>A 20</td>
<td>CITY</td>
<td>A 25</td>
</tr>
<tr>
<td>STATE</td>
<td>A 20</td>
<td>ZIP</td>
<td>A 5</td>
</tr>
</tbody>
</table>

### Figure 100. Example of Three-Column Display Format Created for a CONSOLE File

<table>
<thead>
<tr>
<th>M</th>
<th>1</th>
<th>1,2,3,4</th>
<th>1,2,3,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCTNO</td>
<td>A 5</td>
<td>DISCNT</td>
<td>N 3.3</td>
</tr>
<tr>
<td>SLSMAN</td>
<td>A 40</td>
<td>COST</td>
<td>N10.2</td>
</tr>
<tr>
<td>DIST</td>
<td>A 3</td>
<td>REGION</td>
<td>A 4</td>
</tr>
<tr>
<td>CITYST</td>
<td>A 20</td>
<td>ZIP</td>
<td>A 5</td>
</tr>
</tbody>
</table>
Prompt Format

The display format generator uses the field names on the input specifications to create prompts for these display formats. The prompts are 14 positions long and have the following format:

<table>
<thead>
<tr>
<th>Position</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Control character for the prompt. This character appears as a blank.</td>
</tr>
<tr>
<td>2-7</td>
<td>Field name.</td>
</tr>
<tr>
<td>8</td>
<td>Blank.</td>
</tr>
<tr>
<td>9</td>
<td>N for a numeric field, or A for an alphabetic field.</td>
</tr>
<tr>
<td>10-13</td>
<td>Length of the field. For an alphabetic field, positions 10 and 11 are blank, and positions 12 and 13 contain the length. For a numeric field, positions 10 and 11 contain the length of the field, position 12 contains a decimal point, and position 13 contains the number of decimal positions in the field.</td>
</tr>
<tr>
<td>14</td>
<td>Control character for the input field. This character appears as a blank.</td>
</tr>
</tbody>
</table>

Changing the Display Format

After you have used the RPGR procedure or the CRTS36RPGR command to create the source input for the display format generator and the input has been cataloged in the library, you can change this source input, if you wish, by using the Source Entry Utility (SEU). For information about SEU, see SEU User’s Guide and Reference.
USING A KEYBORD FILE

Erasing the CONSOLE File Buffer

To erase, or blank, the entire buffer for the CONSOLE file, use the SET operation with ERASE coded in the result field. Entries are required in the unshaded columns of the calculation specifications as shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>Indicators</th>
<th>Control Level</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Name</th>
<th>Length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>C</td>
<td></td>
<td></td>
<td>SET</td>
<td></td>
<td>ERASE</td>
<td>CONSOLE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 18 through 27 (factor 1) must be blank.

Columns 28 through 30 must contain the operation code SET.

Columns 33 through 42 (factor 2) must contain the name of the CONSOLE file.

Columns 43 through 47 (the result field) must contain ERASE.

ERASE causes the RPG program to change the contents of the buffer to blanks just before the program reads a record at the beginning of the next program cycle. Because the buffer is not erased until the beginning of the next program cycle, the program continues to process the current record after the ERASE operation occurs.

If the ERASE operation occurs because of invalid input data, you should insert code in your program to avoid further calculations and to return to the start of the program cycle. Then the user at the display station can enter a correct form of the record containing invalid input data and can re-enter any records that were entered after that record.

Using a CONSOLE File with KEYBORD and CRT Files

When a program uses a CONSOLE file, a KEYBORD file, and a CRT file, and the user at the display station is entering data for the CONSOLE file, the following occurs when a KEY or SET operation for the KEYBORD file occurs:

1. The user must finish entering data for the current record in the CONSOLE file.
2. The prompt for the SET or KEY operation, or the output to the CRT file, is then displayed.
3. Normal processing of the CONSOLE file continues after the SET or KEY operation is completed. That is, the user at the display station can enter data for the next records in the CONSOLE file during the next program cycle.

Using a KEYBORD File

A KEYBORD file can be used as both an input and an output file. To use a KEYBORD file, you must code file description specifications. You do not code input or output specifications, however. Instead, you describe the data on the calculation specifications for the KEY operation or for the KEY and SET operations.
File Description Specifications

To create a KEYBORD file, code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the file.

Column 15 must contain I to indicate that the file is an input file.

Column 16 can contain P (primary) or D (demand) to indicate how the program uses the file. If you use a KEYBORD file as a primary input file, no other files can be used as primary or secondary files. In this case, you must provide an exit for your program by turning on the last-record indicator in the calculation specifications. If you use a KEYBORD file as a demand file, you use the KEY operation, not the READ operation, to read records from the file. Therefore a KEYBORD file as a primary file acts just like a demand file.

Column 19 must contain F or blank to indicate that every record in the file has the same length.

Columns 20 through 23, the block length, must equal the record length coded in columns 24 through 27 or be blank.

Columns 24 through 27 must contain the length of the largest field to be entered. This number must equal the largest field length coded in columns 49 through 51 of the calculation specification for the KEY operation. If you use the KEY operation to display a message, you must also consider the length of the message when you code the record length for the KEYBORD file. The maximum length for an alphanumeric field is 79 characters. The maximum length for a numeric field is 15. If the record length coded for a KEYBORD file is 40 or less, a display of six lines with 40 characters per line is centered both vertically and horizontally. If the record length is more than 40, the display consists of 24 lines with 79 characters per line.

Columns 40 through 46 must contain KEYBORD.

Calculation Specifications for a KEY Operation

Although a KEYBORD file is an input file, you do not code input specifications for a KEYBORD file. Instead, you define the input data on the calculation specifications for a KEY operation. The KEY operation causes a pause in calculations. During that pause, the user at the display station can enter data from the keyboard.
To use the KEY operation, code entries in the unshaded columns of the calculation specifications as shown below:

Columns 7 and 8 can contain a control-level indicator (L1 through L9), AN, OR, or blanks. Leave these columns blank if the KEY operation is not part of a subroutine or if it occurs only at detail time.

Columns 9 through 17 can contain conditioning indicators, function-key indicators (KA through KN, KP through KY) coded in a SET or SETOF operation, or blanks.

Columns 18 through 27 (factor 1) can contain the constant, literal, field name, or table or array element displayed.

Columns 28 through 30 must contain the operation code KEY.

Columns 31 and 32 can contain the message identification code (01 through 99) corresponding to the displayed message. The message length is truncated to the record length. The message itself is in your message member. For information on how to create a message member, see $MGBLD Utility Program in the System Reference for the System/36 Environment. This message prompts the user at the display station to do a KEY operation. An entry is required in columns 31 and 32 when columns 18 through 27 are blank. If you do not code the OCL MEMBER statement that specifies your message member before you run the program, or if columns 31 and 32 contain a message identification code that does not correspond to a message in your message member, or if the message file does not exist, the computer displays the prompt nn-Message indicator, where nn is the contents of columns 31 and 32. If factor 1 contains an entry that prompts the KEY operation, the message identification code in columns 31 and 32 is ignored.

Columns 33 through 42 (factor 2) must be blank.

Columns 43 through 48 (the result field) can contain the name of the field entered.

Columns 49 through 51 must contain the length of the field to be entered if the field is not defined somewhere else. The maximum length for a numeric field is 15. The maximum length for an alphameric field is 40 if the record length is 40 or less, or 79 if the record length is more than 40.

Column 52 must be blank for alphameric fields. For numeric fields, enter the number of decimal positions (0 through 9) in the field to be entered if that field is not defined somewhere else.

Columns 54 through 59 can contain resulting indicators (01 through 99) to test the condition of a numeric field (plus, minus, or zero) or to test an alphameric field for blanks (columns 58 and 59).
Figure 102 on page 301 shows examples of KEY operations.

C* Keying operations with user message member prompts.
C*
C* The following operations allow the operator to key a numeric field (FIELDA) and an alphabetic field (FIELDB). These fields have not been defined previously. The operations are prompted by messages 0001 and 0002 from the user message member, respectively.
C*
C C KEY01 FIELDA 50
C C KEY02 FIELDB 12
C*
C* The following operation allows the operator to key a numeric field defined previously. This field is tested for a plus, minus, or zero condition. The operation is prompted by user message 0030.
C*
C C KEY30 AMOUNT 010203
C*
C* Display keying operations with factor 1 prompts.
C*
C* The following operations cause the previously defined field (FIELDC) in factor 1 to be displayed and then allow the operator to key a numeric field (FIELDA). The numeric literal 40 is displayed and the operator is allowed to key an alphabetic field (FIELDB). FIELDA and FIELDB are not defined elsewhere. Note that factor 1 overrides user messages 0004 and 0005.
C*
C C FIELDC KEY04 FIELDA 50
C 40 KEY05 FIELDB 12
C* The following operation displays the alphameric literal specified in factor 1 (ALTER) on the display screen. The operator is then allowed to type data into the numeric field specified in the result field defined elsewhere. Factor 1 overrides user message 0006.

C* C 'ALTER' KEY06 AMOUNT 040506

Figure 102 (Part 2 of 2). Possible KEY Operations

Using a KEY Operation

As the user at the display station types data, it is displayed in one of two formats:

- If the record length is 40 or less, the display consists of six lines with 40 characters per line. The display is centered vertically and horizontally.
- If the record length is more than 40, the display consists of 24 lines with 79 characters per line. The computer reserves one character per line for field attributes.

When the user at the display station uses the KEY operation, the contents of the result field depend on the user's response. The possible responses are:

- **The user types the data and presses an entry key.** The user can use any of the following as an entry key: Field Exit, Field-, Field+, or Enter. However, if the user enters data into a numeric field, the Enter key cannot be used as an entry key. If the user does not type data into all positions of a numeric field, the data is moved into the rightmost positions of the field and zeros are placed in the unused positions to the left. If the user does not type data into all positions of an alphameric field, the computer leaves the data in the leftmost positions of the field and puts blanks in the unused positions to the right.

- **The user presses only an entry key.** This action causes the computer to change any numeric data in the result field to zeros or any alphameric data to blanks.

- **The user presses the Dup key and then an entry key.** This action does not change the data in the result field.

Bypassing a KEY Operation

When the KEY operation causes a pause in the calculations, the user at the display station can go to the next calculation without entering any data for the current calculation. To do this, the user simply presses an entry key as described in "Using a KEY Operation." This action changes the data in the result field to zeros or blanks. The user can also press the Dup key, followed by the Enter key. This action will not change the result field value. After each KEY operation (if data is entered or not), the user must press an entry key before the next operation can occur.
Using a Message Member

You can create messages or prompts that will be displayed during your RPG program. These messages or prompts must be in your message members (see $MGBLD Utility Program in the System Reference for the System/36 Environment). The messages or prompts are displayed when you use a halt indicator (H1 through H9) or a message identification code on a KEY or SET operation. You must assign message identification codes 0001 through 0109 to specific kinds of messages in the message member:

<table>
<thead>
<tr>
<th>Message Identification Code</th>
<th>Kind of Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-0099</td>
<td>Message displayed as coded in columns 31 and 32 of a KEY or SET operation</td>
</tr>
<tr>
<td>0100</td>
<td>Message displayed at the end of a program cycle after all halt indicators are processed</td>
</tr>
<tr>
<td>0101-0109</td>
<td>Message displayed at the end of a program cycle in which a halt indicator (H1 through H9) occurs (0101 through 0109 correspond to H1 through H9)</td>
</tr>
</tbody>
</table>

Message text that will be displayed must be in an object message member. The message member must be coded in the OCL MEMBER USER1 statement, and the RPG program must use a KEY or SET operation or a halt indicator. The user message must have the prefix ‘USR’. (For information about the OCL MEMBER USER1 statement, see MEMBER Statement in the System Reference for the System/36 Environment.)

For each message 0101 through 0109 (corresponding to halt indicators H1 through H9), you can add a second-level message containing up to 225 characters. The second-level message must have the same message identification code as the first, and the second-level message member must be coded in the OCL MEMBER USER2 statement. After halt indicators H1 through H9 turn on, the program does all calculations and detail output operations for the record before processing ends and a message is displayed. If the halt indicators turn on during the processing of the last record in a file, the program does not stop processing but continues to completion.

Figure 103 on page 304 shows how to code the calculation specifications required to display a message.

The messages that are displayed as a result of the calculation specifications shown in Figure 103 on page 304 depend on if you coded the OCL MEMBER statement before running the program.

If the OCL program is:

```
// LOAD USER
// RUN
```

then the messages are displayed from the system message member. 01-Message indicator is the message displayed as the prompt for the KEY operation. If the user at the display station types HALT, halt indicator H1 turns on and the computer displays message RPG9101 RPG II indicator H1 is on. If the user enters option 0 in...
response to that message, the computer displays message RPG9100 All halt indicators have been displayed.

If the OCL program includes the MEMBER statement:

```
// LOAD USER
// MEMBER USER1-MESG1
// RUN
```

then the displayed messages come from your message member MESG1, which is coded on the MEMBER statement. The prompt *Type HALT to end the program is the text* displayed for the KEY01 operation. This prompt is the contents of your message 0001 in MESG1. Later, when the user at the display station types the literal HALT, the message 0101, HALT has been entered with a key operation, is displayed. If the second message, 0100, has not been loaded into your message member, it cannot be displayed. Instead, the message *Message not found in specified message member* is displayed.

Use the SEU or the $MAINT utility to load the source member (MESG1 for this example) into a library. The message object member MESG1 must exist before you run the program. For information on creating the message source member, see *SEU User’s Guide and Reference*. For information on creating a message load member, see the *System Reference for the System/36 Environment*.

```
 1       2       3       4       5       6       7
CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments++++++
C          KEY01    FLD  4
C          FLD    COMP 'HALT'    H1
C*
```

*Figure 103. Calculation Specifications Required to Display a Message*

**Calculation Specifications for a SET Operation**

The SET operation allows any or a set combination of the following to occur for a KEYBORD file:

- Pressing of function keys
- Displaying the field, literal, or array or table element coded in factor 1
- Displaying messages from your message member. The message that is displayed is determined by the message identification code in columns 31 and 32 of the SET operation.
- Deleting the buffer for a CONSOLE file if ERASE is coded in the result field of the SET operation.
To use the SET operation, code entries in the unshaded columns of the calculation specifications as shown below:

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Indicators</th>
<th>Control Level (L1 through L9)</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns 7 and 8 can contain a control-level indicator (L1 through L9). However, leave these columns blank if the SET operation is not part of a subroutine or if it is processed only at detail time.

Columns 9 through 17 can contain a conditioning indicator. However, leave these columns blank if the SET operation is processed on every program cycle.

Columns 18 through 27 (factor 1) can contain the constant, literal, field name, or table or array element displayed.

Columns 28 through 30 must contain the operation code SET.

Columns 31 and 32 can contain the message identification code (01 through 99) corresponding to the message displayed. The message length is truncated to the record length. This message must be in your message member. (For information about how to create a message member, see $MGBLD Utility Program in the System Reference for the System/36 Environment.) The message prompts the user at the display station to do a SET operation. An entry is required in columns 31 and 32 when columns 18 through 27 (factor 1) are blank and columns 54 through 59 contain a function key. However, if you code an entry in factor 1 and a message identification code in columns 31 and 32, the message identification codes are ignored. If you do not code the OCL MEMBER statement that specifies your message member before you run the program, or if columns 31 and 32 contain a message identification code that does not correspond to a message in your message member, the prompt nn-MESSAGE INDICATOR is displayed, where nn is the contents of columns 31 and 32. If both factor 1 and a message identification code are specified, the message identification code is ignored.

Columns 33 through 42 (factor 2) must contain the name of the CONSOLE file if ERASE is coded in columns 43 through 48 (the result field). For all other SET operations, leave columns 33 through 42 blank.

Columns 43 through 48 (the result field) must contain ERASE if the name of a CONSOLE file is coded in columns 33 through 42 (factor 2). For all other SET operations, leave columns 43 through 48 blank.

Columns 49 through 53 must be blank.

Columns 54 through 59 can contain one, two, or three function keys (KA through KN, or KP through KY) that the user at the display station can press when the program is at this specification line. If only one or two function keys are used, you can code them in any of the three sets of columns. When the user at the display station presses a function key coded in these columns, that function-key indicator
USING A KEYBORD FILE

turns on and stays on until it is used again in a SET operation or until it is turned off by the SETOF operation. If the user at the display station presses a function key other than those coded in columns 54 through 59 of a SET operation, the program stops. Several lines can be displayed before the program stops for input if you stack SET operations with factor 1 or a message identification code and no function key entries (see Figure 104). The program does not stop until a function key is pressed or a KEY operation occurs.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
C*
C* Stacking SET operations allows several prompt lines to appear as one C* prompt before input is required.
C*
C            LINE1      SET
C            LINE2      SET
C            LINE3      SET
C            LINE4      SET
C*
C* System halts when command-key indicators are specified.
C*
C            LINE5      SET            KAKBKC

Figure 104. Using SET Operations to Display a Prompt with More Than One Line
Figure 105 is a summary of calculation specifications for SET operations.

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

CC1N01N02N03Factor1+++OpCodeFactor2+++ResultLenDHHiLoEqComments+++++++*

C*
C* Display the contents of the first element of the array PROMPT.
C*
C PROPPMT,1 SET
C*
C* Allow function keys to be pressed. Prompt the operation by displaying
C* the contents of the field SELECT.
C*
C SELECT SET KAKB
C*
C* Display message 0013 from the user message member.
C*
C SET13
C*
C* ERASE or blank the existing CONSOLE buffer of the specified
C* CONSOLE file.
C*
C SET CONSFILE ERASE
C*
C* Allow function key 1, 2, or 3 to be pressed. Prompt the
C* operation by MIC 0023.
C*
C SET23 KAKBKCC

Figure 105. Summary of Calculation Specifications for SET Operations
Using a Keyboard File

Figure 106 shows possible combinations of uses for the SET operation.

\*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
C*
C* Displays contents of FIELDA. FIELDA is specified in factor 1 and
C* overrides message 0017.
C*
C FIELDA SET17
C*
C* Displays contents of FIELDA and allows function keys 3, 6, and 8
C* to be pressed. FIELDA in factor 1 overrides message 0026.
C*
C FIELDA SET26
C CKFKH

Figure 106. Possible Combinations of Uses for SET Operations

Allowing Function Keys To Be Pressed

The SET operation allows you to specify the function keys that the user can press
when the program is at a certain specification line. When the user presses a func-
tion key, the corresponding function-key indicator can be used to condition calcu-
lation or output operations that follow. Function-key indicators remain on until they
are used again in a SET operation or until they are turned off by the SETOF opera-
tion. The SET operation can be specified with function keys on factor 1.

When the program is at a certain specification line, you can allow the user to press
from one to three function keys. The keyboard keys may vary, depending on the
type of keyboard you have. After all function-key responses are entered, the user
presses an entry key.

There are 24 function keys. Each one corresponds to a separate function-key
indicator:

<table>
<thead>
<tr>
<th>Function Key</th>
<th>Function-Key Indicator</th>
<th>Function Key</th>
<th>Function-Key Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KA</td>
<td>13</td>
<td>KM</td>
</tr>
<tr>
<td>2</td>
<td>KB</td>
<td>14</td>
<td>KN</td>
</tr>
<tr>
<td>3</td>
<td>KC</td>
<td>15</td>
<td>KP</td>
</tr>
<tr>
<td>4</td>
<td>KD</td>
<td>16</td>
<td>KQ</td>
</tr>
<tr>
<td>5</td>
<td>KE</td>
<td>17</td>
<td>KR</td>
</tr>
<tr>
<td>6</td>
<td>KF</td>
<td>18</td>
<td>KS</td>
</tr>
<tr>
<td>7</td>
<td>KG</td>
<td>19</td>
<td>KT</td>
</tr>
<tr>
<td>8</td>
<td>KH</td>
<td>20</td>
<td>KU</td>
</tr>
<tr>
<td>9</td>
<td>KI</td>
<td>21</td>
<td>KV</td>
</tr>
</tbody>
</table>
If the user presses the wrong function key, the key can be reset and the command re-entered, as long as the user has not pressed the entry key. The user can reset all the function keys on a 5250 terminal by pressing the Cmd key and then by pressing the Character backspace (Clear) key while holding down the Shift key. The user can reset all the function keys on a 3180 terminal by pressing the ALT and CLEAR keys at the same time. The user can then retype the correct keys. If the user presses a function key that is not specified in the SET operation, error message RPG9049 The function key pressed is not defined is displayed. For more information about your keyboard see New User’s Guide.

If no function keys are pressed, the user responds to the SET operation by pressing only an entry key. This action turns off the function-key indicators. Coding your program to allow this response is not recommended because the user could do this accidentally. For example, the user could forget to press the Cmd key before pressing a number key and an entry key. This action turns off the function-key indicator that the user actually wanted to use.

Using the SET and KEY Operations Together

Normally, the user must press an entry key after doing each KEY operation or after pressing function keys coded in a SET operation. However, it is possible to combine these operations so that the user can press function keys (coded in columns 54 through 59 of a SET operation), type a field (specified in a KEY operation), and press an entry key only once (see Figure 107 on page 310), eliminating the need to press an entry key after SET.

This combination is possible only if:

- The SET operation immediately precedes the KEY operation
- The SET and KEY operations are conditioned by the same indicators (columns 7 through 17), coded in the same order
- The SET and KEY operations use the same message identification codes in columns 31 and 32. These columns can be blank in both operations if factor 1 is used to display messages
- Factor 1 for the SET and KEY operations can be the same, different, or missing from one operation.

If factor 1 is coded for both the SET and KEY operations, the contents of both factor 1s are displayed.

If the data field is numeric, the user must first press the specified function key, type the field, and then press the Field Exit, Field+, or Field- key. The Enter key cannot be used as an entry key for a numeric field.

### Table 5 (Page 2 of 2). Function Keys and Their Corresponding Function-Key Indicators

<table>
<thead>
<tr>
<th>Function Key</th>
<th>Function-Key Indicator</th>
<th>Function Key</th>
<th>Function-Key Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>KJ</td>
<td>22</td>
<td>KW</td>
</tr>
<tr>
<td>11</td>
<td>KK</td>
<td>23</td>
<td>KX</td>
</tr>
<tr>
<td>12</td>
<td>KL</td>
<td>24</td>
<td>KY</td>
</tr>
</tbody>
</table>
Using a CRT File

The CRT (cathode ray tube), or display screen, is designed to display messages and instructions for the user at the display station and to display that user's responses. You should not use the display screen like the printer as a major output device because data moves on and off the screen too fast.

To use a CRT file, you must code both file description and output specifications.

File Description Specifications

Code entries in the unshaded columns of the file description specifications as shown below:

Columns 7 through 14 must contain the name of the CRT file.

Column 15 must contain O to indicate that the file is an output file.
Columns 20 through 23, the block length, must equal the record length coded in columns 24 through 27 or be blank.

Columns 24 through 27 must contain the length of the largest record in the file. The maximum length is 79.

Columns 40 through 42 must contain CRT.

Columns 71 and 72 can contain an external indicator (U1 through U8).

Output Specifications

Because a CRT file is an output file, entries are also required on the output specifications. Code the unshaded columns on the output specifications as shown below:

File- and Record-Identification Entries

Columns 7 through 14 must contain the name of the file.

Column 15 must contain H (heading), D (detail), T (total), or E (exception) to indicate the type of record written.

Column 17 can contain a number from 0 to 3 to indicate how many lines to leave blank before writing the current line.

Column 18 can contain a number from 0 to 3 to indicate how many lines to leave blank after writing the current line. If the CRT file has a record length (columns 24 through 27 of the file description specifications) of 40 or less, columns 17 and 18 of the output specifications cannot both contain 3. Data moves onto the screen from bottom to top. Therefore, if you code an entry in column 18 for the bottom line of a full screen, the top line moves off the screen.

Columns 19 and 20 can contain 01 or blanks. Either entry tells the computer to clear the display before writing a record. If you code an entry other than 01 in columns 19 and 20, the computer assumes that the entry is 01 and erases the display.

Columns 23 through 31 can contain output conditioning indicators.

Columns 32 through 37 can contain an EXCPT name if column 15 contains E.
Field-Description Entries
Columns 23 through 31 can contain indicators.

Columns 32 through 37 can contain the names of the individual fields in the record.

Column 38 can contain an edit code.

Column 39 can contain B to indicate that the field is reset to blank or zero.

Columns 40 through 43 can contain the end position of each field in the output record.

Columns 45 through 70 can contain an edit word or literal constant.

Displaying Data
Data is displayed at the normal output times (total and detail) or at calculation time for exception output. (See “Column 15 (Type)” on page 607, for information on exception output.) Any alphameric character can be displayed. If the record length is 40 or less, up to 40 characters can be written across the width of the screen, and a maximum of six such lines can appear at one time. The display is centered both vertically and horizontally. If the record length is more than 40, up to 79 characters can be written across the width of the screen, and a maximum of 24 lines can appear at one time.
A BSCA file is one way to send and receive data between your AS/400 system and another system. The letters BSCA stand for **binary synchronous communications adapter**. The **adapter** is part of the hardware. BSC is part of the Operating System/400 (OS/400). BSC allows you to communicate binary data (data represented as 0s and 1s) that is synchronized (the sending and receiving of data is controlled by timing signals).

BSC has several limitations compared to the Intersystem Communications Function (ICF), which uses a WORKSTN file to communicate with other systems:

- A program using BSCA files is limited to using one communication line.
- Block length is restricted to 4075 bytes.
- BSC has no timed receive capability.
- The BSCA file interface does not provide the ability to start programs automatically on the remote system without operator intervention.

To avoid these limitations, you may want to use ICF.

You must define and activate the communications configuration prior to using the BSCA file. You create a line description with the CL command Create Line Description (BSC) (CRTLINBSC), a controller description with Create Controller Description (BSC) (CRTCTLBSC), and a device description with Create Device Description (BSC) (CRTDEVBSC). See "Translating T-SPEC Entries To AS/400 Configuration" on page 320 for more information on creating the communications configuration specific to your application. Refer to the *Communications: Operating System/400* Communication Configuration Reference for additional information about these commands and their parameters.

You activate the configuration with the CL command Vary Configuration (VRYCFG). Refer to the *OS/400* Communication Configuration Reference for additional information about this command.

You must also supply a // SESSION OCL statement between the // LOAD and // RUN statements that call this program. This provides the link between the RPG II program and the communications configuration. Refer to page 323 and to the *Concepts and Programmer’s Guide for the System/36 Environment and System Reference for the System/36 Environment* for more information about // SESSION OCL for BSCA files.

Applications running in the System/36 environment that use the BSCA file automatically use a system-supplied ICF file called QCRGBSCA in QSSP. This file should not be deleted or changed. These applications also use support provided by the BSC Equivalence Link. Therefore, you may need to refer to the *Communications: BSC Equivalence Link Programmer’s Guide, SC41-9593* for additional information on using BSC support.
Defining a BSCA File

To define a BSCA file, you must make entries on the file description and telecommunications specifications. In addition, the control specification must contain a blank or I in column 37 to indicate that the program does not recognize an inquiry request. A BSC program must not be interrupted, because an interruption might cause the remote system to stop communications.

File Description Specifications

Code entries in the unshaded columns of the file description specifications are shown below:

Columns 7 through 14 must contain the name of the BSCA file. The same name must be used in columns 7 through 14 of the telecommunications specification.

Column 15 must contain I or O to indicate that this file is an input (receive) file or an output (send) file.

Column 16 must contain P (primary), S (secondary), T (table), or D (demand) if column 15 contains I. If column 15 contains O, column 16 must be blank. D (demand) is the required entry when you use the file for interspersed sending and receiving of data. D should also be used for receiving programs that do not process BSCA files immediately. For example, if the BSCA file is defined as a secondary file, the communications line opens as soon as the program begins. This may cause your wait time to be used up before you are ready to process the BSCA file. If the BSCA file is defined D (demand), the line opens when the program is ready to receive the first record from the BSCA file.

Column 17 must contain E or blank if column 16 contains P, S, or T. Column 17 must also be blank if column 16 is blank or contains D. E must be used if the program end is determined by the end of file on the input (receive) file. The BSCA file might be the only file with E in column 17. If any other input file contains E in column 17, all BSCA input files should contain E in column 17. The E is not necessary for the BSCA files, but is necessary for the system on the other end of the communication. The E closes the BSCA input file, but the other input file is unaware of what has happened. When E is specified for all the BSCA files, all systems end the program successfully.

Column 18 must be blank if column 16 contains D or blank. Column 18 must contain A, D, or blank if column 16 contains P, S, or T. The program checks the record order. A indicates that the records in the file are to be in ascending sequence. D indicates that the records are to be in descending sequence. Blank indicates that the program does not check the record sequence.
Column 19 must contain F or blank to indicate that all records in the file have the same length.

Columns 20 through 23 must contain the block length of data processed by BSC. The block length must be a multiple of the record length. The maximum block length is 4075 positions. This value is syntax-checked for compatibility with System/36, but is ignored on the AS/400 system. The block length specified on the line description and the device description is used. For the line description, it is specified on the MAXBUFFER parameter of the CRTLINBSC command. For the device description, it is specified on the BLKLEN parameter of the CRTDEVBSC command or the BLKL parameter of the // SESSION OCL statement.

**Note:** On System/36 RPG II Telecommunications, the blocksize and record length parameter could be changed in the program at initialization time. On the AS/400 system, dynamic change of record length and block size within the program is not supported. If you want to change record or block length, you must end your program, then change the record or block length with a // SESSION OCL or a Change Device Description (BSC) (CHGDEVBSC) CL command.

Columns 24 through 27 must contain the record length. The record length specified on the device description, RCDLEN parameter of the CRTDEVBSC command, or the RECL parameter of the // SESSION OCL statement must be as large as the largest record length used by the program. Records with a length of zero are ignored by RPG unless the other system is in 3740 mode. The zero length record is considered a file separator when in 3740 mode. All remaining space in records with lengths that are greater than zero but less than the specified record length are filled with blanks.

**Note:** On System/36 RPG II Telecommunications, when you specified a smaller record length than what was actually received, the System/36 support gave the application a partial record (based on the application length) on each read operation until the complete record had been processed. On the AS/400 system, a permanent error and message (CPF 4768 appears in the job log) results if you specify a shorter record length than is actually received.

Column 32 can contain any number 1 through 9 or a blank. A number indicates that the program uses two input/output areas. A blank indicates that the program uses only one input/output area. While this value is syntax-checked, for compatibility with System/36, it is ignored on OS/400. On OS/400, double buffering is always used.

Columns 40 through 43 must contain the device name BSCA.

Columns 71 and 72 can contain an external indicator, U1 through U8.
Telecommunications Specifications

Code entries in the telecommunications specifications are shown below:

<table>
<thead>
<tr>
<th>Line</th>
<th>Form Type</th>
<th>Filename</th>
<th>P/S/M Configuration</th>
<th>T/R Type of Station</th>
<th>T/ Type of Control</th>
<th>U/E Type of Code</th>
<th>Y/N Transparency</th>
<th>M/E/S/A/B</th>
<th>S/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many of the specifications below are syntax-checked, but ignored by the OS/400 operating system. The correct placement of data will be given with each field.

Columns 7 through 14 must contain the name of the BSCA file. This must be the same name coded on the file description specifications for the BSCA file.

Column 15 must contain P, M, S, or blank. P and blank indicate that the file is a point-to-point nonswitched network. M indicates that this is a multipoint network where the control station selects the tributary station through polling or addressing. The AS/400 system cannot be the control station. (If this column contains M, column 17 must contain T.) S indicates that the file is a point-to-point switched network. The option chosen in column 15 must correspond to the network connection parameter (CNN) on the line description used by this program.

Column 16 must contain T or R. T indicates that this station sends data from this BSCA file. T must also be defined on the output specifications. R indicates that this station receives data in the BSCA file. The BSCA file must also be defined on the input specifications. The column-16 entry is independent of the entry in column 20.

Column 17 must contain T or blank. T indicates that this is a tributary station on a multipoint network. (Column 17 must contain T if column 15 contains M.) A blank indicates that polling is not used. The AS/400 system cannot be the control station. This value is syntax-checked for compatibility with System/36, but is ignored on the AS/400 system. The multipoint tributary option must be specified on the line description, the controller description, and on the device description. This is defined through the CNN parameter of the CRTLINBSC and CRTCTLBSC commands, and on the the TYPE parameter of the CRTDEVBSC command.

Column 18 must contain A, U, E, or blank. A and U indicate that ASCII transmission control characters are used and that each station must provide file translation when it is required. E or blank indicates that EBCDIC transmission control characters are used. If the ASCII transmission code is required, the CODE parameter in the line description for this program must be specified as *ASCII. For more information about translating EBCDIC characters to ASCII, see “ASCII-EBCDIC Character Translation” on page 319.

Column 19 must contain Y, N, or blank. Y indicates that EBCDIC transparency is used. and that the data being transferred can be packed-decimal numeric or alphanumeric and can contain transmission control characters. Y or blank in column 19 indicates that EBCDIC transparency is not used; that is, the data being transferred is zoned-decimal numeric or alphanumeric and does not contain transmission characters.
Transparency must also be indicated on the device description using the TRNSPY parameter on the CRTDEVBSC command or the TRANSP parameter of the // SESSION OCL statement.

Column 20 must contain M, A, B, or blank. M indicates that the program user makes the connection by dialing the number manually. A indicates that the program uses autoanswer. B indicates that the program uses manual answer. Blank indicates that this is not a switched network. The entry in column 20 is independent of the entry in column 16. The AUTOANS, AUTODIAL, and SWTCNN parameters on the CRTLINBSC must correspond to the option chosen in column 20. The INLCNN and CNNNBR parameters of the CRTCTLBSC must also correspond to the option chosen in column 20.

Column 32 must contain E, S, or blank. E indicates that this station’s identification is the entry in columns 33 through 39. S indicates that this station’s identification is at the position specified by the symbolic name in columns 33 through 39. Blank indicates that this station uses no identification. This station’s identification must also be specified on the LCLID parameter of the CRTCTLBSC command. Symbolic names for local identification are not supported on the AS/400 system.

Columns 33 through 39 must contain this station’s actual identification if column 32 contains E. If column 32 contains S, the symbolic name of the location of this station’s identification must be in column 33 through 39.

Column 40 must contain E, S, or blank. E indicates that the remote station’s identification is the entry in columns 41 through 47. S indicates that the remote station’s identification is at the position specified by the symbolic name in columns 41 through 47. Blank indicates that the remote station uses no identification. The remote station’s identification must also be specified on the RMTID parameter of the CRTCTLBSC command. Symbolic names for remote identification are not supported on the AS/400 system.

Columns 41 through 47 must contain the remote station’s actual identification if column 40 contains E. If column 40 contains S, the symbolic name of the location of the remote station’s identification must be in columns 41 through 47.

Column 52 must contain I or blank. I indicates that intermediate block checking is used. Blank indicates that intermediate block checking is not used. Intermediate text blocking must also be specified on the device description. To specify this, use the BLOCK parameter on the CRTDEVBSC command or the ITB parameter of the // SESSION OCL statement. This blocking option cannot be specified if transparency is also specified.

Columns 53 and 54 contain a permanent-error indicator (01 through 99 L1, through L9, LR, or H1 through H9) or blanks.

Columns 55 through 57 contain the number of seconds (1 through 999) that BSC waits with no messages sent or received before a permanent error occurs. If you leave these columns blank, BSC waits 180 seconds before a permanent error occurs. On OS/400 the line retry limits are set by a count value in the TMTRTY and RCVRTY parameters on the CRTLINBSC command.

Columns 58 and 59 contain a record-available indicator (01 through 99, L1 through L9, LR, or H1 through H9) or blanks. The record-available indicator turns on when-
ever a reverse interrupt is received from the other system, indicating a request to send.

Column 60 must contain L or blank. L indicates that this BSCA file is processed only after all other input files are processed. Blank indicates that this BSC file may not be the last input file processed. The entry in column 60 does not affect demand files.

Columns 61 and 62 must contain the station’s polling identification if it is part of a multipoint network and if the BSC file is an output file. Otherwise, leave these columns blank. Polling identification must also be specified in the STNADR parameter on the CRTLINBSC command and the LOCADR parameter on the CRTDEVBSC command.

Columns 63 and 64 must contain the station’s addressing identification if it is part of a multipoint network and if the BSCA file is an input file. Otherwise, you can leave these columns blank as the AS/400 system ignores this parameter.

**Note:** On the AS/400 system, the parameters from columns 15, 18, 19, 20, 32, 40, 52 and 55 through 57 cannot be changed at program initialization as they could be on the System/36. On the AS/400 system, the specified parameters are part of the configuration for BSCEL and cannot be changed by the user’s program. On the System/36, no configuration was used other than the hardware type specified by SETCOMM. The parameters of ALTERCOM could effectively change the configurations between job steps. On the AS/400 system, there is only equivalence on parameters supported on the // SESSION OCL statement and on parameters that are changeable on the configuration CL commands.

---

**Programming Considerations**

**First RPG Program Cycle**

During the first RPG program cycle, all primary and secondary input files are opened. The program reads one record from each primary and secondary input file before it processes any input file. You might want to delay the first-time logic for your BSCA input files by designating each BSCA input file as a demand file. You designate them by entering D in column 16 of the file description specifications. One or more BSCA input files can also be designated as the last file by entering L in column 60 of the telecommunications specifications. If 3740 multiple-file support is being used, all secondary input files should have the L in column 60. Remember that an entire BSCA input file, until EOF occurs, must be received before another BSCA input file can be received or a BSCA output file can be transmitted. If the primary file has more than one record, a permanent error will result, because a secondary file is read before EOF is reached on the primary file.

**Removing Strings of Embedded Blanks**

To use the communications line more efficiently, OS/400 BSC allows RPG users to send and receive data with strings of two or more embedded blanks removed. Removing strings of embedded blanks is called **compressing** the data. The same method is used by the IBM 3780 Data Communications Terminal.
For output files, data is moved from the logical buffer to the BSC input/output buffer with blanks removed and compression control characters inserted. The receiving station automatically inserts the same number of blanks where they were removed.

For input files, the procedure is reversed. The OS/400 recognizes the compression control characters, inserts the blanks removed by the remote station, and moves the record from the BSC input/output buffer to the logical buffer.

To remove a string of embedded blanks, use the DTACPR parameter on the device description.

Blanks cannot be inserted or removed if you use EBCDIC transparency (Y in column 19 of the telecommunications specifications) or intermediate block checking (I in column 52).

**Removing Trailing Blanks**

OS/400 BSC also allows you to send and receive data with trailing blanks removed. Removing trailing blanks is called **truncating** the data.

For output files, data is moved from the logical buffer to the BSC input/output buffer with all trailing blanks removed. The receiving station automatically inserts blanks after the data portion up to the record length specified.

To remove trailing blanks, use the TRUNC parameter on the device description.

Trailing blanks cannot be removed when you use intermediate block checking (ITB) or blocked records with no separator character.

When you add or remove blanks with blocked records, the number of records per block varies depending on the number of blanks in each record.

**ASCII-EBCDIC Character Translation**

The AS/400 system will perform the translation from EBCDIC to ASCII based on the CODE parameter on the CRTLINBSC command. All you need to do is specify CODE(*ASCII) on the CRTLINBSC command and the AS/400 system will perform the translation for you. You must also remove the file translation being performed on the BSCA files in the program that translates between EBCDIC and ASCII, so that only EBCDIC is passed to the AS/400 system. For this reason, a program written or modified for the AS/400 system ASCII-EBCDIC Character Translation is not compatible with System/36 support. You cannot interchange new or modified ASCII-EBCDIC AS/400 system programs with those of the System/36.

If the file translation you are doing differs from the standard translation, then you may still use file translation. To get the desired results you translate EBCDIC to EBCDIC in order to get a ASCII character for a particular hex character. To do this you translate EBCDIC characters to other EBCDIC characters that will result in the expected ASCII character produced by BSCEL. For example '27'x is normally translated to ASCII as '1B'x and 'F0'x is to '30'x, but suppose you want '27'x to translate to '30'x. You can translate the '27'x to 'F0'x, which will be translated to '30'x by BSCEL. This will yield the desired ASCII character for EBCDIC '27'x.
Control Breaks

Take care when sending data during total time in any RPG program that both sends and receives. Data might not be available for output even though it is read because of the sequence of total and detail operations in the RPG program cycle. For example, when a record is read that causes a control break, total output will be performed before the record just read is moved into the input area and is available for processing. For more information see “RPG Program Cycle” on page 6.

Reclaim Resources

Do not use the RCLRSC CL command during the execution of programs using BSCA files. If you do, the BSCA files will be closed and the current communications session will be terminated.

RPG Diagnostics

See “Using, Displaying, and Printing Messages” on page 61 for a discussion of RPG diagnostics.

Configuring Your System for BSC

Configuring means defining to the system the devices, controllers, and lines installed on the system.

You must define and activate the communications configuration prior to using the BSCA file. You create the line description with the CL command CRTLINBSC, the controller description with the CL command CRTCTLBSC, and the device description with the CL command CRTDEVBSC. The configuration is activated using the CL command VRYCFG.

Translating T-SPEC Entries To AS/400 Configuration

Perform the following steps to execute an RPG II program that uses BSCA file(s) and telecommunications specifications:

1. Create a line description (using the CRTLINBSC command) and specify the following parameters to match values in the RPG II program:
   - MAXBUFFER - Should be set to a value of the largest block length (column 20-23 of file description specification).
   - APPTYPE - Set to *PGM.
   - CNN - From column 15 of the telecommunication specification, determine which of the following should be specified:
     - If column 15 is P or blank, set CNN to *NONSWTPP.
     - If column 15 is M, set CNN to *MPTRIB.
     - If column 15 is S, set CNN to *SWTPP (be sure to follow step 5 on page 323).
   - STNADR - From column 61-62 of the telecommunication specification, enter one of the polling characters in hexadecimal. Example: If BB was specified in column 61-62, enter C2 as the STNADR (where C2 is the hexadecimal value of the character B).
   - SWTCNN - From column 20 of the telecommunication specification, determine which of the following should be specified:
     - If column 20 is M, set SWTCNN to *DIAL.
If column 20 is A or B, set SWTCNN to *ANS.

- **AUTOANS** - From column 20 of the telecommunication specification, determine which of the following should be specified:
  - If column 20 is A, set AUTOANS to *YES.
  - If column 20 is something other than A, set AUTOANS to *NO.

- **CODE** - From column 18 of the telecommunication specification, following:
  - If column 18 is A or U, set CODE to *ASCII.
  - If column 18 is E or blank, set CODE to *EBCDIC or ignore this parameter because it will default to *EBCDIC.

- **TMTRTY** and **RCVRTY** - From column 55-57 of the telecommunication specification, determine if additional retries are needed. In the OS/400 operating system, the retries are specified as a count instead of a time value as is specified on the System/36 Telecommunications Specifications coding form.

Specify any unique name for the LIND parameter. This will be the name of the line description. Refer to the OS/400 Communications Configuration Reference for information concerning determining the resource name to specify in the RSRCNAME parameter.

2. Create a controller description (using the CRTCTLBSC command) and specify the following parameters to match values in the RPG II program.

- **CNN** - Follow the procedure described above for CNN on the line description.

- **APPTYPE** - Set to *PGM.

- **INLCNN** - From column 20 of the telecommunication specification, determine which of the following should be specified:
  - If column 20 is M, set INLCNN to *DIAL.
  - If column 20 is A or B, set INLCNN to *ANS.

- **CNNNBR** - If column 20 of the telecommunication specification is M, specify the dial telephone number.

- **LCLID** - If column 15 of the telecommunication specification is S, from column 32 and 33-39, specify the following:
  - If column 32 is blank, set LCLID to *NOID.
  - If column 32 is E, use the value from column 33-39 as the value for LCLID, but convert it to hexadecimal. (See sample configurations on pages 329 and 335.)
  - If column 32 is S, a different programming method will need to be determined because symbolic local station ID capability is not supported.

- **RMTID** - If column 15 of the telecommunication specification is S, then from column 40 and 41-47 specify the following:
  - If column 40 is blank, set RMTID to *NOID
  - If column 40 is E, use the value from column 41-47 as the value for RMTID, but convert it to hexadecimal. (See sample configurations on pages 329 and 335.)
  - If column 40 is S, a different programming method will need to be determined since symbolic remote station ID capability is not supported.
An option that may help is that up to 64 remote IDs can be specified for RMTID, or a value of "ANY" can be specified which allows any remote ID to be accepted.

- **LINE** - If CNN is not "SWTPP", specify the name of the BSC line created in step 1.
- **SWTLINLST** - If CNN is "SWTPP" then specify the name(s) of the BSC

Specify any unique name for the CTLD parameter. This will be the name of the controller description.

3. Create a device description (using the CRTDEVBSC command) and specify the following parameters to match values in the RPG II program:

- **LOCADR** - If column 15 of the telecommunication specification is M, set the value of LOCADR to 2D. If column 15 is not M, set LOCADR to 00.
- **RMTLOCNAME** - This name must match the LOCATION name specified on the // SESSION OCL statement.
- **APPTYPE** - Specify "RPGT". If "BSCEL" is used, a "line monitor" function will respond to incoming line bids when an application program is not started. This may cause an undesired transmission end sequence to be sent.
- **BLOCK** - From column 52 of the telecommunication specification, determine which of the following should be specified:
  - If column 52 is I, set BLOCK to "ITB".
  - If column 52 is blank, but the block length (column 20-23 of file description specification) is a multiple of the record length (column 24-27 of file description specification), set BLOCK to "NOSEP".
- **RMTBSCEL** - Set to "NO".
- **RCDLEN** - From column 24-27 of the file description specification, set the maximum record length to be used.
- **BLKLEN** - From column 20-23 of the file description specification, set the maximum block length to be used.
- **TRNSPY** - From column 19 of the telecommunication specification, determine which of the following should be specified:
  - If column 19 is N or blank, set TRNSPY to "NO".
  - If column 19 is Y, set TRNSPY to "YES".
- **GRPSEP** - If you are communicating with an IBM 3740 using multiple-file support, specify "DEV3740"; if you are not communicating with a 3740, specify "EOT".
- **CTL** - Specify the name of the controller created in step 2 on page 321.

Specify any unique name for the DEVD parameter. This will be the name of the device description.
Notes:

a. The above values (under numbers 1, 2, and 3) are always used from the line, controller, and device descriptions. Defaults are used if you do not explicitly set them. Therefore, the values specified in the RPG program will be ignored by the system.

b. You can use the CHGLINBSC, CHGCTLBSC, and CHGDEVBSC to alter the communications configuration.

c. Refer to the OS/400* Communications Configuration Reference for more information on communications configuration.

4. If the CNN parameter of the CRTLINBSC command was set to *SWTPP, use the CHGLINBSC command to update the SWTCTLLST parameter with the name of the controller description from step 2 on page 321.

5. Change the state of the line, controller, and device descriptions by varying on or off. Do this by using the VRYCFG command or the WRKCFGSTS command.

6. Code a // SESSION OCL statement between the //LOAD and //RUN statements to link together the program and the configuration descriptions. Use the same BSCEL parameters as documented in the Concepts and Programmer’s Guide for the System/36 Environment for the // SESSION OCL statement. For the SYMID parameter, you must always specify ‘9#’. RPG II will look for this device when the BSCA file(s) are opened, and a run-time error will occur if it is not defined. For this reason, you should not use ‘9#’ as the SYMID parameter for ICF devices if you are planning to use both BSCA files and ICF in one RPG program. See Concepts and Programmer’s Guide for the System/36 Environment for more information about BSCEL terminology, configuration, and programming consideration.

Note: You can use the // SESSION OCL with the RPG II BSCA support to temporarily override configuration information. It is not recommended that you use the OVRICFDEVE CL to override the configuration information because unexpected results will occur.

Descriptions of BSC Functions

This section highlights some of the functions that the AS/400 system can perform as part of a data communications network. The sample RPG programs later in this chapter illustrate these functions.

Receive-Only Function

The receive-only function allows you to receive input data from another station. The file can be a primary, secondary, table, or demand file. The records can be blocked.

Code a receive-only file as an input file on the file description specifications (I in column 15) and as a receive file on the telecommunications specifications (R in column 16).
DESCRIPTIONS OF BSC FUNCTIONS

Send-Only Function

The send-only function allows you to send BSC data to a remote location.

Code a send-only file as an output file on the file description specifications (O in column 15) and as a transmit file on the telecommunications specifications (T in column 16).

Send-and-Receive Function

To both send and receive data, use two files. Code one as an output file on the file description specifications and as a transmit file on the telecommunications specifications. Code the other as an input file on the file description specifications and as a receive file on the telecommunications specifications.

In any BSC program that sends and receives, column 15 and columns 17 through 47 of the telecommunications specifications must be identical for the two files.

BSC programs that send and receive can be written in any of these three ways:

- Send a file, and then receive a file
- Receive a file, and then send a file
- Send records interspersed with receive records.

Send a File, Then Receive a File

The receive file must not be defined as the primary input file on the file description specifications. If the receive file is a secondary file, column 60 of the telecommunications specifications must contain an L. The matching-fields and record-available indicators must not be specified for the BSCA file.

Receive a File, Then Send a File

The receive file can be defined as a primary, secondary, table, or demand file on the file description specifications. Columns 58 and 59 (record-available indicator) on the telecommunications specifications must be blank.

Send Records Interspersed with Receive Records

A program can send records interspersed with records it receives. Such a program sends records from one file and receives records in another; the two files might not be related. Unlike conversational programs, this kind of program might intersperse several records or several blocks of data at a time.

The receive file must be defined as a demand file on the file description specifications. The record-available indicator must be specified on the telecommunications specifications. The AS/400 system must begin by sending data, then suspend the transmit file to receive data from the other station. See Figure 110 on page 336 for an example of this type of program when the record-available indicator is set.

Once BSC begins to process the last record in the transmit file, the AS/400 system ignores the record-available indicator, whether or not the last record was actually sent. When BSC accepts the last record in the file for transmission, RPG completes last-record processing and begins to close the file.

As a result, if the next-to-last record or block of records intended for transmission prompts the other station to send data back to the AS/400 system, the request may be ignored. The request will always be ignored if it is prompted by the final record.
or block of records. This occurs because a reverse interrupt (RVI) can only be received on an output operation, so if no more output is being sent, any RVI sent by the remote program cannot be received.

You can avoid this problem by adding a special record to the end of the AS/400 system transmit file. This record signals that the AS/400 system went to the end of the job but cannot honor a request to receive, even though that request was just sent. The programmers from both systems must agree on the meaning of that special record.

**Systems That Use BSC**

You can use BSC for data communication between your AS/400 system and any of the following IBM systems and devices:

- Another AS/400 system with RPG telecommunications, or BSCEL
- System/38 using RPG III or COBOL
- System/36 with RPG telecommunications, or SSP-ICF BSCEL
- System/34 with RPG telecommunications, or SSP-ICF BSCEL
- System/32 with RPG or assembler
- System/3 with RPG, CCP, or MLMP
- System/7 with MSP/7
- Operating System or Disk Operating System Basic Telecommunications Access Method (OS, OS/VS, DOS/VS, or DOS BTAM)
- Customer Information Control System (CICS/DOS/VS or CICS/VS)
- Information Management System (IMS/VS)
- 3741 Model 2 Data Station or Model 4 Programmable Work Station
- 3747 Data Converter
- 5231 Data Collection Controller Model 2 (as a 3741 in transmit mode only)
- 3750 Switching System (world trade only)
- 5110 or 5120 (in 3741 mode)
- Series 1 (in System/3 mode)
- 5260 Point of Sale Terminal (in 3740 mode)
- 5280 Distributed Data System (in 3740 mode).

**Device-Dependent Considerations**

**IBM 3740 Data Entry System**

RPG data communications programming supports the IBM 3741 Model 2 Data Station, the IBM 3741 Model 4 Programmable Work Station, or the IBM 3747 Data Converter in communicate mode as a remote device by using the AS/400 system communications adapter.
Restrictions
The following restrictions apply when an AS/400 system communicates with a 3740 Data Entry System:

- A 3741 with an Expanded Communications Buffer feature (feature number 1680) has a maximum buffer size of 512 positions.
- A 3747 with the Blocking/Reformatting feature (feature number 1480) has a maximum buffer size of 8050 positions. AS/400 system RPG can only handle a maximum of 4075 positions.
- The Operator Identification Card Reader Feature (feature number 5450) and the Expanded Communications/Multipoint Data Link Control Feature (feature number 1685) on the 3741 are not supported by RPG II on the AS/400 system.
- You can communicate with the 3741 or the 3747 by using either single-file or multiple-file support. Single-file support allows one input file, one output file, or one of each. Multiple-file support allows more than one input file, more than one output file, or more than one of each.
- You can send and receive blocked records from a 3741 with the Expanded Communications Buffer feature or a 3747 with the Blocking/Reformatting feature.
- If you send blocked records to a 3741, you must specify a record separator of hexadecimal 1E. To do this specify BLOCK(*IRS) on the CRTDEVBSC command.
- RPG receives 3741 STATUS messages as data, and these must be handled by the user. Refer to IBM 3741 Data Station Reference Manual, GA21-9183, for details of the possible status messages and the format of the data that will be received.

Single-File Support
If you have single-file support when you communicate with the 3741, a maximum of two BSCA files are allowed (one input and one output) per RPG program. If you use two BSCA files, you must process the input file completely before you process the output file.

When you communicate with the 3747 Data Converter, only one BSCA file is allowed (either input or output).

Multiple-File Support
When you communicate with the 3741 or with 3741 emulators, multiple files can be received, sent, or sent and then received.

All input files from the 3740 must be received before the AS/400 system can begin sending files to the 3740.

When you communicate with the 3747, multiple files can be either received or sent.

When you communicate with the 5110 or 5120, multiple files can be received, sent, received and then sent, or sent and then received.

To specify multiple-file support, specify GRPSEP(*DEV3740) on the device description. You can use the CHGDEVBSC command to change this parameter. If you are using the OVRICFDEVE CL command to temporarily override configuration
information, do not change the group separator (GRPSEP), or unpredictable results will occur.

**Blocked Records**

You can send blocked records to a 3747 with the Blocking/Reformatting feature by using columns 20 through 23 of the file description specifications, which contain a multiple of the record length.

RPG input files can be primary, secondary, or demand files. The 3740 files must be processed one file at a time to the end of the file and in the order that the 3740 sends them.

Secondary files are processed in the order listed on the file description specifications in the source program.

Demand files are processed in the order determined by the logic of your calculation specifications.

Output files must be processed one file at a time. That is, all records for one file must be sent before the first record for the next file is sent. When you communicate with a 3741, if multiple files are received and then multiple files are sent in the same program, all input files must be processed before any output files are processed.

**RPG Specifications**

Use of the 3740 affects RPG file descriptions, telecommunications, and output specifications. Only the entries unique to the 3740 are described here.

**File Description Specifications**

**Columns 20-23 (Block Length):**

Maximum block length is 128 positions without the Expanded Communications Buffer feature on the 3741 or with the Blocking/Reformatting feature on the 3747.

If blocked records are sent to a 3741 with the Expanded Communications Buffer feature, the block length can be any multiple of the record length not exceeding 512 positions.

If blocked records are received from a 3741 with the Expanded Communications Buffer feature, the block length must be $N$ times the record length, where $N$ is the result (disregarding the remainder) of dividing 512 by the record length plus one.

For example, if the record length is 128:

- Record length plus one = 129
- 512 divided by 129 = 3 with a remainder of 125
- $N = 3$
- Block length = 3 times 128 = 384.

When you communicate with a 3747 using the Blocking/Reformatting feature, the block length depends on the use of the data at the 3747 and on the amount of storage available (feature number 7690, 7691, or 7692). Blocking on the 3747 can be identical to that on the 3741 with the Expanded Communications Buffer feature through the use of C3 control records. Blocking can also be handled in a manner similar to RPG blocking through use of the C3 control records. For an explanation of the C3 control records format, see *IBM 3747 Data Converter Reference Manual and Operator’s Guide*, GA21-9170.
Columns 24-27 (Record Length): The maximum record length when communicating with a 3741 is 128 positions. The 3747 maximum record length depends on the use of the data at the 3747 and on the Blocking/Reformatting feature being installed.

Telecommunications Specifications
The 3740 files require some restrictions to the telecommunications specifications. Only the columns affected are listed here:

Column 15 (Multipoint network): M must not be specified.

Column 17 (Tributary system on a multipoint network): T must not be specified.

Column 52 (ITB): I must not be specified.

Columns 61 through 74: Must be blank.

Output Specifications
Columns 17-22: Must be blank.

IBM 3750 (World Trade Only)
When the AS/400 system is connected to an IBM 3750 Switching System, the RPG data communications program must allow message exchanges between the two systems. The RPG program can be written for message exchanges related to the following 3750 functions:

- Recording calls
- Monitoring contact under control of the data processing system
- Inquiring to the data processing system with a recorded answer
- Collecting real-time data to the data processing system
- Transferring recorded data to the data processing system.

Communications between the AS/400 system and the 3750 are binary synchronous, point-to-point operations in transparent mode. Only EBCDIC can be used. The AS/400 system operates as a send and receive station.

When you receive the end-of-transmission character and the next step in your program is to receive more data, do another read operation to the same BSC input file.

Sample Programs
The following three sample programs are provided as examples of the various types of RPG BSC programs:

- The first example is a send-only program.
- The second is a AS/400 system-to-3740 program.
- The third is a send-interspersed-with-read program.
Send Only

Figure 108 on page 330 shows a program that reads a DISK file and then sends it. The week’s data has been sorted by name of salesman. The amount of each sale is written on the disk, and the total sales for each salesman is transmitted to the branch office. After all disk records containing sales information are read, the total of all sales is sent to the branch office.

AS/400 system Configuration Descriptions

1. CRTLINBSC
   - LIND(BSCLIN1)
   - RSRCNAME(LIN021)
   - CNN(*SWTPP)
   - SWTCNN(*ANS)
   - AUTOANS(*YES)
   - CODE(*EBCDIC)
   - TMTRTY(10)
   - RCVRTY(10)

2. CRTCTLBSC
   - CTLD(BSCCTL1)
   - CNN(*SWTPP)
   - APPTYPE(*PGM)
   - INLCNN(*ANS)
   - LCLID(D4E8C9C4)
   - RMTID(E8D6E4D9C9C4)
   - SWTLINLST(BSCLIN1)

3. CRTDEVBSC
   - DEVD(BSCDEV1)
   - LOCADR(00)
   - RMTLOCNAME(RMTLOC1)
   - APPTYPE(*RPGT)
   - BLOCK(*NOSEP)
   - RCDLEN(30)
   - BLKLEN(90)
   - TRNSPY(*YES)
   - GRPSEP(*EOT)
   - CTL(BSCCTL1)

4. CHGLINBSC LIND(BSCLIN1) SWTCTLLST(BSCCTL1)

5. // SESSION LOCATION-RMTLOC1,SYMID-9#

---

1. D4E8C9C4 is hexadecimal for MYID. The character format is used on System/36 for this field, but the hexadecimal format is used on the AS/400 system.

2. E8D6E4D9C9C4 is hexadecimal for YOURID. The character format is used on System/36 for this field, but the hexadecimal format is used on the AS/400 system.
### RPG File Description Specifications

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-14</td>
<td>WKLYSMRY is a BSCA file.</td>
</tr>
<tr>
<td>15</td>
<td>Because WKLYSMRY is transmitted, it is an output file.</td>
</tr>
<tr>
<td>19</td>
<td>BSCA files always have a fixed-length format.</td>
</tr>
<tr>
<td>20-27</td>
<td>Records are blocked.</td>
</tr>
<tr>
<td>32</td>
<td>Dual I/O areas are used.</td>
</tr>
<tr>
<td>40-46</td>
<td>BSC is the device name.</td>
</tr>
</tbody>
</table>

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
H...OLExeD..CDYI....S............I..1.F.H............T...................

**Figure 108 (Part 1 of 3). Send-Only Program**

### RPG Telecommunications Specifications

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-14</td>
<td>WKLYSMRY is the BSCA file for this program.</td>
</tr>
<tr>
<td>15-17</td>
<td>This station is part of a switched network (S), and is transmitting (T). Because polling is not used, column 17 is blank.</td>
</tr>
<tr>
<td>18-19</td>
<td>EBCDIC (E) and the transparency feature (Y) are both used.</td>
</tr>
<tr>
<td>20</td>
<td>Automatic answer (A) is used by this called station.</td>
</tr>
<tr>
<td>32-47</td>
<td>Explicit station identification sequences are given for each station. Station IDs help ensure data security on the switched network. Local and remote IDs are specified in hexadecimal on the CRTCTLBSC command.</td>
</tr>
<tr>
<td>53-54</td>
<td>The permanent-error indicator used is 25.</td>
</tr>
<tr>
<td>55-57</td>
<td>The data communications line is kept open for 70 seconds when no messages are being sent or received. After 70 seconds elapse, a permanent error condition results if the line is still not being used.</td>
</tr>
</tbody>
</table>

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
TFilenameCSCCTS.............LIdthis+LIdremot....IERWtmInLPoAd...........

**Figure 108 (Part 2 of 3). Send-Only Program**
SAMPLE PROGRAMS

RPG Calculation Specifications

C* If a permanent error occurs, LR is set on to enter end-of-job processing.
C*
C  25  SETON                                                             LR
C  02N25  AMOUNT  ADD  MANSUM  MANSUM  62
CLIN25  MANSUM  ADD  TOTSUM  TOTSUM  62

RPG Output Specifications

C* If a permanent error occurs, a message printed at total time
C* identifies the record being processed when the permanent error
C* occurred.
C* Because of record blocking and the use of dual I/O areas, not all
C* the records preceding the record identified were transmitted.
C*
OSTOPT  T  25
O.................N01N02N03Field+YBEnd+PConstant/editword+++++++++++++++*
0          19 'PROGRAM INTERRUPTED'
0          36 'WHILE PROCESSING'
0          SLSMAN  51
O*
O* When on, the permanent-error indicator prevents the program from
O* transmitting totals
O*
OWKLYSMRYT  LIN25
0          SLSMAN  10
0          MANSUM  B  30
0          T  LRN25
0          20 'TOTAL = '
0          TOTSUM  30

Figure 108 (Part 3 of 3). Send-Only Program
AS/400 System to 3740

Figure 109 shows a program that receives two files from an IBM 3740, then sends two files to the 3740. The first file from the 3740 forms the input file BS1; the second file forms BS2. The data received as input to BS1 and BS2 is written by the AS/400 system to the output file PRINTER. Then the AS/400 system reads the disk file FILEA. Records that start with 1 are sent to the 3741 in file BS3. Records that start with 2 are sent to the 3741 in file BS4. (All records that start with 1 must precede any record that starts with 2.) The job ends when the last record from the disk has been read.

AS/400 Configuration Descriptions

1. CRTLINBSC
   - LIND(BSCLIN2)
   - RSRCNAME(LIN021)
   - CNN(*NONSWTPP)
   - CODE(*EBCDIC)

2. CRTCTLBSC
   - CTLD(BSCCTL2)
   - CNN(*NONSWTPP)
   - APPTYPE(*PGM)
   - LINE(BSCLIN2)

3. CRTDEVBSC
   - DEVD(BSCDEV2)
   - LOCADR(00)
   - RMTLOCNAME(RMTLOC2)
   - APPTYPE(*RPGT)
   - BLOCK(*NOSEP)
   - RCDLEN(80)
   - BLKLEN(480)
   - GRPSEP(*DEV3740)
   - CTL(BSCCTL2)

4. // SESSION LOCATION-RMTLOC2,SYMID-9#
Figure 109 (Part 1 of 2). AS/400 System-to-3740 Program
Figure 109 (Part 2 of 2). AS/400 System-to-3740 Program

Send Interspersed with Receive

Figure 110 on page 336 shows a program that reads a disk file, STUDENT, containing information about a student test, and then sends that information in a BSC file, GRADES, to a remote station. While the AS/400 system is sending data, the other station might interrupt to send back data in a file called RESULTS. When this interruption occurs, the RPG program turns on record-available indicator 04, reads the input file RESULTS, and prints it as the output file PRINTER. Then the AS/400 system continues sending file GRADES.
AS/400 Configuration Descriptions

1. CRTLINBSC
   LIND(BSCLIN3)
   RSRCNAME(LIN021)
   CNN(*SWTPP)
   SWTCTNN(*ANS)
   AUTOANS(*YES)
   CODE(*EBCDIC)
   TMTRTY(10)
   RCVRTY(10)

2. CRTCTLBSC
   CTLD(BSCCTL3)
   CNN(*SWTPP)
   APPTYPE(*PGM)
   INLCNN(*ANS)
   LCLID(F9F8F7F6)
   RMTID(F5F6F7F8)
   SWTCLIST(BSCLIN3)

3. CRTDEVBSC
   DEVD(BSCDEV3)
   LOCADR(00)
   RMTLOCNAME(RMTLOC3)
   APPTYPE(*RPGT)
   BLOCK(*NONE)
   RCDLEN(80)
   BLKLEN(80)
   TRNSPY(*YES)
   GRPSEP(*EOT)
   CTL(BSCCTL3)

4. CHGLINBSC LIND(BSCLIN3) SWTCLIST(BSCCTL3).

5. // SESSION LOCATION-RMTLOC3,SYMIC-9#

---

3 F9F8F7F6 is hexadecimal for 9876. The character format is used on System/36 for this field, but the hexadecimal format is used on AS/400 system.

4 F5F6F7F8 is hexadecimal for 5678. The character format is used on System/36 for this field, but the hexadecimal format is used on AS/400 system.
SAMPLE PROGRAMS

RPG File Description Specifications

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-14</td>
<td>RESULTS and GRADES are BSCA files.</td>
</tr>
<tr>
<td>15</td>
<td>RESULTS receives data; therefore, it is an input file. GRADES is transmitted and is an output file.</td>
</tr>
<tr>
<td>16</td>
<td>A BSCA file must be a demand file to receive data intermittently.</td>
</tr>
<tr>
<td>19</td>
<td>BSCA files always have a fixed-length format.</td>
</tr>
<tr>
<td>20-27</td>
<td>Neither BSCA file is blocked.</td>
</tr>
<tr>
<td>40-46</td>
<td>BSC is the device for BSCA files.</td>
</tr>
</tbody>
</table>

RPG CONTROL AND FILE DESCRIPTION SPECIFICATIONS

Control Specifications

| Line | File Name | Form Type | Alternate Collating Sequence | Inquiry | Size to Execute | Size to Compile | Object Output | Listing Options | Currency Symbol | Date Format | Date Edit | Inverted Print | Reserved | Debug | Reserved | Sign Handling | 1P Forms Position | Indicator Setting | File Translation | Punch MFCU Zeros | Nonprint Characters | Reserved | Table Load Halt | Shared I/O | Field Print | Formatted Dump | RPG to RPG II Conversion | Number of Formats | S/3 Conversion | Subprogram | CICS/DL/I |
|------|-----------|-----------|------------------------------|---------|-----------------|----------------|---------------|----------------|----------------|----------------|-----------|-----------|--------------|---------|-------|-----------|----------------|----------------------|------------------|------------------|----------------|----------------|----------------|---------|--------------|-----------|-----------|-------------|---------------------|-----------------|-----------------|-----------|---------|
| 12   | STUDENT   | P         |                             |         |                 |                |               |                |                |                |           |           |              |         |       |           |                |                      |                 |                 |               |                |              |         |             |           |           |
| 13   | RESULTS   | D         |                             |         |                 |                |               |                |                |                |           |           |              |         |       |           |                |                      |                 |                 |               |                |              |         |             |           |           |
| 14   | GRADES    | D         |                             |         |                 |                |               |                |                |                |           |           |              |         |       |           |                |                      |                 |                 |               |                |              |         |             |           |           |
| 15   | PRINT     | D         |                             |         |                 |                |               |                |                |                |           |           |              |         |       |           |                |                      |                 |                 |               |                |              |         |             |           |           |
| 16   |           |           |                             |         |                 |                |               |                |                |                |           |           |              |         |       |           |                |                      |                 |                 |               |                |              |         |             |           |           |

Figure 110 (Part 1 of 5). Send-Interspersed-with-Receive Program
RPG Telecommunications Specifications

Column Description

7-14 RESULTS and GRADES are the BSCA files for this program.
15 This station is part of a switched network (S).
16 RESULTS is an input file and receives data (R). GRADES is an output file and is transmitted (T).
17 Because polling is not used, column 17 is blank.
18-19 EBCDIC (E) and the transparency feature (Y) are both used.
10 Automatic answer is used by this station.
32-47 Explicit station identification sequences are given. Station IDs help ensure data security on the switched network.
53-54 The permanent-error indicator used is 44.
55-57 The data communications line is kept open for 70 seconds when no messages are being sent or received. After 70 seconds elapse, a permanent error condition results if the line still is not being used.
58-59 The record available indicator is 04; it is set on when the other system sends an RVI (reverse interrupt). This indicates it is ready to send a record received by RESULTS.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

<table>
<thead>
<tr>
<th>Extension Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
</tbody>
</table>

RPG TELECOMMUNICATIONS SPECIFICATIONS

<table>
<thead>
<tr>
<th>Telecommunications Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
</tbody>
</table>

Figure 110 (Part 2 of 5). Send-Interspersed-with-Receive Program
RPG Input Specifications

Lines 07-09: RESULTS receives records from the remote station. These records contain a student number and the student’s score. Student scores are calculated by the remote station from the data transmitted from GRADES.

---

<table>
<thead>
<tr>
<th>Field Name</th>
<th>External Field Name</th>
<th>Field Location</th>
<th>Record Identification Codes</th>
<th>Control Level (L1-L9)</th>
<th>RPG Field Name</th>
<th>Field Indicators</th>
<th>Occurs nTimes</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT AA</td>
<td>01 1 CA 2 CN 3 CS</td>
<td>5 78 QUES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>02 1 CS 2 CT 3 CD</td>
<td>3 5 STUD ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESULTS CC</td>
<td>03 1 CR</td>
<td>3 5 STD NT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 110 (Part 3 of 5). Send-Interspersed-with-Receive Program
RPG Calculation Specifications

The calculation sequence obtains the scores computed by the remote station from GRADES and makes them available for exception output to PRINT.

**Lines 01-06:** Loop 1 is processed during detail time, that is, before end of file is reached on STUDENT. When the remote station sets on the record available indicator (04), the READ operation accepts a record from the remote station and places it in RESULTS. Indicator 10 is set on, and indicator 04 is set off when READ finds an end-of-file condition. Until indicator 04 is set off and indicator 10 is set on, records are placed in RESULTS and are available for exception output to PRINT. Loop 1 can be entered or reentered anytime 04 is set on, except after end of file is reached on STUDENT (indicator 10 is set off). Therefore, after one group of records is read by the receiving station, the receiving station must set off the end-of-file indicator (10) so that the next group of records can be read.

**Lines 07-12:** Loop 2 is processed during total time, that is, after end of file is reached on STUDENT and the LR indicator is set on. Loop 2 processes records for results in the same way as loop 1 with one difference: Loop 2 always compares the student number received by RESULTS (STDNT) to the last student number transmitted (STUDID). Loop 2 continues until these numbers are equal. This continuing loop ensures that the results for all the student records transmitted, are received. The READ statement after Loop 2 will read the EOF for the file so that transmission can end normally.

If a permanent error occurs during a transmit or receive operation, LR and 10 are set on to enter end-of-job processing. Indicator 09 is then set off to exit from the total-time loop.

---

**RPG Calculation Specifications**

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Form Type</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Name</th>
<th>Length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>LOOP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>C</td>
<td></td>
<td>04</td>
<td>READ</td>
<td>RESULTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>03</td>
<td>C</td>
<td></td>
<td>03N10</td>
<td>EXCPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>C</td>
<td></td>
<td>44</td>
<td>SETOF</td>
<td>LR10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>C</td>
<td></td>
<td>03N10</td>
<td>GO TO</td>
<td>LOOP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>C</td>
<td></td>
<td></td>
<td>SETOF</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>C</td>
<td></td>
<td>LR</td>
<td>LOOP2</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>C</td>
<td></td>
<td>LR</td>
<td>READ</td>
<td>RESULTS</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>C</td>
<td></td>
<td>LR 03N10</td>
<td>EXCPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C</td>
<td></td>
<td>LR 44</td>
<td>SETOF</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>C</td>
<td></td>
<td>LNR44</td>
<td>STDNT</td>
<td>COMP</td>
<td>STUDID</td>
<td>909</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td></td>
<td>LR 09</td>
<td>GO TO</td>
<td>LOOP2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>C</td>
<td></td>
<td>LNR09</td>
<td>READ</td>
<td>RESULTS</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td></td>
<td>LNR09</td>
<td>READ</td>
<td>RESULTS</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 110 (Part 4 of 5). Send-Interspersed-with-Receive Program*
RPG Output Specifications

Lines 01-06: GRADES is transmitted to the remote stations.

Lines 07-08: Permanent error prints an error message.

Lines 09-13: Records received by RESULTS are printed.

Figure 110 (Part 5 of 5). Send-Interspersed-with-Receive Program
Chapter 12. Primary/Secondary/Multifile Processing

In an RPG II program, the processing of a primary input file and one or more secondary input files, with or without match fields, is termed multifile processing. Selection of records from more than one file based on the contents of match fields is known as multifile processing by matching records. Multifile processing can be used with input files that are designated as primary/secondary files.

Multifile Processing with No Match Fields

If match fields are not used in a program that has more than one input file, the program processes records from one file at a time. After the program processes all the records from one file, it processes all the records from the next file. Files are processed in this order:

1. Primary file, if specified
2. Secondary files in the order in which they are coded in the file description specifications.

Multifile Processing with Match Fields

When match fields are used, the program selects the records for processing based on the contents of the match fields.

When the match field in the primary file record is the same as the match field in one or more of the secondary file records, the matching-record indicator turns on. Your program can use this indicator to control calculation or output operations for the matching records.

If some files contain match fields and other files do not, the program processes all the files without match fields before it processes any files with match fields. The files without match fields are processed in this order:

1. Primary file, if specified
2. Secondary files in the order in which they are coded in the file description specifications.

If some records in a file use match fields and other records do not, the records without match fields are processed immediately after the record they follow.

Coding Matching Records

To process matching records, entries are required on both the file description specifications and the input specifications.

File Description Specifications

For matching records, the following entries are required on the file description specifications. For information about columns not mentioned here, see Chapter 6, “Using a DISK File” for DISK files, Chapter 9, “Using a SPECIAL File” for SPECIAL files, or Chapter 10, “Using a CONSOLE, KEYBOARD, or CRT File” for CONSOLE files.
Column 15 must contain I, U, or C to indicate that the file is an input, update, or combined file.

Column 16 must contain P or S to indicate that the file is a primary or secondary file.

Column 17 must contain E or blank. E indicates that the program must process all records from the file before the program can end. Blank indicates that the program can end before it processes all records from the file. However, if column 17 is blank for all files, the program must process all records from every file before it can end.

Column 18 must contain A, D, or blank. A indicates that the program checks that the records in the file are in ascending sequence. D indicates that the program checks that the records are in descending sequence. Sequence checking is required for all files that use match fields. If column 18 is left blank and match fields are used, ascending sequence is assumed. Column 18 must contain the same entry for all files that specify matching fields on the input specifications.

Columns 40 through 46 must contain DISK, CONSOLE, or SPECIAL.
Input Specifications

For matching records, the following entries are required on the input specifications. For information about columns not mentioned here, see Chapter 23, “Input Specifications.”

Columns 61 and 62 must contain any value from M1 through M9 to indicate that the field named in columns 53 through 58 is a match field.

Rules for Coding Match Fields

You can use one field, many fields, or an entire record to match records.

You can use as many as nine match fields by coding a different value, M1 through M9, for each match field.

M1 through M9 are not indicators. They identify the match fields, and they cause the matching-record indicator (MR) to turn on.

Not all files used by the program must have match fields. Not all record types within one file must have match fields either. However, at least one record type from two files must have match fields in order for the files to match.

The same number of match fields must be coded for all record types that are used in matching. The same matching record values must also be used for all types.

Whenever more than one match-field value is used, all match fields must match before the matching-record indicator (MR) turns on. For example, if you use match-field values M1, M2, and M3, all three fields from one record must match all three fields from the other record. A match on only the M1 and M2 fields does not turn on the matching-record indicator (MR).

If you use more than one match field for a record type, all the fields are combined and treated as one continuous match field (see Figure 111 on page 344). The fields are combined according to descending sequence (M9 to M1) of match-field values.
Three record types are used in matching records. All record types have match fields specified, and all use the same values (M1, M2, M3) to indicate the fields that must match. The MR indicator turns on only if all three match fields in either of the record types from the MASTER file are the same as all three fields from the record in the WEEKLY file.

The three match fields in each record type are combined and treated as one match field organized as follows:

DIVSON   DEPT  EMPLNO
M3       M2     M1

The order in which the fields are specified by the input specifications does not affect the organization of the match fields in the computer.
Match fields cannot be split. That is, the same match-field value cannot be used twice for one type of record.

All match fields that have the same match-field value (M1 through M9) must be the same length. If the match field contains packed data, the zoned-decimal length, which is \((2 \times \text{packed-decimal length}) - 1\), is used as the length of the match field.

Record positions of different match fields can overlap, but the total length of all fields must not be more than 256 characters.

All match fields that have the same match-field value (M1 through M9) must be the same type (alphameric or numeric). If any of the match fields is described as numeric, all match fields that have the same match-field value are considered numeric.

When numeric fields having decimal positions are matched, they are treated as if they had no decimal position. For example, 3.46 is considered equal to 346.

Only the digit portions of numeric match fields are compared. Even if a field is negative, it is considered positive because the sign of the numeric field is ignored. Thus, a -5 matches a +5.

The match field value must be valid alphameric or numeric characters. Other values may cause unpredictable results.

A field coded as binary (B in column 43) cannot have a match-field value. However, a field coded as packed (P in column 43) can have a match-field value.

Field names are ignored in match field operations. Therefore, fields from different record types that have the same match-field value can have the same name.

If the program uses an alternative collating sequence, alphameric fields are matched according to the alternative sequence.

Additional rules apply to match fields when entries are coded in columns 63 and 64 of the input specifications (see “Columns 63-64 (Field Record Relation)” on page 581 in Chapter 23, “Input Specifications”).
Processing Matching Records

Figure 112 is a flowchart for the processing of matching records.

1. The program determines if there is more than one input file.
2. If there is more than one input file, the program reads one record from each file and moves the records to a hold area. The program compares the contents of the match fields in these records to determine the record to process next. Records that are not processed stay in the hold area, where they are compared again during the next program cycle. During the next program cycle, the program reads a record from the file containing the record that was processed during the previous program cycle.
3. The program determines if the match fields are in sequence.
4. If the match fields are not in sequence, the program stops.
If the match fields are in sequence, the program processes the record selected.

When the match field from one record is the same as the match field from another record, the matching-record indicator (MR) turns on. Your program can use this indicator to control the calculation or output operations that you want to do for matching records. For example, when the matching-record indicator (MR) turns on, you can enter data from primary records into their matching secondary records because the program processes the primary record before the matching secondary record. However, you can enter data from the first record of a secondary file to matching primary records only when you use look-ahead fields (see “Look-Ahead” under “Columns 19-20 (Record-Identifying Indicator, **, DS)” on page 557 in Chapter 23, “Input Specifications”).

When a record from the primary file matches a record from the secondary file, the program processes all the matching records from the primary file first. Then it processes all the matching records from the secondary file. The record-identifying indicator that identifies the record type just selected is on when the record is processed. Programs often use record-identifying indicators to control the type of processing.

When records in files that are in ascending order do not match, the program first processes the record whose match field contains the lowest value. When records in files that are in descending order do not match, the program first processes the record whose match field contains the highest value.

If a record type does not use a match field, the program processes it immediately after the record is read. The matching-record indicator is off. If this record type is first in the file, it is processed first even if it is not in the primary file.

The program checks that the contents of match fields are in the correct sequence. If the contents are not in sequence, error message RPG9032 File [file name ] contains a record not in sequence, is displayed. If you respond by choosing option 1, the program does not process the record that is out of sequence. When the program starts again, it reads the next record from the same file. Therefore, all match fields must be in ascending order, or all must be in descending order (see “Column 18 (Sequence)” on page 497 in Chapter 19, “File Description Specifications”).

Figure 113 on page 348 shows how to code the file description and input specifications for a program that uses match fields in three DISK files. Figure 114 on page 349 and Figure 115 on page 351 show how the program coded in Figure 113 on page 348 selects records from the three files.
Figure 113. Coding for Match Fields in Three DISK Files
The records from the three DISK files are selected in the order indicated by the circled numbers.

Figure 114. Selecting Matching Records from Three DISK Files
Table 6. Order of Primary/Secondary/Multifile Processing

<table>
<thead>
<tr>
<th>Cycle</th>
<th>File Processed</th>
<th>Indicators On</th>
<th>Reason for Record Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>02</td>
<td>No match field specified</td>
</tr>
<tr>
<td>2</td>
<td>PRIMARY</td>
<td>02</td>
<td>No match field specified</td>
</tr>
<tr>
<td>3</td>
<td>FIRST SEC</td>
<td>04</td>
<td>No match field specified</td>
</tr>
<tr>
<td>4</td>
<td>SEC SEC</td>
<td>05</td>
<td>Second secondary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No primary match</td>
</tr>
<tr>
<td>5</td>
<td>PRIMARY</td>
<td>01,MR</td>
<td>Primary matches first secondary</td>
</tr>
<tr>
<td>6</td>
<td>PRIMARY</td>
<td>01,MR</td>
<td>Primary matches first secondary</td>
</tr>
<tr>
<td>7</td>
<td>FIRST SEC</td>
<td>03,MR</td>
<td>First secondary matches primary</td>
</tr>
<tr>
<td>8</td>
<td>FIRST SEC</td>
<td>03</td>
<td>First secondary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No primary match</td>
</tr>
<tr>
<td>9</td>
<td>FIRST SEC</td>
<td>03</td>
<td>First secondary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No primary match</td>
</tr>
<tr>
<td>10</td>
<td>SEC SEC</td>
<td>05</td>
<td>Second secondary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No primary match</td>
</tr>
<tr>
<td>11</td>
<td>PRIMARY</td>
<td>01</td>
<td>Primary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No secondary match</td>
</tr>
<tr>
<td>12</td>
<td>PRIMARY</td>
<td>01,MR</td>
<td>Primary matches second secondary</td>
</tr>
<tr>
<td>13</td>
<td>PRIMARY</td>
<td>02</td>
<td>No match field specified</td>
</tr>
<tr>
<td>14</td>
<td>SEC SEC</td>
<td>05,MR</td>
<td>Second secondary matches primary</td>
</tr>
<tr>
<td>15</td>
<td>SEC SEC</td>
<td>05,MR</td>
<td>Second secondary matches primary</td>
</tr>
<tr>
<td>16</td>
<td>SEC SEC</td>
<td>06</td>
<td>No match field specified</td>
</tr>
<tr>
<td>17</td>
<td>PRIMARY</td>
<td>01,MR</td>
<td>Primary matches both secondary files</td>
</tr>
<tr>
<td>18</td>
<td>FIRST SEC</td>
<td>03,MR</td>
<td>First secondary matches primary</td>
</tr>
<tr>
<td>19</td>
<td>FIRST SEC</td>
<td>04</td>
<td>No match field specified</td>
</tr>
<tr>
<td>20</td>
<td>SEC SEC</td>
<td>05,MR</td>
<td>Second secondary matches primary</td>
</tr>
<tr>
<td>21</td>
<td>FIRST SEC</td>
<td>03</td>
<td>First secondary low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No primary match</td>
</tr>
<tr>
<td>22</td>
<td>PRIMARY</td>
<td>01,MR</td>
<td>Primary matches both secondary files</td>
</tr>
<tr>
<td>23</td>
<td>FIRST SEC</td>
<td>03,MR</td>
<td>First secondary matches primary</td>
</tr>
<tr>
<td>24</td>
<td>FIRST SEC</td>
<td>03,MR</td>
<td>First secondary matches primary</td>
</tr>
<tr>
<td>25</td>
<td>SEC SEC</td>
<td>05,MR</td>
<td>Second secondary matches primary</td>
</tr>
<tr>
<td>26</td>
<td>SEC SEC</td>
<td>05,MR</td>
<td>Second secondary matches primary</td>
</tr>
</tbody>
</table>
The first record from each file is read. The P and S records have no match field, so they are processed before the T record that has a match field. Because the P record comes from the primary file, it is selected for processing first.

The next P record is read. It contains no match field and comes from the primary file, so the new P record is also selected for processing before the S record.

The next P record read has a match field. The S record has no match field, so it is selected for processing.

The next S record is read. All three records have match fields. Because the value in the match field of the T record is lower than the value in the other two, the T record is selected for processing.

The next T record is read. The matching P and S records both have the low match field value, so they are processed before the T record. Because the matching P record comes from the primary file, it is selected for processing first.

The next P record is read. Because it contains the same match field and comes from the primary file, the new P record is selected instead of the S record.

Figure 115 (Part 1 of 2). Selecting Matching Records from Three DISK Files
Step 7

The next P record is read. The value of the match field in the S record is the lowest of the three, so the S record is selected for processing.

Step 8

The next S record is read. Because the S and T records match and have the lowest match field, they are selected before the P record. Because the S record comes from the first secondary file, it is selected for processing before the T record.

Step 9

The next S record is read. Because it also has the same match field as the S record just selected, it too is selected before the T record.

Step 10

The next S record is read. The T record contains the lowest match field value, and is selected for processing.

Figure 115 (Part 2 of 2). Selecting Matching Records from Three DISK Files
Chapter 13. Using Indicators

The RPG program cycle is built around indicators. To you, an indicator is a two-character entry on a specification form; the indicator turns on or off as the result of an operation, or it conditions when or if an operation occurs. To RPG, an indicator is an internal switch; the program uses the indicator to determine when or if an operation occurs in the program and what to do when the operation occurs.

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

Figure 116 on page 354 lists all the two-character entries that can be used for each type of indicator.
<table>
<thead>
<tr>
<th>Defined on RPG Specifications</th>
<th>01-99</th>
<th>1P</th>
<th>H1-H9</th>
<th>L1-L9</th>
<th>LR</th>
<th>MR</th>
<th>0A-0G</th>
<th>0V</th>
<th>U1-U8</th>
<th>KA-KN</th>
<th>KP-KY</th>
<th>LO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow indicator</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record-identifying indicator¹</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control-level indicator</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field indicator</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resulting indicator</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X²</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X³</td>
<td></td>
</tr>
<tr>
<td>Defined by RPG Program</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External indicator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal indicator</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used to Represent a Condition</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File-conditioning indicator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field-record-relation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X⁵</td>
<td>X⁵</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indicator¹</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level-zero indicator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command-key indicator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halt indicator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditioning indicators</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on calculation specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditioning indicators</td>
<td>X</td>
<td>X⁶</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on output specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
1. Not allowed on look-ahead field.
2. Not allowed for SETOF operation.
3. Allowed for SET, KEY, and SETOF operations only.
4. Not allowed for table input files.
5. When field named is not a match field or a control field.
6. Only for detail or heading lines.
7. Cannot condition an exception line, but can condition fields within the exception record.

Figure 116. Entries for Each Type of Indicator
INDICATORS DEFINED ON RPG SPECIFICATIONS

You define the following indicators on the RPG specifications:

- Overflow indicator defined in columns 33 and 34 of the file description specifications
- Record-identifying indicator defined in columns 19 and 20 of the input specifications
- Control-level indicator defined in columns 59 and 60 of the input specifications
- Field indicator defined in columns 65 through 70 of the input specifications
- Resulting indicator defined in columns 54 through 59 of the calculation specifications.

You must define these indicators in the specifications before you can use them on other specifications to condition operations in the program.

Overflow Indicators

An overflow indicator is defined by an entry in columns 33 and 34 of the file description specifications. The purpose of an overflow indicator is to signal when the end of a printed page has been passed. The indicator is assigned to the PRINTER file and turns on when the overflow line on the page is passed. This could occur at exception, detail, or total output time. You use the overflow indicator to condition those lines that you want to print at the end of one page or at the beginning of another.

Figure 117 on page 356 shows the RPG program cycle related to overflow indicators.
OVERFLOW INDICATORS

Figure 117. RPG Program Cycle for Overflow Indicators

The two-character entries allowed as overflow indicators are:

OA through OG

OV

On the file description specifications:

- Columns 33 and 34 define an overflow indicator for each PRINTER file. If no overflow indicator is defined, the RPG program automatically handles overflow.

On the calculation specifications:

- Columns 9 through 17 can contain the overflow indicator defined in columns 33 and 34 of the file description specifications to condition calculations processed when overflow occurs.
- Columns 54 through 59 can contain the overflow indicator defined in columns 33 and 34 of the file description specifications set on or off as the result of calculations.
On the **output specifications**:
- Columns 23 through 31 must contain the overflow indicator defined in columns 33 and 34 of the file description specifications to condition all lines that are to be written to the associated printer when overflow occurs.

For more information on using overflow indicators, see “Handling Overflow” on page 265.

### Record-Identifying Indicators

You assign a record-identifying indicator to each type of record in the input file. You can also use a record-identifying indicator to associate a field with a particular record type by using the record-identifying indicator as a field-record-relation indicator. You do not have to assign the record-identifying indicators in any order. If certain operations in calculations and output are processed for one record type only, you can condition those operations by the appropriate record-identifying indicator. By this method you can tell the RPG program the operations to do when it processes a specific record type. When several record types are specified in an OR relationship, all fields that do not have a field-record-relation indicator in columns 63 and 64 of the input specifications are associated with all record types in the OR relationship.

After the program selects the next record to process, it turns on the record-identifying indicator that you assigned to that record type. You can use this indicator to condition total and detail operations. This indicator is turned off by RPG before input occurs at input time of the RPG cycle.

**Note:** If you use a READ, READE, READP, or CHAIN operation in your program, input occurs during calculation time and your record-identifying indicators are not turned off before the READ, READE, READP, or CHAIN operation.

Figure 118 on page 358 shows specific steps in the RPG program cycle related to record-identifying indicators.

The two-character entries allowed as record-identifying indicators are:
- 01 through 99
- H1 through H9
- L1 through L9
- LR

On the **input specifications**:
- Columns 19 and 20 define the record-identifying indicator. These columns should contain a different record-identifying indicator for each record type in a file.
- A record-identifying indicator must be assigned to the first input record in a WORKSTN file if this record is blank. The first input record is blank unless:
  - A read under format is run.
  - PDATAY-YES is specified in the procedure that called the program (see the explanation of the $MAINT utility program in *System Reference for the System/36 Environment*, or the explanation of end of job in *SEU User's*
Output to the WORKSTN file was done first.

**Note:** When you use a control-level indicator (L1 through L9) as a record-identifying indicator and it turns on to indicate the type of record read, only that one control-level indicator turns on. All lower control-level indicators that you used remain unchanged.

---

**AND Relationship**

Each line on the input specifications can contain up to three identifying characters (columns 27, 34, and 41). If the identification code you are using consists of more than three characters, an AND line must be used to describe the additional characters. To specify an AND line, code AND in columns 14 through 16.

You can combine any number of AND lines and any number of OR lines to describe the record-identifying code for a record sequence. The record must contain all the characters specified as its record identification code before the
record-identifying indicator turns on. You cannot use a record-identifying indicator in the AND line of an AND relationship. AND lines are not allowed on CONSOLE files used for interactive data entry.

**OR Relationship**

If a particular record type can be identified by two different codes, you must use OR lines to specify that either of the codes can be present to identify the record. Any number of AND and OR lines can be used. To specify an OR line, code the word OR in columns 14 and 15.

You can use the OR relationship to assign the same record-identifying indicator to two or more different record types if the same operation is processed on all record types. You can also use record-identifying indicators on OR lines or every record type in the OR relationship that requires special processing.

When several record types are used in an OR relationship, all fields that do not have a field-record-relation indicator are associated with all record types in the OR relationship.

**Example of Using Record-Identifying Indicators**

You can use record-identifying indicators in a billing program. Suppose that you keep a monthly file that contains records of purchases and payments made by each customer. In addition, the file contains a balance forward record for each customer. Figure 119 on page 360 shows the three input record types used and the output records required.
Figure 119. Input and Output for a Billing Program
The three record types are defined on the input specifications. Each type has a different record-identifying indicator. The record-identifying indicators are then used to show the operations that are done for each record type. Figure 120 shows the input, calculation, and output specifications for the program. Use these specifications to help you follow, step by step, the operations done in the program cycles shown in Figure 121 on page 362.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCNCpos2NCNCpos3NCNC........................................**
IBILLING 011 10 96 CB
I.................................PFromTo++DField+L1MIFrPoNeEq...*
I 1 7 NUM L1
I 8 30 NAME
I 31 382BALFOR
I 02N020 96 CA
I 1 7 NUM
I 31 382PURCHS
I 03N030 96 CS
I 1 7 NUM
I 31 382PAYMNT
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CCIN01N02N03Factor1+++Opctefactor2+++ResultLenDHHiliOEqComments++++++
C 20 ADD PURCHS BALFOR
C 30 SUB PAYMNT BALFOR
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaNo1No2No3Excptn..............................................**
OREPORT D 306 10
O...............................No1No2No3Field+YBEn+PConstant/editword+++++++...
0 NUM 15
0 NAME 45
0 BALFORA 60
0 D 1 20 PURCHS1 20
0 D 1 30 PAYMNT1 40
0 T 12 L1 BALFORA 60
0 61 '*'
```

*Figure 120. Specifications Using Record-Identifying Indicators for a Billing Program*
Figure 121 (Part 1 of 3). RPG Program Cycle for Record-Identifying Indicators
Figure 121 (Part 2 of 3). RPG Program Cycle for Record-Identifying Indicators
Figure 121 (Part 3 of 3). RPG Program Cycle for Record-Identifying Indicators
Control-Level Indicators

A control-level indicator tells the program when calculation or output operations are done. You can assign a control-level indicator to any field; this field is then known as a control field. The program checks the field for a change in information. When the information changes, a control break occurs. All records that have the same information in the control field are known as a control group.

Whenever the program reads a record containing a control field, it compares the data in the control field with data in the same control field from the previous record. When a control break occurs, the control-level indicator turns on. Operations conditioned by the control-level indicator are then processed.

There are nine control levels (level 1 through level 9). Each control level has a corresponding control-level indicator (L1 through L9). When a control-level indicator turns on, all control-level indicators with a lower number also turn on. For example, if indicator L3 turns on, indicators L2 and L1 automatically turn on. However, when a control-level indicator used as a record-identifying indicator turns on to indicate the type of record read, or when the SETON operation turns on a control-level indicator, only that single control-level indicator turns on. In that case, all lower control-level indicators remain unchanged.

The two-character entries allowed as control-level indicators are:

- L0
- L1 through L9
- LR

On the input specifications:

- Columns 59 and 60 define the control-level indicator used to condition input fields so that the program can check for any change in the information in this field. L1 through L9 are the only control-level indicators allowed here.

On the calculation specifications:

- Columns 7 and 8 must contain L0, LR, or the control-level indicator defined in columns 59 and 60 of the input specifications to specify a calculation operation processed when the appropriate control break occurs at total calculation time.

- Columns 9 through 17 can contain L0, LR, or the control-level indicator defined in columns 59 and 60 of the input specifications to control the conditions under which a detail calculation operation is done on the record that caused the control break.

On the output specifications:

- Columns 23 through 31 can contain L0, LR, or the control-level indicator defined in columns 59 and 60 of the input specifications to tell the program the conditions under which a total record or field is written. Also, the control-level indicator can condition detail output operations to be done on the record that caused the control break.
Assigning Control-Level Indicators

The following points apply to control-level indicators:

- If the same control-level indicator is used in more than one record type or in more than one file, the control fields associated with that control-level indicator must be the same length and the same type (alphabetic or numeric).

- In the same record type, record positions in control fields assigned different control-level indicators can overlap (see Figure 122). However, the total number of positions assigned as control fields must not be greater than 256.

- Field names are ignored in control-level operations. Therefore, fields from different record types that have been assigned the same control-level indicator can have the same name.

- Control levels need not be assigned in any order. For example, you can use indicator L2 before L1. You can also leave gaps in the control levels you assign.

- When numeric control fields with decimal positions are compared to determine if a control break has occurred, they are always treated as if they have no decimal positions. For instance, 3.46 is considered equal to 346.

- If a field is specified as numeric, only the digit portion determines if a control break has occurred. This means that a field is always considered positive. For instance, -5 is considered equal to +5.

- All control fields given the same control-level indicator are considered numeric if any one of those control fields is described as numeric (that is, if column 52 of the calculation specifications has an entry). Therefore, when numeric control fields are compared to determine if the information has changed, only the digit portion of each character is compared.

- Control fields are initialized to hexadecimal zeros.

![Figure 122. Overlapping Control Fields in a Disk Record](image-url)
• A control break can occur after the first record containing a control field is read. The control fields in this record are compared with an area in storage that contains hexadecimal zeros. Because the fields being compared are not from two different records, total calculations and total output operations are bypassed for this cycle. A control break does occur then, but it is not considered a true control break.

• If different record types in a file do not have the same number of control fields, unwanted control breaks can occur. See Figure 123 on page 368 for an example of how to avoid unwanted control breaks.

• A control field cannot be specified as having a binary format (B in column 43 of the input specifications). However, it can be specified as having a packed-decimal format (P in column 43 of the input specifications).

• A control field can be related to a particular record type in an OR relationship with a field-record-relation indicator. If the control field does not have a field-record-relation indicator, the control field is used with all record types in the OR relationship.
CONTROL-LEVEL INDICATORS

Different record types normally contain the same number of control fields. However, some applications require a different number of control fields in some records.

The salesman records contain only the L2 control field. The item records contain both L1 and L2 control fields. With normal RPG coding, an unwanted control break is created by the first item record following the salesman record. This is recognized by an L1 control break immediately following the salesman record and results in an asterisk being printed on the line below the salesman record.

Output Showing Unwanted Control-Level Break

Corrected Output

Figure 123 (Part 1 of 2). Unwanted Control Breaks
This coding prevents the unwanted control break. Line 01 of the calculation specifications sets on indicator 11 when the salesman record is read. When the next item record causes an L1 control break, no total output is printed because indicator 11 is on (line 07 of output specifications). Detail calculations are then processed for the item record, and line 02 of the calculation specifications sets indicator 11 off. This allows the normal L1 control break to occur.

Figure 123 (Part 2 of 2). Unwanted Control Breaks
Split Control Fields

If a control field is made up of more than one field of a record, it is known as a **split control field**. A split control field is created when the same indicator is assigned to two or more fields (connected or unconnected) on the same record type.

All fields in one record that have the same control-level indicators are combined by the program in the order specified by the input specifications and are treated as one control field (see Figure 124).

The following rules apply to split control fields:

- For one control-level indicator, a field can be split in some record types and not in others if the field names are different. However, the length of the field, split or not, must be the same in all record types.
- The length of the portions of a split control field can vary for different record types if the field names are different. However, the total length of the portions must always be the same.
- No other specifications can come between lines that describe split control fields.
- If one section of a split control field is numeric, the whole field is considered numeric.
- A numeric split control field can have more than 15 characters if no portion of the split field has more than 15 characters and if the sum of all control fields is not more than 256 characters.
- A split control field cannot be made up of a packed-decimal field and a zoned-decimal field. Both portions of the control field must be packed decimal, or both must be zoned decimal.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC......................................................*
IMASTER AA 01 1 CM
I..............................................PFromTo++DField+L1M1FrPoNeEq...*    I*
I* All portions of a split control field must be assigned the same
I* control level indicator.
I*
I 28 31 CUSNO L4
I 15 20 ACCTNOL4
I 50 52 REGNO L4

*Figure 124. Split Control Fields*
Field Indicators

Field indicators are used to test a field on an input record for a plus, minus, zero, or blank value. You can use the appropriate field indicator to condition operations that are done only when a numeric field is plus, minus, or zero, or when an alphameric field is blank.

Note: A numeric field that is all blanks turns on an indicator used for all zeros. However, an alphameric field that is all zeros does not turn on an indicator used for all blanks. You cannot specify indicators in columns 65 through 68 for an alphameric field.

Field indicators turn on or off after data from the record to be processed moves into the processing area. Figure 125 shows the RPG program cycle related to field indicators.

For each program cycle, field indicators are set to reflect the result of the test on a field. If the condition tested for exists, they turn on; if the condition does not exist, they turn off. After the program tests the result of a field, a field indicator stays on or off until the program uses the same indicator as a resulting indicator.
RESULTING INDICATORS

When the indicator is on, any detail and total operations conditioned by the field indicator can be done before the program resets the indicator by testing a field. However, at total time the field indicator has the setting established in the previous cycle.

The following considerations apply to field indicators:

- A numeric input field can be assigned two or three field indicators. However, only the indicator that signals the result of the test on that field turns on; the others remain off.
- If the same field indicator is assigned to fields in different record types, its status is always based on the last record type selected.
- When different field indicators are assigned to fields in different record types, a field indicator turned on remains on until another record of that type is read. Similarly, a field indicator assigned to more than one field within a single record type always reflects the status of the last field defined.

Field indicators assigned in columns 65 through 70 can also be set on or set off by SETON or SETOF operations in the calculation specifications.

The two-character entries allowed as field indicators are:

- 01 through 99
- H1 through H9

Use the two-character entries 01 through 99 to test if a numeric field is plus, minus, zero, or blank. Use the two-character entries H1 through H9 to check for an error condition in your data.

On the input specifications:

- Columns 65 and 66 define a field indicator to check for a plus condition. The indicator turns on if the numeric field is greater than zero.
- Columns 67 and 68 define a field indicator to check for a minus condition. The indicator turns on if the numeric field is less than zero.
- Columns 69 and 70 define a field indicator to check for zeros or blanks. The indicator turns on if the numeric field is all zeros or if an alphameric field is all blanks.

On the output specifications:

- Columns 23 through 31 can contain a two-character entry, H1 through H9, to prevent data that causes an error from being used.

Resulting Indicators

Resulting indicators signal something about the result of a calculation operation. You can use a resulting indicator to condition any operation that depends on the result of the calculation.

You can use a resulting indicator in columns 54 through 59 on the calculation specifications to reflect the result of an operation, or to indicate an end-of-file condition, a no-record-found condition, or an exception/error condition. The indicator specified turns on only if the result field satisfies the condition being tested for. If the condi-
tion tested for is not met, the indicator is turned off. This indicator can then be used to condition following calculations or output operations (see Figure 126 on page 373). If you use the same indicator to test the result of more than one operation, the last operation that runs determines the setting of the indicator.

In Figure 126, indicators 10 and 20 in columns 54 through 57 are used to test for the different conditions in a subtract operation. These indicators are used to condition the calculations that must be done for a payroll job. Indicator 10 turns on if the hours worked (HRSWKD) are greater than 40 and is then used to condition all operations necessary to calculate overtime pay. Indicator 20 turns on if HRSWKD is less than 40. Indicator 20 is also used to condition other operations. In line 03, if indicator 20 is not on (the employee worked 40 or more hours), regular pay is calculated based on a 40-hour week. In line 06, if indicator 20 is on (employee worked less than 40 hours), pay is calculated based on less than a 40-hour week.

The two-character entries allowed as resulting indicators are:

- 01 through 99
- H1 through H9
- KA through KN, and KP through KY
- L1 through L9
- LR
- OA through OG, and OV
- U1 through U8

![Figure 126. Conditioning Operation (Resulting Indicators)](image)

On the calculation specifications:
- Columns 54 through 59 define the resulting indicator as turned on or off by the SETON or SETOF operation codes. The headings (high, low, and equal) for columns 54 through 59 have no meaning for SETON or SETOF operations.
- Columns 54 and 55 (plus or high) must define a resulting indicator when testing:
  - If the result field in an arithmetic operation is positive
  - If factor 1 is higher than factor 2 in a compare (COMP) operation
  - If factor 2 is higher than factor 1 in an array or table LOKUP operation
  - If a CHAIN operation is not successful
  - If each bit named in factor 2 is off for a TESTB operation
  - If the character tested in a TESTZ operation is one of the following: &d, A through I
  - If the numeric field entered in a KEY operation is positive
  - If the system operator has requested shutdown on a SHTDN operation.
Columns 56 and 57 (minus or low) must define a resulting indicator when testing:

- If the result field in an arithmetic operation is negative
- If factor 1 is lower than factor 2 in a compare (COMP) operation
- If factor 2 is lower than factor 1 in a table or array LOKUP operation
- If the bits named in factor 2 are of mixed status (some bits on, some bits off) for a TESTB operation
- If the character tested in a TESTZ operation is one of the following: - (minus), J through R
- If the numeric field entered in a KEY operation is negative
- If the ACQ, REL, NEXT, READ, or POST operation to a WORKSTN file is not successful.

Columns 58 and 59 (zero or equal) must define a resulting indicator when testing:

- If the result field in an arithmetic operation is zero
- If factor 1 is equal to factor 2 in a compare (COMP) operation
- If factor 2 is equal to factor 1 in a table or array LOKUP operation
- If the program reached the end of a file that is read by a READ operation, the end of an equal key for a READE operation, or the beginning of file condition for a READP operation
- If each bit named in factor 2 is on for a TESTB operation
- If the character tested in a TESTZ operation is any character other than &, A through I, - (minus), or J through R
- If the numeric field entered in a KEY operation is zero or an alphameric field is blank.

Indicators Not Defined on the RPG Specifications

You do not need to define all the indicators in your program to condition operations. External indicators (U1-U8) are defined by an OCL statement in a previous RPG program. The internal indicators first-page (1P) and last-record (LR) are defined for you by the RPG program cycle itself. The matching-record indicator (MR) is defined for you if you use M1 through M9 in columns 61 and 62 of the input specifications.

External Indicators

External indicators are usually set prior to processing by the OCL SWITCH statement or by a previous RPG program. External indicators are automatically read into the program when the program begins running and are passed to other programs at the end of the job. Their setting can be changed during processing, allowing the program to change the status of these indicators. However, if an external indicator conditions a file, that indicator must be set on before the program is loaded in order to use the file in the program.

Use external indicators to:

- Determine if a file is used for a program
- Condition calculation operations
- Condition output operations
- Indicate the relation of a field to a record
• Provide communication between programs.

**Note:** You can also use SUBR20 to read and to write external indicators. For more information, see Chapter 7, "Using a WORKSTN File."

The two-character entries allowed as external indicators are:

U1 through U8

On the **file description specifications**:

• Columns 71 and 72 can contain an external indicator to condition a file. A file conditioned by an external indicator is used only when the indicator is on. When the indicator is off, the file is treated as though the program reached the end of the file; that is, no records can be read from or written to the file.

On the **input specifications**:

• Columns 63 and 64 can contain an external indicator to tell the program to accept and use data from a particular field only when the external indicator is on.

On the **calculation specifications**:

• Columns 9 through 17 can contain external indicators to condition the operations that should be done for a specific job. If a file is conditioned by an external indicator, any calculations that are done only on that file should be conditioned by the same external indicator.

On the **output specifications**:

• Columns 23 through 31 can contain an external indicator to condition certain output records on external conditions.

If you want to pass information to other programs, you can use external indicators as resulting indicators.

---

**Internal Indicators**

Internal indicators are set by the RPG II program cycle. There are three types:

• First-page indicator
• Last-record indicator
• Matching record indicator.

**First-Page Indicator**

In the first program cycle, the first-page indicator is on during the beginning of the cycle. Heading and detail lines are printed by the first page indicator before the first record is read.

The purpose of the first-page indicator is to condition records that are printed on the first page of a report. These records are usually headings used to identify information found on the page, but they can also be detail lines.

The first-page indicator is an internal indicator that is defined by the RPG program cycle itself. It turns on only for the beginning of the first cycle. It turns off before a record is read and is never used again during the program (see Figure 127).
INTERNAL INDICATORS

First Cycle

START

Perform heading and detail output for which conditions have been met, including 1P output (first cycle only).

First Page Indicator

Turn off control-level indicators L1-L9 and first-page indicator.

Figure 127. RPG Program Cycle for the First-Page Indicator

Notice in Figure 127 that the program does first-page output and other heading and detail output first. This happens in every RPG program. The program writes first-page output and any other heading or detail output for which specified conditions are met before the first record is read. After the first cycle, however, it is easier to think of reading a record as the first step in the cycle.

The only two-character entry allowed as a first-page indicator is:

1P

On the output specifications:

- Columns 23 through 31 can contain a first-page indicator to condition lines that are printed on only the first page.

You can use the first-page indicator only in columns 23 through 31 of heading or detail output lines, not with total or exception output lines. You can use the first-page indicator in an OR relationship with an overflow indicator to allow printing on every page (see Figure 128). You cannot use the first-page indicator in an AND relationship with control-level indicators, to condition output for a WORKSTN file, or to condition calculation operations.
The first-page (1P) indicator is used when headings are to be printed on the first page only.

```
OPRINT H 3 1P
```

The first-page (1P) indicator and an overflow indicator can be used to print headings on every page.

**Last-Record Indicator**

You use the last-record indicator to condition all operations done at the end of your program. These operations usually include calculating totals for all records or writing summary information. When the last-record indicator turns on, the control-level indicators L1 through L9 also turn on. Therefore, all total operations conditioned by L1 through L9 and LR are done. See Figure 129 on page 378 for the specific steps that occur at the end of a job.

The RPG program cycle sets on the last-record indicator when end of file occurs for a primary file. End of file occurs for a primary DISK file when you read past the last record in the file. End of file occurs for a primary WORKSTN file when:

- All display stations are released (by an R in column 16 of the output specifications or by the REL operation code) if the program is not a never-ending program.
- All display stations are released and the operator entered a command to stop the system if the program is a never-ending program.

You must set on the last-record indicator if:

- The program contains no primary file.
- KEYBORD is specified as the device for a primary input file.

If you use any of the L0 through L9 indicators in an OR relationship with a last-record (LR) indicator, the specified operation is done twice when LR is on. One operation is done at total time and the other during last-record (LR) processing.

Once the LR indicator has been set on, it must not be set off.
The only two-character entry allowed as a last-record indicator is:

LR

On the **input specifications**:
- Columns 19 and 20 can contain LR as a record-identifying indicator.

On the **calculation specifications**:
- Columns 7 and 8 must contain LR for all operations done at the end of the job. When the last-record indicator turns on at the end of the job, the other control-level indicators you specified also turn on.
- Columns 9 through 17 can contain LR to condition operations when the last-record indicator turns on during calculations.
- Columns 54 through 59 can contain LR except for the SETOF operation. When the last-record indicator turns on in calculations, the other control-level indicators you specified do not turn on until the beginning of the next cycle.
On the **output specifications**:

- Columns 23 through 31 can contain LR to condition output after all records are processed.

### Matching-Record Indicator

Use the matching-record indicator only when you are processing primary and secondary files. Its purpose is to indicate when fields or records from different files match. The matching-record indicator is set on or off only after total operations are done. Therefore, at detail time, it always signals the matching status of the record just selected for processing; at total time, it reflects the matching status of the previous record.

In processing primary and secondary files, you must specify match fields to compare records from two or more input or update files to determine the record that is selected for processing. You can use one field, many fields, or an entire record to match records. Whenever the contents of the match field from the primary file record are the same as the contents of the match field from a secondary file record, the matching-record (MR) indicator turns on. The matching-record indicator can then be used to condition those operations that are done only when records match.

**Note:** All match fields that have no field-record-relation indicator should be described before those that do.

For more information on processing primary and secondary files, see Chapter 12, “Primary/Secondary/Multifile Processing.”

Figure 130 on page 380 shows the general steps in the RPG program cycle for programs that use more than one input file.
The only two-character entry allowed as a matching-record indicator is:

MR

On the input specifications:
- Columns 61 and 62 must contain the matching-record indicator to tell the program to accept and use data from a particular field when fields or records from different files match.

On the calculation specifications:
- Columns 9 through 17 must contain the matching-record indicator to condition an operation that is done only when matching records are found.

Figure 130. RPG Program Cycle for Matching Records
On the output specifications:
- Columns 23 through 31 must contain the matching-record indicator to tell the program to write a line or field when matching records are found.

CONDITIONING INDICATORS

The following indicators are not set on or off when used as conditioning indicators. You can change the status (on or off) only by defining the indicator to represent a certain condition.

Conditioning indicators cannot be used on the END operation of the CAS group or the IF group.

File-Conditioning Indicators

The purpose of the file-conditioning indicator is to condition a file so that the program uses that file only when the file-conditioning indicator is on. When the file-conditioning indicator is off at the beginning of the program, the file is treated as though the end of the file is reached; in other words, no records can be read from or written to the file.

The two-character entries allowed as file-conditioning indicators are:
- U1 through U8

On the file description specifications:
- Columns 71 and 72 can contain a file-conditioning indicator to determine if a file is used for a job.

Field-Record-Relation Indicators

Field-record-relation indicators are used to associate fields with a particular record type when that record type is one of several in an OR relationship. The field described on the specification line is available for input only if the indicator specified in the field record relation entry is on or if the entry is blank. If the entry is blank, the field is common to all record types defined by the OR relationship.

An indicator that was previously defined in the program can also be used as a field-record-relation indicator. Control fields (specified by a control-level indicator in columns 59 and 60 on the input specifications) and match fields (specified by a match value in columns 61 and 62 on the input specifications) can also be related to a particular record type in an OR relationship by a field-record-relation indicator. Control fields or match fields in the OR relationship that do not have a field-record-relation indicator are used with all record types in the OR relationship.

When two control fields have the same control-level indicator or two match fields have the same match value, a field-record-relation indicator can be assigned to just one of the control fields or match fields. In this case, the field with the field-record-relation indicator is used only when that indicator is on. If none of the field-record-relation indicators are on for that control field or match field, the field without a field-record-relation indicator is used. Control fields and match fields can only use the two-character entries 01 through 99 or H1 through H9 in columns 63 and 64.
FIELD-RECORD-RELATION INDICATORS

The two-character entries allowed as field-record-relation indicators are:

- 01 through 99
- H1 through H9
- MR
- L1 through L9
- U1 through U8

On the **input specifications**:
- Columns 63 and 64 can contain a field-record-relation indicator to associate fields with a particular record type when that record type is one of several in an OR relationship.

**Assigning Field-Record-Relation Indicators**

When assigning field-record-relation indicators in the input specifications, consider the following:

- All fields, including match or control fields, that have no field-record-relation indicator should be described before those that do.
- All fields having the same field-record-relation indicator should be defined on consecutive specification lines for more efficient use of storage. These fields can, however, be entered in any order.
- All portions of a split control field must be assigned the same field-record-relation indicator and must be defined on consecutive specification lines (see Figure 131 on page 383). For more information on split control fields, see “Split Control Fields” on page 370.
- When the field-record-relation indicator is used with control or match fields, the field-record-relation indicator must match a record-identifying indicator for this file, and the match or control fields must be grouped according to the field-record-relation indicator. The field-record-relation indicator for control or match fields can only be 01 through 99 or H1 through H9.
- When any match value (M1 through M9) is specified for a field without a field-record-relation indicator, all match values used must be specified once without a field-record-relation indicator. If all match fields are not common to all records, a dummy match field should be used (see Figure 132 on page 384).
The record identified by the 1 in position 95 has two split control fields:

FLD1A and FLD1B
FLD2A and FLD2B

The record with a 2 in position 95 has three split control fields:

FLD1A and FLD1B
FLD2A and FLD2B
FLD3A, FLD3B, and FLD3C

All portions of the split control field must be assigned the same control-level indicator and all must have the same field-record-relation entry.

Record Identification Code = 1

FLD1A FLD1B FLD2A
FLD1A FLD1B FLD2A

Record Identification Code = 2

FLD1A FLD3D FLD3A FLD3C
FLD2B FLD3B

Record Identification Code = 3

The third record type, identified by the 3 in position 95, also has three split control fields:

FLD1A and FLD1B
FLD2A and FLD2B
FLD3D and FLD3E

Field Identification Code = 2

FLD1A
FLD1B
FLD2A
FLD2B
FLD3A
FLD3B
FLD3C
FLD3D

Figure 131. Field-Record-Relation Indicator (Split Control Field)
Three different record types are found in the input file. All three contain a match field in positions 1 through 10. Two of them have a second match field. Because M1 is found on all record types, it can be specified without an entry in columns 63 and 64. If one match value (M1 through M9) is specified without field-record-relation entries, all match values must be specified once without field-record-relation entries. Because the value M1 is specified without field-record relationship, an M2 value must also be specified once without field-record relationship. The M2 field is not on all record types; so a dummy M2 field must be specified next. The dummy field can be given any unique name, but its specified length must be equal to the length of the true M2 field. The M2 field is then related to the record types on which it is found by field-record-relation entries (lines 06 and 07).

![Figure 132. Dummy Match Fields](image-url)
Level-Zero Indicator

The level-zero indicator is always on and cannot be set off with the SETOF operation code. You need never assign this indicator, but you can use it to condition operations, especially when no control fields have been assigned. When a control break occurs, all operations conditioned by control-level indicators (including the level-zero indicator) are done before those that are not conditioned. If no control field is assigned, but total calculations are done and total output records are written, use the level-zero indicator (L0) to condition those operations (see Figure 133 on page 386).

The only two-character entry allowed as a level-zero indicator is:

L0

On the calculation specifications:

- Columns 7 and 8 can contain the level-zero indicator to specify that the total calculation be done every time.

On the output specifications:

- Columns 23 through 31 can contain the level-zero indicator to define total output done every cycle.

The program (Figure 133 on page 386) shows how total operations can be done even though there is no control field (no L1 through L9 indicators).

The program requires:

- A list of items sold in each district
- A total of all sales for each district
- A grand total of all sales in all districts.

The input records have ITEM and COST fields and a one-position record identification field. The records are grouped in ascending sequence by district. That is, the district 1 records as a group are followed by a blank record, the district 2 records as a group are followed by a blank record, and the district 3 records as a group are followed by a blank record.

No field can serve as a control field because the district number is not on the records. Instead of a control field, the blank record is used to signal a new district. When the blank record is read, indicator 02 turns on. The blank record tells the program that total calculations and total output operations must be done. However, no total operations can be processed unless they are conditioned by some kind of control-level indicator.

Even though L0 is on all the time, it must be used in columns 7 and 8 because some type of control-level indicator must be assigned to all total operations.
**Figure 133 (Part 1 of 2). Use of the Level-Zero (L0) Indicator**

<table>
<thead>
<tr>
<th>Line</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>#1</td>
<td>COST</td>
<td>ADD</td>
<td>DISTOT</td>
<td>DISTOT #2</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CL</td>
<td>#2</td>
<td>DISTOT</td>
<td>ADD</td>
<td>GDSTOT</td>
<td>GDSTOT #2</td>
</tr>
<tr>
<td>6</td>
<td>CL</td>
<td>DISTOT</td>
<td>ADD</td>
<td>GDSTOT</td>
<td>GDSTOT</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Format of the Printed Report

J102  4.50
J202  3.75
K450  2.98
B231  9.08

20.31 *

G10H  92.79
G10K  98.89
A126  4.29

195.97 *

216.28 **

Figure 133 (Part 2 of 2). Use of the Level-Zero (L0) Indicator
Function-Keys Indicators

There are 24 function keys. Each one corresponds to a separate function-key indicator:

<table>
<thead>
<tr>
<th>Function-Key Indicator</th>
<th>Corresponding Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>KA</td>
<td>1</td>
</tr>
<tr>
<td>KB</td>
<td>2</td>
</tr>
<tr>
<td>KC</td>
<td>3</td>
</tr>
<tr>
<td>KD</td>
<td>4</td>
</tr>
<tr>
<td>KE</td>
<td>5</td>
</tr>
<tr>
<td>KF</td>
<td>6</td>
</tr>
<tr>
<td>KG</td>
<td>7</td>
</tr>
<tr>
<td>KH</td>
<td>8</td>
</tr>
<tr>
<td>KI</td>
<td>9</td>
</tr>
<tr>
<td>KJ</td>
<td>10</td>
</tr>
<tr>
<td>KK</td>
<td>11</td>
</tr>
<tr>
<td>KL</td>
<td>12</td>
</tr>
<tr>
<td>KM</td>
<td>13</td>
</tr>
<tr>
<td>KN</td>
<td>14</td>
</tr>
<tr>
<td>KP</td>
<td>15</td>
</tr>
<tr>
<td>KQ</td>
<td>16</td>
</tr>
<tr>
<td>KR</td>
<td>17</td>
</tr>
<tr>
<td>KS</td>
<td>18</td>
</tr>
<tr>
<td>KT</td>
<td>19</td>
</tr>
<tr>
<td>KU</td>
<td>20</td>
</tr>
<tr>
<td>KV</td>
<td>21</td>
</tr>
<tr>
<td>KW</td>
<td>22</td>
</tr>
<tr>
<td>KX</td>
<td>23</td>
</tr>
<tr>
<td>KY</td>
<td>24</td>
</tr>
</tbody>
</table>

Note: The keyboard keys used to enter these indicators may vary, depending on the type of keyboard that you have.

The purpose of function-key indicators is:

- To condition calculation and output operations for a program with a WORKSTN or a KEYBORD file. All 24 function-key indicators are defined for a WORKSTN file.
- To specify the function keys that the person using a display station can press for a SET operation with a KEYBORD file. To tell the person how you used the function keys in your program, you can fill out the template assignment form on the IBM 5251 Display Station Keyboard Template Assignment Sheet and Display Screen Layout Sheet, GX21-9271.
On the **calculation specifications**:

- Columns 9 through 17 can contain function-key indicators that are used with a KEYBORD file or a WORKSTN file to condition calculation operations.
- Columns 54 through 59 can contain a function-key indicator for a SETOF operation.

On the **output specifications**:

- Columns 23 through 31 can contain function-key indicators that are used with a KEYBORD file or WORKSTN file to condition output operations. If a KEYBORD file is used, any function keys entered in these columns must also be coded in columns 54 through 59 of the calculation specifications for a SET operation.

When the program allows the person using the display station to enter data from the keyboard, all function-key indicators are turned off. If the person presses a function key, the corresponding function-key indicator turns on.

---

### Halt Indicators

The purpose of halt indicators is to stop the program when an unacceptable condition exists. You can use halt indicators as record-identifying, field, or resulting indicators. When you use a halt indicator as a record-identifying indicator, a halt is caused by a specific type of record. When you use a halt indicator as a field indicator, a halt is caused by incorrect input data. When you use a halt indicator as a resulting indicator, a halt is caused by incorrect results from calculations.

A halt indicator can turn on at one of four times in the program cycle, depending on how you use it (see Figure 134 on page 390). The program does not halt immediately when a halt indicator turns on. All total and detail operations remaining in the cycle are processed first; then the program halts. This means that the program completes processing the information from the record that caused the halt. Therefore, you must write specifications that bypass calculation and output operations when an error occurs.

When the halt is issued, you have the following options to choose from:

- **0**: Control is returned to the program, and processing continues.
- **2**: End-of-job operations specified by the program are done, tables are dumped, and file labels are cataloged.
- **3**: The job is canceled. Any data created or work done by previous programs in this job is saved. Any records added or updates made to existing files by the current program are saved. Records deleted by the current program no longer exist.

If you want to display your own message text when a halt indicator is on, create and use a message member as described in “Using a Message Member” on page 303.
The two-character entries allowed as halt indicators are:

H1 through H9

On the input specifications:

- Columns 19 and 20 can contain a halt indicator used as a record-identifying indicator.
- Columns 63 and 64 can contain a halt indicator used as a field-record-relation indicator.
- Columns 65 through 70 can contain a halt indicator to prevent a calculation or output operation from being done if the program finds a specified error condition in the input data.
On the **calculation specifications**:  
- Columns 9 through 17 can contain a halt indicator to condition calculation operations.  
- Columns 54 through 59 can contain a halt indicator that is set on or set off as the result of an operation.

On the **output specifications**:  
- Columns 23 through 31 can contain a halt indicator to prevent or condition output operations if an error condition occurs.

---

**Indicators Conditioning Calculations**

Indicators that you use to specify the conditions under which a calculation is done must be defined elsewhere in the program.

You can use the operation codes SETON or SETOF to turn indicators on or off. See “SETON and SETOF Operations,” for more information on these operations. Any indicators that you want turned on or off by the SETON or SETOF operation codes can be specified in any of the three resulting indicator fields (columns 54 through 59 on the calculation specifications). However, you cannot turn on function-key indicators with the SETON operation or turn off the last-record indicator with the SETOF operation. The headings for columns 54 through 59 (high, low, and equal) have no meaning for SETON or SETOF operations.

The two-character entries in columns 9 through 17 of the calculation specifications must be previously defined as one of the following types of indicators:

- Overflow indicators
- Record-identifying indicators
- Control-level indicators
- Field indicators
- Resulting indicators
- External indicators
- Internal indicators.

On the **calculation specifications**:  
- Columns 9 through 17 can contain conditioning indicators that control the conditions under which an operation is done.

From one to three indicators (specified in columns 10 and 11, 13 and 14, and 16 and 17) can be used on each line. If the indicator must be off to condition the operation, place an N before the indicator (in column 9, 12, or 15).

**Using Indicators in AN/OR Lines on the Calculation Specifications**

Use columns 7 and 8 of the calculation specifications to specify that lines of indicators are in an AN/OR relationship. When you use the AN/OR relationship, many lines of indicators can be grouped together to condition an operation. A maximum of seven AN lines, seven OR lines, or seven of any combination of AN and OR lines can condition an operation.

The first line of such a group contains blanks in columns 7 and 8 of the calculation specifications or an L0 through L9, LR, or SR entry if the group of lines is condi-
Indicators specified in columns 9 through 17 of the calculation specifications are in an AND relationship with each other if the indicators are on the same line. The indicators on one line or indicators in grouped lines plus the control-level indicator (if used in columns 7 and 8 of the calculation specifications) must all be exactly as specified before the operation is done (see Figure 136 on page 393).

An indicator that is specified in columns 9 through 17 can also be entered as a resulting indicator on the same line. If the indicator in columns 9 through 17 is on, the calculation is done.
Assume that indicator 25 represents a record type and that a control-level-2 break occurred when record type 25 was read. Both indicators L1 and L2 are on. All operations conditioned by control-level indicators in columns 7 and 8 are processed before operations conditioned by control-level indicators in columns 9 through 17. Therefore, the operation conditioned by L2 is done first. When indicator 25 is on, the operation is done on the first record of the new control group. When indicators L2 and 10 are on, and indicator L3 is not on, the operation is done for all records of the previous control group. The operation conditioned by both indicators L2 and NL3 is done only when a control-level-2 break occurs. These two indicators are used together because this operation is not done when a control-level-3 break occurs, even though indicator L2 is also on.

**Figure 136. Conditioning Operations (Control-Level Indicators)**

**Indicators Conditioning Output**

Indicators that you use to specify the conditions under which an output record or an output field is written must be previously defined in the program.

The two-character entries in columns 23 through 31 of the output specifications must be previously defined as one of the following types of indicators:

- Overflow indicator
- First-page indicator
- Record-identifying indicator
- Control-level indicator
- Field indicator
- Resulting indicator
- Internal indicator
- External indicator.

On the output specifications:

- Columns 23 through 31 can contain a conditioning indicator to specify the conditions under which an output record or an output field is written. When the indicator is to condition an entire output line, enter it on the record line (column...
15 contains a D, H, T, or E). When an indicator is to condition when a field is to be written, enter it on the same line as the field name.

Using Indicators in an AND/OR Relationship on the Output Specifications

Use an AND line if more than three indicators are needed to condition an output operation. Enter AND in columns 14 through 16 on the output specifications for each additional line. The conditions for all indicators in an AND relationship must be satisfied before the output operation is done. Any number of AND lines can be used for an output operation.

Output indicators can also be in an OR relationship. Enter OR in columns 14 and 15 for each OR relationship. If one or the other condition is met, the output operation is done. Any number of OR lines can be used for an output operation.

A maximum of 255 record line groups can be used in an OR relationship under the following conditions (see Figure 138 on page 395):

- Column 15 of the record line contains a D, T, or H
- The first-page indicator or an overflow indicator is used. The first page or overflow indicator may appear on the record line, the OR line, or an AND line.

AND and OR lines can be used to condition entire output lines, but they must not be used to condition fields (see Figure 137 on page 395). However, you can condition an output field with more than three indicators by using the SETON operation in calculations. For example, if indicators 10, 12, 14, 16, and 18 are used to condition an output field named PAY, you can set indicator 20 on in the calculations if indicators 10, 12, and 14 are on. Then condition the output field PAY indicators 20, 16, and 18 on in the output specifications.

The use of any of the L0 through L9 indicators in an OR relationship with an LR indicator can result in the specified operation being done twice when LR is on. One operation is done at total time and the other during LR processing. Figure 138 on page 395 shows how to correctly use the L0 through L9 indicators in an OR relationship.
The detail line is printed if either of two sets of conditions is met.

If indicators 21, 40, 01, and 16 are all on, the line is printed; or if indicators 21 and 40 are both on and indicators 01 and 16 are off, the line is also printed.

```
OTRSACTN D  21 40 01
  AND  16
  OR   21 40N01
  AND  N16
```

A maximum of three indicators can be used to condition a field.

```
MR L1 02BALNC  70
```

---

**Figure 137. Output Indicators in AND and OR Lines**

---

**Figure 138. Correct Use of Control-Level Indicators (L0-L9) in OR Relationship**
Chapter 14. Using Arrays and Tables

Both arrays and tables are systematic arrangements of data items, called elements. Each element in an array or table has the same field length, the same data type (alphameric or numeric), and the same number of decimal positions if numeric. You can use an array or a table for the same purposes. Both arrays and tables are described on the extension specifications.

However, arrays and tables differ in two important ways:

- When they can be loaded
- How they can be processed.

When Arrays and Tables Can Be Loaded

**Loading** an array or table means reading it into main storage so that the program can process it.

Arrays can be loaded while the source program is being compiled, before the program is run, or while the program is running.

Tables can be loaded while the source program is being compiled or before the program is run.

How Arrays and Tables Can Be Processed

A program can process an array in one of two ways:

- It can process an entire array at one time. That is, the program does the same operation on every element in the array.

- It can process specific array elements that you refer to by their position relative to other elements. To do this, you must provide an index to specific elements in the array.

The only way that a program can process a table is to process the operation on only one element. The operation must be specified separately for each additional element that you want to process.

Kinds of Arrays and Tables

- **Compile-time arrays and tables** are loaded with the source program and become a permanent part of the program object. The initial content of a compile-time array or table can be changed in the program, or you can recompile the source program with new array or table data (see “Changing the Contents of Arrays and Tables” on page 414).

- **Prerun-time arrays and tables** are loaded with the program object before you run the program, that is, before any input records are read, calculations are done, or output records are written.

- **Run-time arrays** are loaded or created by input or calculation specifications. The arrays are loaded while the program is running, that is, they are read in as input data or created during calculations in the program. A run-time array is also described on the extension specifications. Tables cannot be specified for run-time load.
Related arrays and tables are two arrays or two tables that are read and stored together and that are defined on the same extension specification (see example on line 01 of the second part of Figure 139). Each element in the second array or table gives additional information about its corresponding element in the first array or table. An array can be related to another array, or a table can be related to another table; however, an array cannot be related to a table, or vice versa.

For example, Figure 139 on page 399 shows related arrays ARR1 and ARR2. An element in array ARR1 provides a part number, and the corresponding element in array ARR2 provides the cost for that part. Although all elements within one table or array must have the same characteristics, corresponding elements of related arrays or tables can have different characteristics. Therefore, in Figure 139 on page 399, all elements in array ARR1 are alphameric, and all elements in array ARR2 are numeric.

Related arrays and tables should have the same number of elements. If the arrays or tables do not have the same number of elements, the program might find the desired element in one array or table but find no corresponding element in the related array or table. In this case, undesirable results can occur.

<table>
<thead>
<tr>
<th>ARR1 (Part Number)</th>
<th>ARR2 (Unit Cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>345126</td>
<td>00373</td>
</tr>
<tr>
<td>38A473</td>
<td>00498</td>
</tr>
<tr>
<td>39K143</td>
<td>01297</td>
</tr>
<tr>
<td>40B125</td>
<td>00093</td>
</tr>
<tr>
<td>41C023</td>
<td>03998</td>
</tr>
<tr>
<td>42D893</td>
<td>00087</td>
</tr>
<tr>
<td>43K832</td>
<td>00349</td>
</tr>
<tr>
<td>44H111</td>
<td>00679</td>
</tr>
<tr>
<td>45P673</td>
<td>00898</td>
</tr>
<tr>
<td>46C732</td>
<td>47587</td>
</tr>
</tbody>
</table>

Arrays ARR1 and ARR2 can be described as two separate arrays or as two related arrays.

The following extension specifications show how to describe ARR1 and ARR2 as two separate arrays. The entries in columns 33 through 35 are required for records that are in the source program.
Creating Input Records for Arrays or Tables

Input records for arrays or tables must be formatted according to certain rules:

- The first array or table element for each input record must begin in position 1.
- An entire record need not be filled with array or table elements. If it is not, blanks or comments can be included after the elements (see Figure 140 on page 400). The unused space in numeric arrays or tables is filled with zeros; the unused space in alphabetic arrays or tables is filled with blanks.
DEFINING ARRAYS AND TABLES

- Each input record, except the last, must contain the same number of elements. In the last record, unused space must be blank. You can include comments after the blank space. That is, comments in the last record must begin in the same position as comments in preceding records (see Figure 140 on page 400).

- Each element must be contained entirely on one input record; an element cannot be split between two records. Therefore, the length of a single element is limited to the maximum record length for the input device. If you use related arrays or tables and describe them in alternating format, corresponding elements must be on the same input record; the combined length of corresponding elements cannot exceed the maximum record length for the device.

- Related arrays or tables can be described separately or in alternating format. Alternating format means that elements of one array or table alternate with elements of the related array or table on the input record (see Figure 139 on page 399).

- The total number of array names and table names used in a program cannot exceed 200.

```
* *
1 2 3 4 5 1 2 3 4 5 Comments can be
1 2 3 4 5 1 2 3 4 5 anywhere out here
1 2 3 4 5 1 2 3 4 5 or here
1 2 3 4 5 or here (that is, after the last entry for the longest record).
* *
1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5
1 2 3 4 5
1 2 3 4 5 If comments begin here, the compiler cannot tell if you intend them as comments or if you provided too much data for the table/array. Therefore, it prints a warning message.
```

* Figure 140. Array or Table Input Record with Comments

Each of the two tables/arrays contains seven entries, each entry is five positions long, with two entries per record. The last record contains only one entry. The remaining 5 positions in the last record should be left blank, because using these positions for comments prints the warning message, QRG8039 Remaining positions of last Compile-Time table/array source record not blank. Defaults to blanks, during compilation. Therefore comments should begin after the last entry position for the longest record; that is, (the number of entries per record multiplied by the number of positions per entry) plus 1.

---

**Defining Arrays and Tables**

All arrays and tables must be defined on extension specifications. Each extension specification defines one set of array or table input records.

If only one array or table is being defined, use columns 11 through 45.

If alternating arrays or tables are being defined, use columns 11 through 57. For compile-time and prerun-time arrays or tables, the array or table named in columns 46 through 51 is entered in alternating format with the array or table named in columns 27 through 32.
If compile-time arrays or tables are being defined, columns 11 through 26 must be blank.

If prerun-time arrays or tables are being defined, entries are required in columns 11 through 18 and in columns 27 through 45.

If run-time arrays are being defined, columns 11 through 26 must be blank.

If the array or table being defined is written to a file at the end of the job, enter the name of the file in columns 19 through 26. Run-time arrays cannot be written to a file at end of job.

If you are defining a table, the name you assign in columns 27 through 32 must begin with TAB. If you are defining an array, it must not begin with TAB.

Arrays and tables can be defined in any sequence on the extension specifications. The sequence in which they are defined determines the order in which they are loaded at the start of the program.

Figure 141 on page 402 shows the extension specifications required for the three types of arrays.

- Line 01 specifies two compile-time arrays, AR1 and AR2, in alternating format. Each array has three elements per record and a total of eight elements in the array. Each element is 12 positions long, including four decimal positions, although the length of elements and the number of decimal positions in AR2 do not have to be the same as those in AR1.

- Line 06 specifies a prerun-time array, AR3, read from file DISKIN. AR3 has 12 elements per record and a total of 250 elements. Each element is 5 positions long. Decimal positions are not specified, so the elements are alphameric data. The elements are arranged in ascending sequence.

- Line 09 specifies a run-time array, AR4, which contains 10 elements. Each element is 10 positions long. Zero decimal positions are specified, so the elements are numeric data.

For all arrays and tables except run-time arrays, columns 19 through 26 can also contain the name of a file to which the array or table is written at the end of the program, and columns 46 through 57 can also define an array or table that is entered in alternating format with the array or table named in columns 27 through 32.
Arrays can be loaded at compilation time, prerun-time, or run-time. Tables can be loaded at compilation time or at prerun-time.

Loading Compile-Time Arrays and Tables

A compile-time array or table is loaded at compilation time with the data supplied at the end of the source program. Rules for loading arrays and tables at compilation time are as follows:

- A compile-time array or table must have entries in columns 33 through 35 of the extension specifications and must not have entries in columns 11 through 18 of the extension specifications.
- Compile-time array or table data must be entered at the end of the source program, after all records for translating files and for changing the collating sequence of characters (see Chapter 17, "Changing the Hexadecimal Value of Characters").
- A record with ** in positions 1 through 3 must precede the data for each compile-time array or table (see Figure 142).
- Compile-time arrays or tables must be in zoned-decimal or alphameric format.
- For compile-time arrays and tables, the maximum length of an alphameric element is 96 because the maximum length of a record in the source program is 96 characters. If a compile-time array or table is not large enough to hold all the data, warning message RPG8839 Remaining positions of last Compile-Time table/array source record not blank. Defaults to blanks., is issued. The extra data is ignored.
ARRA and ARRB are located in the source member with the source program.

*Figure 142. Arrangement on Disk of the Source Program and Compile-Time Array Data*

**Loading Prerun-Time Arrays and Tables**

A prerun-time array or table is loaded by the load program from an input file on disk just before the program runs. The file must be described on the file description specifications as an input table file (IT in columns 15 and 16). The OCL FILE statement must also be present for the input file. If two or more arrays or tables are to be loaded, they must be loaded from different disk files, except when the arrays or tables are specified in alternating format. The filename must be specified in columns 11 through 18 of the extension specification that defines the prerun-time array or table.

Most of the rules that apply to compile-time arrays and tables also apply to prerun-time arrays and tables except for the following:

- A prerun-time array or table must have an entry in columns 11 through 18 and in columns 33 through 35 of the extension specifications.
- The file description specification for the file containing the data for a prerun-time array or table must have an I in column 15, a T in column 16, and an E in column 39.
- For numeric arrays and tables loaded at prerun-time, the data in a DISK file can be in zoned-decimal, packed-decimal, or binary format.
- Prerun-time array or table records must be in a sequential file, and each record must have the same length.
- If a prerun-time array or table is not large enough to hold all the data, error message RPG9017 [Program name] has object table data exceeding the table length, is displayed. In response to that message, you can ignore the extra data, end the job step, or cancel the job.
LOADING ARRAYS AND TABLES

Loading Run-Time Arrays

To load an array from information in input records, describe that information in the input specifications. The format of the specifications depends on the array information: if it is contained in one record or in more than one record. The input specifications can describe any type of array (compile-time, prerun-time, or run-time). When you use input specifications to fill an array with data, the program must read complete data elements.

You can also use an arithmetic or move operation in the calculation specifications to load a run-time array, either the entire array at one time or one element at a time.

Run-time arrays are not sequence checked. If you use the SORTA operation, the array is sorted into the sequence (ascending or descending) specified in column 45 of the extension specifications for that array. If no sequence is specified, the array is sorted into ascending sequence. If you use the LOKUP operation with an indicator in columns 54 and 55 or in columns 56 and 57 of the calculation specifications, you must also specify a sequence in column 45 of the extension specifications.

To load an array from a single input record, code entries in the unshaded columns of the input specifications as shown below:

<table>
<thead>
<tr>
<th>Field Indicators</th>
<th>Sequence</th>
<th>Control Level (L1-L9)</th>
<th>Field Record Relation</th>
<th>Matching Fields or Chaining Fields</th>
<th>Plus Minus</th>
<th>Zero or Blank</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>Number (1/N),E</td>
<td>Option (O),U,S</td>
<td>Stacker Select (P/B/L/R)</td>
<td>Record Identification Codes</td>
<td>Data Structure Name</td>
<td>Data Structure</td>
<td>Occurs nT i m e s</td>
</tr>
</tbody>
</table>

Column 43 must contain P (packed decimal), B (binary), or blank (zoned decimal) to indicate the format of the array data.

Columns 44 through 47 must contain the starting position of either the entire array with consecutive elements or an individual element in the array.

Columns 48 through 51 must contain the ending position of either the entire array with consecutive elements or an individual element in the array.

Columns 53 through 58 must contain the name of the array (the same name used on the extension specifications) or the name of an individual array element (array name plus comma and index).

Columns 63 and 64 can contain an indicator to indicate the relation of a field to the record.
Array Information in One Record

If the array information is contained in one record, the information can occupy consecutive positions in the record or it can be scattered throughout the record. If an array is contained in a data structure, all elements in the array are consecutive.

If the array elements are consecutive on the input record, you can load the array with a single input specification. Figure 143 shows the extension and input specifications for loading an array, INPARR, that contains six elements (12 characters each) from a single record in the file ARRFILE.

```
**.1 2 3 4 5 6 7.
E...FromfileTofile++ArrnamEntParrLenPDSAlnamLenPDSComments+++++++
E  INPARR    6 12
**.1 2 3 4 5 6 7.
I\filenameSqNORiPos1NCCPos2NCCPos3NCC
IARRFILE AA O1
I..............................PFromTo++DField+L1M1FrPoNeEq...
I 1    72 INPARR
```

Figure 143. Defining a Run-Time Array with Consecutive Items

If the array elements are scattered throughout the record, they can be defined and loaded one at a time, with each element described on a separate specification line. Figure 144 on page 406 shows the extension and input specifications for loading an array, ARRX, that contains six elements (12 characters each) from a single record in the file ARRFILE. A blank separates each element from the others.
LOADING ARRAYS AND TABLES

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++ArrnamEntParrLenPDSAItnamLenPDSComments+++++++*
E ARRX 6 12
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC
-------------------------------
IARRFILE AA 01
I.................................PFromTo++DField+L1M1FrPoNeEq...*
I           12 ARRX,1
I           14 25 ARRX,2
I           27 38 ARRX,3
I           40 51 ARRX,4
I           53 64 ARRX,5
I           66 77 ARRX,6

Figure 144. Defining a Run-Time Array with Scattered Items

Array Information in More than One Record
If the array information is contained in two or more records, you can use any one of
several methods to load the array. The method to use depends primarily on the
size of the array and on if the array elements are consecutive in the input records.

Figure 145 on page 407 shows an array that is loaded from more than one input
record. Records identified by a 1 or 3 in column 1 contain six array elements (four
characters in each element). Records identified by a 2 in column 1 are also in the
same input file, but they do not contain array elements. The program processes
one record at a time; therefore, the program cannot process the entire array until it
reads every record containing the array elements and moves the elements into the
array fields. Therefore, be sure that your program reads the entire array before it
does any calculation or output operations using the array.
Using an Array Name and Index

Your program can refer to an array as a whole or can refer to an individual element in an array. To refer to the entire array, use the array name alone. To refer to a single element of the array, use the array name plus an index. To do so, add a comma and an index after the array name. The index can be either the actual number of the element used (for example, AR, 1) or the name of a field containing the number of the element used (for example, AR, IND).

Remember the following rules when specifying an array name and index:

- The array name must not be the same as the name of a field, data structure, table, another array, or index in your program.
- The array name can be from one to six characters long.
- The first three characters of the array name cannot be TAB.
- The array name plus comma and index can be from three to six characters long. An array name plus comma and index should not be longer than 6 characters because the field name on the output specifications and the result field on the calculation specifications contains only six positions. However, if the array name plus comma and index are specified only in factor 1 or factor 2 of the calculation specifications, the array name plus comma and index can be up to 10 characters long.
- The index can be a numeric field with zero decimal positions or a numeric constant with no plus or minus sign.
The value in the index must not be zero, negative, or more than the number of elements in the array.

The following are examples of valid and invalid array names:

<table>
<thead>
<tr>
<th>Array Names and Indexes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Valid</strong></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>AR,1</td>
<td>This is the first element of array AR.</td>
</tr>
<tr>
<td>X,YY2</td>
<td>YY2 is a field name containing the index value.</td>
</tr>
<tr>
<td><strong>Invalid</strong></td>
<td></td>
</tr>
<tr>
<td>BALANCE</td>
<td>The array name has more than six characters.</td>
</tr>
<tr>
<td>6TOTAL</td>
<td>The first character must be alphabetic.</td>
</tr>
<tr>
<td>TOTAL-</td>
<td>Characters after the first must be alphanumeric or numeric, not special characters.</td>
</tr>
<tr>
<td>CR TOT</td>
<td>The name cannot contain a blank.</td>
</tr>
<tr>
<td>A1,A1</td>
<td>The index cannot be the same as the array name.</td>
</tr>
<tr>
<td>BAL,XX1</td>
<td>The array name, including the comma and index, cannot contain more than six characters. This name is valid only for factor 1 or factor 2 of the calculation specifications.</td>
</tr>
<tr>
<td>AR,+1</td>
<td>The index cannot have a sign.</td>
</tr>
<tr>
<td>AR,0</td>
<td>The index cannot be zero.</td>
</tr>
</tbody>
</table>

*Figure 146. Examples of Valid and Invalid Array Names*

**Searching Arrays and Tables**

To search for a particular element in an array or table, use the LOKUP operation with factor 1, factor 2, and at least one resulting indicator (high, low, or equal) specified. The result field can be specified for a table; it must be blank for an array. Searching an array or table is a useful way to find a sequence of characters or multiple occurrences of a character in a record. For example, you can find all the blanks in a record by defining the record as an array with one-character elements and searching for a blank.
Resulting indicators specify the type of search and reflect the result of the search in the following way:

- A resulting indicator in the equal columns (58 and 59) instructs the program to search the array or table for an element equal to factor 1. The first equal entry found turns on the resulting indicator.
- A resulting indicator in the low columns (56 and 57) instructs the program to search the array or table for an element that is nearest in sequence to, yet lower than, factor 1. The first such element found turns on the indicator.
- A resulting indicator in the high columns (54 and 55) instructs the program to search the array or table for an element that is nearest in sequence to, yet higher than, factor 1. The first such element found turns on the indicator.

At least one resulting indicator must be used, but no more than two can be used (equal and low, or equal and high). If two resulting indicators are used, the program tries to find an equal element before it tries to find the nearest lower or nearest higher element. If resulting indicators are used in both the high and low columns, the indicator in the low columns is ignored.

When you use the LOKUP operation, remember:

- Conditioning indicators can be specified in columns 7 through 17.
- Factor 1 and each array or table element must have the same length and same format (alphameric or numeric).
- The program can search for high, low, high and equal, or low and equal only if the array or table is in sequence. The sequence must be indicated in column 45 of the extension specifications.
- The resulting indicator turns off if the search is not successful.

Searching an Array

To search an array that does not have an index, use a LOKUP operation and specify as factor 1 the data for which you want to find a match in the array searched. Factor 1 can be a constant, a field name, an array element, or a table name. In factor 2, specify the name of the array searched. The search starts at the first element in the array. Leave the result field blank.

To search an array that has an index, you can begin the search at a particular element in the array. Code the same entries for the LOKUP operation as you would to search an array without an index. However, in factor 2, enter the name of the array searched plus a comma and the index for the element at which the search is to begin. If the index is a variable, the index is set to the number of the array element found if the search is successful. If the search is unsuccessful, the index is set to 1.

Figure 147 on page 410 shows an example of a LOKUP for an array with and without an index.
**SEARCHING ARRAYS AND TABLES**

```
E..., FromfileTofile++ArrnamEntParrLenPDSAltnamLenPDSComments++++++
E*
E* MANNOS, a 2100-element array of employee numbers, is read in
E* at run time from the file ARRFILE with 10 six-position elements
E* per record; the array elements are in ascending order.
E*
E* ARRFILE MANNOS 102100 6 0A
*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C 100336 LOKUPMANNOS 20
C 20 GOTO NEXT
C Z-ADD1 INX 40
C 100336 LOKUPMANNOS,INX 20
C N20 GOTO END
C NEXT TAG
C* "
C* "
C* "
C* Calculation Operations
C* "
C* "
C* "
```

*Figure 147. LOKUP Operation for Arrays with and without an Index*

The first calculation specification is a LOKUP of array MANNOS to find the element nearest to, but higher in sequence than, the constant 100336. If this element is found in the array, indicator 20 turns on and the GOTO in columns 28 through 32 is processed. Indicator 20 indicates only if the searched-for element exists in the array.

The specification in columns 18 through 42 shows essentially the same LOKUP operation. Indicator 20 turns on when the first element higher in sequence than 100336 is found. However, in this LOKUP operation, the array MANNOS is indexed by the field INX. This index field was set to 1 column 33, so the LOKUP begins at the first element of MANNOS. If the searched-for element is found, the number of this element (not its contents) is placed in the field INX. In this way, the actual element that satisfied the LOKUP can be used in subsequent calculation operations, as in columns 28 through 52. If no element was found to satisfy the LOKUP, the field INX is reset to 1.
Searching One Table

To search a single table, use the LOKUP operation with factor 1, factor 2, and at least one resulting indicator specified. The result field can be specified for a table (it must be blank for an array).

If the search finds a table element that satisfies the resulting indicator, the program places a copy of that table element in a special area of main storage. Each time a search is successful, the program places the newly found table element in this area, replacing the element that was in the area. If a search is not successful, the contents of the area remain the same as they were before the unsuccessful search. Before the first successful search, the area contains the first element in the table.

If you use a table name as factor 1, the table name actually refers to the table item found in the last successful search. Therefore, the last item found becomes the data for which you are searching in the current search.

Searching Related Tables

When you use the LOKUP operation to search related tables, the program actually searches only one table (see Figure 148 on page 412). If the search is successful, the corresponding elements from both tables are placed in their respective storage areas.

Note: As used here, the phrase related tables means any two tables in the program that use related data, not necessarily tables that are defined as related on the extension specifications.

Factor 1 of the LOKUP operation must contain the data for which you want to search, and factor 2 must contain the name of the table searched. The result field must contain the name of the related table. A resulting indicator must be specified.

The two tables should have the same number of elements. If the table that is searched contains more elements than the second table, the program might find the desired element in the first table but find no corresponding element in the second table. In this case, undesirable results can occur.
Related tables TABEMP and TABPAY are read into storage. Assume that an input record is read with 443 in the EMPNUM field. Then the program searches the table TABEMP for an element equal to 443. When the correct entry is found, the table item 443 is moved into the special storage area for TABEMP. At the same time, the corresponding item 268 is moved into the special storage area for TABPAY. The contents of the storage areas can now be specified in subsequent calculation operations by the appropriate table name. The coding needed to do the LOKUP operation also shows how to specify the contents of the special storage area after a successful LOKUP operation.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments++++++*
```

C* The following operation searches TABEMP for an entry that is equal to the contents of the field named EMPNUM. If the correct entry is found in TABEMP, 09 turns on and the TABEMP entry and its related entry in TABPAY are moved into their separate storage areas.

```
C EMPNUM LOKUPTABEMP TABPAY 09
```

C* The following operation multiplies the contents of the field named HRSWKD by the contents of the special storage area for TABPAY. The special storage area for TABPAY contains the results of the last successful LOKUP operation involving TABPAY.

```
C 09 HRSWKD MULT TABPAY AMT 62H
```

*Figure 148. LOKUP Operation for Related Tables*
Specifying Arrays

You can specify arrays in input, output, or calculation specifications. You can specify individual elements or the array as a whole.

To specify an entire array, use only the array name, which can be used as factor 1, factor 2, or the result field. You can use the following operations with an array name: ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, SQRT, MOVE, MOVEL, MOVEA, MLLZO, MLHZO, MHLZO, MHHZO, DEBUG, XFOOT, SORTA, and LOKUP.

To specify an individual array element, use the array name plus a comma and an index. Process individual elements like fields. Remember, if you use an array element as a result field, the array name with the comma and index cannot exceed six characters. Several operations can be used with an individual array element, but not with an entire array. These operations are COMP, TESTZ, TESTB, BITON, BITOF, KEY, SET, and MVR, as well as IF/ELSE, CASxx, DOUxx, and DOWxx.

When specified with an array name, certain operations are repeated for each element in the array. These operations are ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, SQRT, MOVE, MOVEL, MLLZO, MLHZO, MHLZO, and MHHZO. The following rules apply when these operations are specified with an array name:

- If factor 1, factor 2, and the result field are arrays with the same number of elements, the operation uses the first element from every array, then the second element from every array, and so on until all elements in the arrays are processed.

- If factor 1, factor 2, and the result field are arrays that do not have the same number of entries, the operation ends when the last element of the array with the fewest elements has been processed.

- When one of the factors is a field, a constant, or a figurative constant, and the other factor and the result field are arrays, the operation is processed once for every element in the shorter array. The same field, constant, or figurative constant is used in all of the operations.

- The result field must always be an array.

- Resulting indicators (columns 54 through 59) cannot be used because of the number of operations being processed.

- If an operation code uses factor 2 only (such as Z-ADD, Z-SUB, or SQRT) and the result field is an array, the operation is processed once for each element in the array. The same field, constant, or figurative constant is used in all of the operations.
Changing the Contents of Arrays and Tables

Changing the Contents Temporarily

You can change the contents of an array or table in a program, and they remain changed for the duration of the program. However, the next time the program is run, the array or table contains the original contents.

One way to change the contents of an array or table temporarily is to use the array or table name as the result field in a MOVE operation. Figure 149 shows an example of changing the contents of related tables.

Figure 149. Changing Related Tables by Using MOVE Operations

```
[*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcedesFactor2+++ResultLenDHHiLoEqComments+++++++S
C     25       LOKUPTABFIL     TABLIT        10 10 = FOUND
C     10       MOVE 500       TABLIT
C     10       MOVE 30        TABFIL
```

The item in TABFIL that contains 25 is changed to 30. The corresponding item in TABLIT is changed to 500. The search word is the constant 25. When a match is found in the table TABFIL, the item from TABFIL and its corresponding item in TABLIT are placed in their respective storage areas. The number 500 is then moved into the storage area for TABLIT; the number 30 is moved into the storage area for TABFIL. The contents of the appropriate original table entry are now changed to agree with the new entry in the special storage areas.

A second way to change the contents of an array temporarily is to use the SORTA operation. This method cannot be used with tables.

A third way is to use the array or table name as the result field of a calculation. If you use an element in that array as factor 1 or factor 2 in that calculation, your program will use the new value of the element in later calculations.

For example, suppose that you have two numeric arrays with the following values:

```
ARR1,1 = 2  ARR2,1 = 2
ARR1,2 = 4  ARR2,2 = 8
ARR1,3 = 6  ARR2,3 = 1
```
Now suppose that you code the following calculation specification:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C ARR1 ADD ARR2,2 ARR2
```

This operation adds the three elements in ARR1, one at a time, to the second element in ARR2 and places the result in ARR2. Here is what happens as the three elements are added:

1. The value of ARR1,1 is added to the value of ARR2,2. That is, 2 is added to 8. The result, 10, is placed in ARR2,1. Therefore, the value of ARR2,1 changes from 2 to 10.

2. The value of ARR1,2 is added to the value of ARR2,2. That is, 4 is added to 8. The result, 12, is placed in ARR2,2. Therefore, the value of ARR2,2 changes from 8 to 12.

3. The value of ARR1,3 is added to the value of ARR2,2. That is, 6 is added to 12 (the new value of ARR2,2), not to 8 (the old value of ARR2,2). The result, 18, is placed in ARR2,3. Therefore, the value of ARR2,3 changes from 1 to 18.

### Changing the Contents Permanently

One way to change the contents of an array or table permanently is to change the input records for the array or table.

A second way to change an array or table permanently is to use one of the methods to change the array or table temporarily, and then to write the array or table to an output file at the end of the program. To do this, define an output file on the file description specifications, and code the name of that output file in columns 19 through 26 of the extension specifications.

### Adding Entries to Arrays and Tables

You can add entries to a short array or table (one in which not all elements are filled) before running the program or while it is running. The simplest way to add entries is to code additional entries on the input records before running the program. While the program is running, you can also add entries that are created by calculation operations or read from an input record.

Figure 150 on page 416 shows an example of adding entries to arrays by using the LOKUP and MOVE operations. These entries are added only temporarily unless the array is written to an output file that is used as input for a prerun-time array the next time the program is run.
The LOKUP operation is conditioned by indicator 01. Indicator 01 is set on when a record containing information in the fields NEWA and NEWB is read. These fields are moved to the arrays ARRA and ARRB, respectively. To get the entry in the correct place in the array, a search is made to find the first empty array element. Unfilled elements in arrays are filled with zeros. Therefore, the value searched for is 000. When the first 000 entry is found, indicator 35 is set on, and the NEWA and NEWB fields are moved into the array elements ARRA,X and ARRB,X. These new entries become part of arrays ARRA and ARRB.

Writing Arrays and Tables

You can write entire arrays and tables to an output file at total time in the RPG program cycle when the last-record indicator is on. To indicate that an entire array or table is to be written, specify the name of the output file in columns 19 through 26 of the extension specifications.
To write an array to an output record by using output specifications, describe the array in the unshaded columns of the output specifications as shown below:

- Columns 23 through 31 can contain output indicators to condition the writing of the array or table.
- Columns 32 through 37 must contain the array name used on the extension specifications.
- Columns 40 through 43 must contain the end position for the last element of the array. If you use an edit code, the end position must account for the skipped positions required by the edit code (see “Editing Arrays”).
- Entries in columns 38 (edit code), 39 (blank after), and 44 (zoned-decimal, packed-decimal, or binary format) apply to each element in the array.
- Columns 45 through 70 can contain an edit word, which applies to each element in the array.

If an output record is to contain only certain elements from a table or array, describe the elements in the same way as normal fields, using either an array name with an index or a table name.

**Editing Arrays**

In column 38 of the output specifications, you can specify an edit code for an entire array or for individual elements in an array.

If you specify an edit code for an entire array, all elements of the array are edited. The program skips two positions before each element in the array. The end position specified in columns 40 through 43 must account for these skipped positions (two times the number of elements). If you are overlaying data in a record, these skipped positions are not blanked out.

If different editing is required for various elements, specify them individually.

If you specify an edit word in columns 45 through 70 of the output specifications, two positions are not skipped before each element. The edit word must contain all the blanks that you want inserted. To include a blank in an edit word, use an ampersand (&) in the edit word to represent a blank.
Examples of Using Arrays

Figure 151 through Figure 157 show examples of the following ways to use arrays:

- Creating an array by using input fields as indexes (Figure 151)
- Creating an array by using fixed indexes (Figure 152 on page 419)
- Calculating totals without using arrays (Figure 153 on page 420)
- Calculating totals by using arrays (Figure 154 on page 422)
- Formatting output fields by using arrays (Figure 155 on page 423)
- Printing one array element per line (Figure 156 on page 425)
- Printing more than one array element per line (Figure 157 on page 426).

Figure 151. Creating an Array by Using Input Fields as Indexes

Figure 151 illustrates a method of loading an array using fields in input records as indexes. The array has 12 elements; each element is five positions long. The array could be defined with any number of elements (to a maximum of 99) without additional input specifications. To build an array using field indexes, assign different values to fields X1 through X10 on each input record type 03 and to fields X1 and X2 on each input record type 04. Succeeding type 03 records can then load 10 additional elements in array AR, up to the maximum defined in the array; each type 04 record can load two additional elements.
EXAMPLES OF USING ARRAYS

Blanks and other fields can appear on the input records because the array elements and their indexes are identified by the From and To entries.

To set up the array in this manner requires:

- A minimum of coding
- No calculations.

However, extra work is required to set up the indexing scheme for the input records.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++ArrnamEntParrLenPDSAltnameLenPDSComments++++++++++
E AR1 30 5
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC..........................*
IFILE1 AA 03 100 C1
I.................................PFromTo++DField+L1M1FrPoNeEq...*
I 1 90 AR1
I BB 04 100 C2
I 1 5 AR1,19
I 6 10 AR1,20
I* "
I* "
I* "
I* More Array Elements
I* "
I* "
I* "
```

*Figure 152. Creating an Array by Using Fixed Indexes*

Figure 152 shows how 18 five-character elements of array AR1 are loaded with only two specification lines. On succeeding input specifications, the remaining elements of AR1 are loaded one after another until the array is full. Each additional element is coded on a separate line. Each new record requires a separate means of identification. For example, if another 03 record followed the first, the fields on the second record would overlay the fields read in from the first record. This method works well for small arrays.
**EXAMPLES OF USING ARRAYS**

The specifications in this figure tabulate three levels of totals. As they are read from input records, the fields FIELDA, FIELDB, FIELDc, and FIELDD are added to the first-level totals L1A, L1B, L1C, and L1D. These first-level totals are added at the time of an L1 control break to totals L2A, L2B, L2C, and L2D. Similarly, at an L2 control break, the second-level totals are added to third-level totals L3A, L3B, L3C, and L3D. In addition, as control breaks occur, L1, L2, and L3 total output is processed; and total fields are set to zeros after they are written to the output device. Figure 14-17 on page 14-26 shows the same tabulations processed by using arrays.

<table>
<thead>
<tr>
<th></th>
<th>FIELD</th>
<th>ADD</th>
<th>L1</th>
<th>L1 Total</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIELD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FIELDA</td>
<td>ADD</td>
<td>L1A</td>
<td>62</td>
<td>ADD TO L1 TOTALS</td>
</tr>
<tr>
<td>C</td>
<td>FIELDb</td>
<td>ADD</td>
<td>L1B</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FIELDc</td>
<td>ADD</td>
<td>L1C</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FIELDd</td>
<td>ADD</td>
<td>L1D</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL1</td>
<td>L1A</td>
<td>ADD</td>
<td>L2A</td>
<td>62</td>
<td>ADD TO L2 TOTALS</td>
</tr>
<tr>
<td>CL1</td>
<td>L1B</td>
<td>ADD</td>
<td>L2B</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL1</td>
<td>L1C</td>
<td>ADD</td>
<td>L2C</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL1</td>
<td>L1D</td>
<td>ADD</td>
<td>L2D</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL2</td>
<td>L2A</td>
<td>ADD</td>
<td>L3A</td>
<td>62</td>
<td>ADD TO L3 TOTALS</td>
</tr>
<tr>
<td>CL2</td>
<td>L2B</td>
<td>ADD</td>
<td>L3B</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL2</td>
<td>L2C</td>
<td>ADD</td>
<td>L3C</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CL2</td>
<td>L2D</td>
<td>ADD</td>
<td>L3D</td>
<td>62</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 153 (Part 1 of 2). Calculating Totals without Using Arrays*
EXAMPLES OF USING ARRAYS

Figure 153 (Part 2 of 2). Calculating Totals without Using Arrays
EXAMPLES OF USING ARRAYS

This figure is similar to Figure 14-16 (Part 1 of 2) except that the three levels of totals are tabulated with arrays. Note the reduction in the coding required to specify the functions. For example, the L1 control break on this calculation specifications, fills the same function as the L1 control break on the calculation specifications shown in Figure 14-16 (Part 1 of 2). Similarly, the output specifications are reduced from 15 lines to 6. The method using arrays results in only two positions between array elements.

```
C      FIELD   ADD SL1,1 SL1,1 ADD FOR L1 TOTL
C      FIELD   ADD SL1,2 SL1,2
C      FIELD   ADD SL1,3 SL1,3
C      FIELD   ADD SL1,4 SL1,4
CL1     SL1  ADD SL2 SL2 ADD FOR L2 TOTL
CL2     SL2  ADD SL3 SL3 ADD FOR L3 TOTL
```

Figure 154. Calculating Totals by Using Arrays
EXAMPLES OF USING ARRAYS

Figure 155 (Part 1 of 2). Formatting Output Fields by Using Arrays
This figure illustrates the use of three arrays to format field output. The arrays are defined as follows:

### Arrays by Name

<table>
<thead>
<tr>
<th>Array Name</th>
<th>Number of Elements</th>
<th>Element Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARA</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>ARB</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>ARC</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Array ARA is contained in the input records with record identifying indicator O1, ARB in the records with record identifying indicator O2, and ARC in both types of records. Array ARC and the first element of array ARA are included together in an output record as are array ARC and an element (identified by field X1) of array ARB. Every element in array ARC is edited according to the edit word '0b.bb&CR' (b = blank).

The contents of the arrays in the first two input record are as follows:

<table>
<thead>
<tr>
<th>Record</th>
<th>Array</th>
<th>Array Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ARA</td>
<td>12345678901234567890</td>
</tr>
<tr>
<td></td>
<td>ARC</td>
<td>01234567890123456789876N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(note than N equals minus 5)</td>
</tr>
<tr>
<td>2</td>
<td>ARB</td>
<td>JOHN␣DOE␣JOE␣SMITH␣LEE␣MARX␣JIM␣KNOTS␣TIM␣TYLER</td>
</tr>
<tr>
<td></td>
<td>ARC</td>
<td>(the same as record 1)</td>
</tr>
</tbody>
</table>

In the first output record, the location and contents of the arrays are as follows (b = blank):

<table>
<thead>
<tr>
<th>Array</th>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARA (first element)</td>
<td>85–89</td>
<td>12345</td>
</tr>
<tr>
<td>ARC</td>
<td>37–84</td>
<td>b1.23bb45.67bb 89.01bb23.45bb 67.89bb87.65CR</td>
</tr>
</tbody>
</table>

For the second output record assume that the content of field X1 is 4; the locations and contents of the arrays are as follows:

<table>
<thead>
<tr>
<th>Array</th>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB (fourth element)</td>
<td>91–100</td>
<td>JIMbKNOTSb</td>
</tr>
<tr>
<td>ARC</td>
<td>37–84</td>
<td>b1.23bb45.67bb 89.01bb23.45bb 67.89bb87.65CR</td>
</tr>
</tbody>
</table>

*Figure 155 (Part 2 of 2). Formatting Output Fields by Using Arrays*

Figure 156 on page 425 shows a method of printing one array element per line. Each time the EXCPT operation on line 03 of the calculation specifications occurs, one element of the 22-element array AR2 is written to the output file ARFILE.
Figure 156. Printing One Array Item per Line

Figure 157 on page 426 shows a method of printing more than one array element per line. The number of elements printed on a line depends on the value coded as factor 2 in the COMP operation COMP 10 calculation specifications. In this example, that value is 10, and the number of elements in array AR2 is 50.

If an edit code is used, each array element is preceded by two spaces. You must take these spaces into account when you compute the end position for the output specifications.
Example of Using Tables

The following payroll program requires two related tables:

<table>
<thead>
<tr>
<th>TABNUM</th>
<th>TABRAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>407</td>
</tr>
<tr>
<td>12346</td>
<td>593</td>
</tr>
<tr>
<td>12347</td>
<td>369</td>
</tr>
<tr>
<td>12348</td>
<td>390</td>
</tr>
<tr>
<td>12349</td>
<td>1379</td>
</tr>
</tbody>
</table>

TABNUM, which contains employee numbers, is the table searched. TABRAT, which contains employee salary rates, is the related table. After the program finds an employee’s number and salary rate, it multiplies the rate by the number of hours worked. The result is the amount earned.
File Description Specifications

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME IPAEBLENRLKLK1AI0VKLOCEDVICE+........EXIT+......A....U+.*
FILE DESCRIPTION SPECIFICATIONS

FILETIMECARDIPE F 96 96 DISK
FRATETABLIT F 72 72 EDISK

The input records are contained in the input file TIMECARD, which is designated as
the primary file (P in column 16). When the file reaches end of file, processing
ends (E in column 17). Each record in this file is 96 positions long. This file is
read from disk.

The related tables are contained in the input file RATETABL, which is designated
as a table file (T in column 16). This file is read from disk before the program is
run. Each record in this file is 72 positions long. The E in column 39 shows that
the extension specifications contain additional information about this file.

Extension Specifications

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++ArrnamEntParrLenPDSAltnamLenPDSComments+++++++*
E RATETABL TABNUM 8 500 5 0 T ABRAT 4 2

The extension specifications complete the description of the file RATETABL. The
table searched is TABNUM (columns 27 through 32), which has eight elements in
each record (columns 33 through 35) and 500 elements in the table (columns 36
through 39). Each element is five positions long (columns 40 through 42) with zero
decimal positions (column 44). The table is organized in ascending sequence
(column 45).

The related table is T ABRAT (columns 46 through 51). Each element is four posi-
tions long (columns 52 through 54) with two decimal positions (column 56).

The table input records are organized in alternating format (although related tables
do not have to be in alternating format). That is, the first record begins with the
first element of TABNUM, which is followed by the first element of T ABRAT, the
second element of TABNUM, the second element of T ABRAT, and so on in alter-
nating sequence. Each element of TABNUM is five positions long, and each
element of T ABRAT is four positions long. Therefore, each pair of related elements
is nine positions long. There are eight elements of each table per record, so each
record in RATETABL is 72 positions long.

Each table has 500 elements, so the file requires 63 records (500 elements divided
by 8 elements per record = 62.5 records). The first 62 records contain data in
positions 1 through 72, but the 63rd record contains data only in positions 1
through 36.
EXAMPLE OF USING TABLES

Input Specifications
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC................................S
ITIMECARDAA 01
I..........................................PFromTo++DField+L1M1FrPoNeEq...*
I 1 50EMPNUM
I 42 441HRSWKD

The input file TIMECARD is assigned a sequence of AA (columns 15 and 16). Record-identifying indicator 01 turns on whenever an input record is present for processing. No record identification codes are specified in columns 21 through 41 because there is only one record type. Lines two and three describe the locations of the two input fields used by the program. The employee number (EMPNUM) is in positions 1 through 5 of the input record. The number of hours worked (HRSWKD) is in positions 42 through 44 of the input record.

Calculation Specifications
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C 01 EMPNUM LOKUPTABNUM TABRAT 03
C 03 TABRAT MULT HRSWKD EARNS 52H
C N03 MOVE 000.00 EARNS

The LOKUP operation instructs the program to search the table TABNUM (factor 2) for an element that matches the value of the field EMPNUM (factor 1). Resulting indicator 03 turns on when an element in TABNUM is found that is equal to the value of EMPNUM. The related table TABRAT is specified as the result field.

When indicator 03 is on, the MULT (multiply) operation in line 02 is done. The salary rate for the employee, taken from the related table TABRAT (factor 1) is multiplied by the number of hours worked, HRSWKD (factor 2). The result is stored in the result field EARNS, which is five positions long with two decimal positions. The result is half-adjusted (H in column 53).

When indicator 03 is not on, the MOVE operation in line 03 occurs. The literal 000.00 (factor 2) is moved into the field EARNS (result field) to indicate that the table does not contain an entry for that employee. The decimal point in the literal is used only to align the data; it is not actually put in the field EARNS.
### Chapter 15. Using Data Structures

A **data structure** is an area in storage that is composed of one or more fields, called **subfields**. You can use a data structure to:

- Define that area of storage in more than one way
- Subdivide an input field so that your program can refer to either the entire field or its subfields
- Reorganize fields in an input record for easier reference.

See “Examples of Data Structures” on page 431, for an example of a data structure used for each of these purposes.

### Coding a Data Structure

Data structures are coded on input specifications. They must be the last entries on the input specifications. That is, they must follow all specifications for input records.

Specifications for a data structure have two parts: the data structure statement and the subfields. Specifications for the subfields must be coded on the lines immediately below the specification for the data structure statement.

To code a data structure statement and subfields, make entries in the unshaded columns of the input specifications as shown below:

<table>
<thead>
<tr>
<th>I</th>
<th>Line</th>
<th>Filename or Record Name</th>
<th>Sequence</th>
<th>Record Identification Codes</th>
<th>RPG Field Name</th>
<th>Field Name</th>
<th>Decimal Positions</th>
<th>From</th>
<th>To</th>
<th>Field Occurs Times</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1994
CODING A DATA STRUCTURE

Data Structure Statement

Columns 7 through 12 can contain blanks or the name of the data structure. Although columns 7 through 14 are normally used as one entry, the name of the data structure cannot be more than six characters long.

Column 18 must contain U if this data structure is used as a local data area for a display station (see “Local Data Area for a Display Station” on page 436). Otherwise, leave this column blank.

Columns 19 through 20 must contain DS, which identifies this statement as a data structure.

Subfields

Columns 44 through 47 must contain the record position in which the subfield begins, relative to the beginning of the data structure, not relative to the beginning of the input record.

Columns 48 through 51 must contain the record position in which the subfield ends, relative to the beginning of the data structure, not relative to the beginning of the input record.

Column 52 must contain the number of decimal positions if the subfield is numeric. It must be blank if the subfield is alphabetic.

Positions 53 through 58 must contain the subfield name. The subfield name can be the same as an input field name or a result field name. Subfields can be used as factor 1, factor 2, or the result field of a calculation specification or as output fields. However, the same subfield name cannot be used in more than one data structure, and a data structure name cannot be used as a subfield name in another data structure.

Rules for Coding Data Structures

A data structure is considered alphabetic data. Therefore, when a data structure is created, it is set to blanks, except for those subfields that are set by an array or by a local data area for a display station. You must ensure that numeric subfields contain numeric data before you use the subfields in CHAIN, READE, LOKUP, COMP, IFxx, DOUxx, DOWxx, CASxx, editing operations, or arithmetic operations.

A data structure can be from 1 to 9999 characters long. However, the maximum length of a data structure used as a local data area for a display station is 512 characters.

The maximum length of an alphabetic subfield is 256 characters; the maximum length of a numeric subfield is 15 characters.

If arrays are specified as subfields, the length specified must equal the amount of storage required to store the entire array.
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 highest entry in columns 48 through 51 of a subfield if the data structure name is not an input field.

The length of the data structure is determined by the first specification in the program that defines a length in one of the ways just listed. Conflicting lengths in later specifications are not valid.

The name of an input field or a result field that is being redefined in a data structure must be the data structure name or must be specified in the data structure; however, it does not have to immediately precede the subfields redefining it.

If a field appears as a data structure name or as a data structure subfield name, the physical space reserved for that field is in the data structure, regardless of where the field was defined.

Look-ahead fields cannot appear as a data structure or a subfield.

An RPG reserved word, array item, or table name cannot be specified as a subfield.

A packed-decimal or binary numeric field cannot be specified as a subfield within the data structure. If a field is defined as packed-decimal or binary in a file, the program converts that field to zoned-decimal format when it places the field in the data structure.

Examples of Data Structures

Example 1. Defining One Area of Storage in More than One Way

Figure 158 on page 432 shows a 40-position data structure that defines one area of storage in three ways:

- Positions 1 through 32 are defined as the sales record (SREC).
- Positions 1 through 35 are defined as the purchase record (PREC).
- Positions 1 through 40 are defined as the transfer record (TREC).

Figure 159 on page 433 shows the coding for the data structure in Figure 158 on page 432. The DS in columns 19 and 20 of line 08 identifies the following lines as a data structure. The data structure allows the programmer to define 19 subfields (lines 10 through 15, 17 through 23, and 25 through 30) within only 40 positions of storage.

If the programmer defined each subfield as a field in the input record instead of using a data structure, each input field would require a separate area of storage. That is, the sales record would require 32 positions, the purchase record would require an additional 35 positions, and the transfer record would require an additional 40 positions.
Together, the three records would require 117 positions of storage, almost three times as much as the 40 positions required for the data structure.

Figure 158. Data Structure That Defines One Area of Storage Three Ways
Example 2. Defining Subfields within a Field

Figure 160 on page 434 shows a data structure that subdivides a field in an input record. Input field PARTNO in file FILEIN has 16 positions (from record position 3 through record position 18; see line 02). The data structure defines subfields in field PARTNO.

If a data structure defines subfields within a field, and if that data structure has a name in columns 7 through 12, the data structure name must be the same as the field name (in this case, PARTNO).
Normally, a data structure name cannot be specified as factor 1, factor 2, or the result field of a calculation specification. However, a data structure name can be specified as the result field of an RLABL operation. A data structure subfield name can be specified in a calculation specification. Subfield PARTDS contains all 16 positions in the data structure, so subfield PARTDS can be used to specify the entire data structure in a calculation specification.

```
*.. 1 +... 2 +... 3 +... 4 +... 5 +... 6 +... 7 ..*
FILENAME\$NORI\$POS1\$NCC\$POS2\$NCC\$POS3\$NCC

FILEIN 01 1 CA 2 CB
I........................................PFROMTO++DFIELD+L1M1FROPONEQ...*
I               3 18 PARTNO
I               19 29 NAME
I               30 40 PATNO
I               41 61 DR

PARTNO     DS
I              1 4 MFG
I              5 10 DRUG
I             11 13 STRNTH
I             14 160COUNT
I              1 16 PARTDS
```

Figure 160. Using a Data Structure to Define Subfields within a Field

A data structure can also redefine subfields within a subfield. Figure 161 shows that subfields KEY, ARRFLD, and ID are redefined by subdividing them into even smaller subfields.

```
*.. 1 +... 2 +... 3 +... 4 +... 5 +... 6 +... 7 ..*
FILENAME\$NORI\$POS1\$NCC\$POS2\$NCC\$POS3\$NCC

FILEIN 01 1 CA 2 CB
I........................................PFROMTO++DFIELD+L1M1FROPONEQ...*
I                       1 10 KEY
I                      1 20DIV
I                      3 30REGION
I                      4 50BRANCH
I                      6 10 EMPL
I                     11 28 ARRFLD
I                      11 15 ID
I                     11 11 PLANT
I                     12 150MANN
I                     16 193RATE
I                     20 260HOURS
I                     27 28 CODE
```

Figure 161. Using a Data Structure to Define Subfields within a Subfield
Example 3. Reorganizing Fields in an Input Record

Figure 162 shows a data structure that is used to reorganize fields in an input record. Records in file TRANSACT contain input fields in the following sequence: PARTNO, QTY, TYPE, CODE, and LOCATN. Data structure KEYDS reorganizes this sequence. The sequence of subfields in the data structure is LOCATN, PARTNO, and TYPE. Input fields QTY and CODE are not part of the data structure. Subfield PRTKEY includes all 16 positions of the data structure, so it allows you to specify the entire data structure in a calculation specification.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC................................
ITRANSACT 01 1 C1 2 C2
I........................................PFromTo++DField+L1M1FrPoNeEq...*
I 3 10 PARTNO
I 11 160QTY
I 17 20 TYPE
I 21 21 CODE
I 22 25 LOCATN
IKEYDS DS
I 1 4 LOCATN
I 5 12 PARTNO
I 13 16 TYPE
I 1 16 PRTKEY
```

Figure 162. Using a Data Structure to Reorganize Fields

Special Data Structures

SAVDS Data Structure

The data structure in Figure 163 on page 436 is used as the SAVDS data structure, which is specified on a continuation line of the file description specification for the WORKSTN file. This data structure contains fields that are saved and restored for each display station that uses the WORKSTN file.

For more information about the SAVDS data structure, see “Continuation-Line Options” on page 182 in Chapter 7, “Using a WORKSTN File.” For an example of a SAVDS data structure, see Figure 78 on page 231, which shows sample program AR330R.
Local Data Area for a Display Station

Figure 164 on page 437 shows a data structure that is used as the local data area for a display station. A local data area contains 512 positions of storage that is used for passing information between programs and procedures. Coding a local data area data structure requires a U in column 18. The data structure name is optional.

At the beginning of the program, the program reads information from the local data area into the data structure; at the end of the program, the program writes information from the data structure into the local data area. You can also use the OCL LOCAL statement or another RPG program to enter information into the local data area.

For a Multiple Requester Terminal (MRT) program, the local data area data structure contains a copy of the local data area for the first display station using the program, and the local data area data structure is not automatically written out at the end of the program. To read and write the local data area for each display station in a MRT, use SUBR21 (see Chapter 7, “Using a WORKSTN File”). For an example of a local data area, see Figure 81 on page 243, which shows sample program AR935R.

File Information Data Structure

A file information data structure (INFDS) is used for passing information about an exception or error in a WORKSTN file to the RPG program. This information includes the type of exception or error that occurred, the operation that the program was running when the exception or error occurred, and the status of various conditions. The INFDS data structure is specified as a continuation-line option on the file description specification for a WORKSTN file.

For more information about the file information data structure, see “Handling Exceptions and Errors” on page 191 in Chapter 7, “Using a WORKSTN File.”
SPECIAL DATA STRUCTURES

Figure 164. Using a Data Structure as a Local Data Area for a Display Station
SPECIAL DATA STRUCTURES
Chapter 16. Editing Numeric Fields

Editing is used to interpret data stored in the computer so that the output is readable and can be printed on a report or displayed on a terminal screen.

Data, and numeric data especially, is usually stored in a condensed form in computer memory in order to save space and processing time.

When the program knows it is dealing strictly with numeric fields, it need not read the four higher order bits of each byte in order to identify the numbers. The four higher order bits of the last digit in any sequence are therefore used to store the sign of the number. This would result in a nonnumeric last character when the data is printed or displayed. The data needs to be edited in order to extract this sign and return the higher order bits of the last number to their numeric status.

Data is also usually stored with leading zeros, and without commas, decimal points, currency symbols, slashes (for dates), and so forth. When the data is printed or displayed, passing the data through an edit word (also called a "mask") suppresses these leading zeros, and restores commas, decimal points, currency symbols and slashes.

The edit word can also be used to insert repetitive character strings (such as the word "date") in all cases, or in only certain cases (such as only for negative numbers). When you print fields that are not edited, the fields appear exactly as they are represented inside the computer. The following examples show why numeric output fields should be edited:

<table>
<thead>
<tr>
<th>Type of Field</th>
<th>Field in the Computer</th>
<th>Printing of Unedited Field</th>
<th>Printing of Edited Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphameric</td>
<td>JOHN T SMITH</td>
<td>JOHN T SMITH</td>
<td>JOHN T SMITH</td>
</tr>
<tr>
<td>Numeric</td>
<td>0047652</td>
<td>0047652</td>
<td>47652</td>
</tr>
<tr>
<td>Numeric (negative)</td>
<td>004765K</td>
<td>004765K</td>
<td>47652-</td>
</tr>
</tbody>
</table>

The unedited alphameric field and the unedited positive numeric field are easy to read when printed, but the unedited negative numeric field is confusing because it contains a K, which is not numeric. The K is a combination of the digit 2 and the negative sign for the field. They are combined so that one of the positions of the field does not have to be set aside for the sign. The combination is convenient for storing the field in the computer, but it makes the output hard to read. Therefore, numeric fields need editing before they are printed.

When you edit fields in a file assigned to any device other than a PRINTER (in columns 40 through 46 of the file description specifications), you must be aware of the contents of the edited field if you want the field read back into the program. You must also be aware of the effects of any operations you plan to use on an edited field. For example, if you add an unedited field to an edited field, the results will be wrong.
There are two ways to edit a numeric output field: using an edit code or using an edit word. Edit codes are easier to use, because you merely select the predefined type of editing you want. On the other hand, edit words allow you to do more, because you define exactly the kind of editing you want.

**Edit Codes**

There are several different edit codes available. Each code edits in a slightly different way according to a set pattern. However, all of them remove the sign of the field so that the rightmost digit always prints as a number.

Figure 165 on page 441 shows the edit pattern for all edit codes. You choose the code that edits a field the way you want it to appear. For example, suppose you want to print blanks instead of zeros, to print decimal points and commas, but not to print the sign of a field. Figure 165 on page 441 shows that edit codes 1 and 2 both do this editing. The difference between edit codes 1 and 2 is that, when the field is zero, edit code 1 prints zeros and edit code 2 prints blanks.

Table 8 on page 442 shows how various edit codes edit the same data.
<table>
<thead>
<tr>
<th>Edit Code</th>
<th>Commas</th>
<th>Decimal Point</th>
<th>Sign for Negative Balance</th>
<th>D or Blank</th>
<th>I</th>
<th>J</th>
<th>Zero Suppress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>No sign</td>
<td>.00 or 0</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td>No sign</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td>No sign</td>
<td>.00 or 0</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>No sign</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>A</td>
<td>Yes</td>
<td>Yes</td>
<td>CR</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Yes</td>
<td>Yes</td>
<td>CR</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>Yes</td>
<td>CR</td>
<td>.00 or 0</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Yes</td>
<td>CR</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>J</td>
<td>Yes</td>
<td>Yes</td>
<td>-(minus)</td>
<td>.00 or 0</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
</tr>
<tr>
<td>K</td>
<td>Yes</td>
<td>Yes</td>
<td>-(minus)</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>L</td>
<td>Yes</td>
<td>-(minus)</td>
<td>.00 or 0</td>
<td>.00 or 0</td>
<td>0.00 or 0</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Yes</td>
<td>-(minus)</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Blanks</td>
<td>Yes</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. The X code performs no editing.
2. The Y code suppresses the leftmost zero only. The Y code edits a three- to six-digit field according to the following pattern:

- nn/n
- nn/nn
- nn/nn/n
- nn/nn/nn

_Figure 165. Edit Codes_
## Table 8. Examples of How Different Edit Codes Edit the Same Data

<table>
<thead>
<tr>
<th>Edit Codes</th>
<th>Positive Number - Two Decimal Positions</th>
<th>Positive Number - No Decimal Positions</th>
<th>Negative Number - Three Decimal Positions</th>
<th>Negative Number - No Decimal Positions</th>
<th>Zero Balance - Two Decimal Positions</th>
<th>Zero Balance - No Decimal Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unedited</td>
<td>1234567</td>
<td>1234567</td>
<td>00012b</td>
<td>00012b</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>1</td>
<td>12,345.67</td>
<td>1,234,567</td>
<td>.120</td>
<td>120</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12,345.67</td>
<td>1,234,567</td>
<td>.120</td>
<td>120</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120</td>
<td>120</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120</td>
<td>120</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>12,345.67</td>
<td>1,234,567</td>
<td>.120CR</td>
<td>120CR</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>12.345.67</td>
<td>1,234,567</td>
<td>.120CR</td>
<td>120CR</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120CR</td>
<td>120CR</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120CR</td>
<td>120CR</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>12,345.67</td>
<td>1,234,567</td>
<td>.120-</td>
<td>120-</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>12,345.67</td>
<td>1,234,567</td>
<td>.120-</td>
<td>120-</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120-</td>
<td>120-</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>12345.67</td>
<td>1234567</td>
<td>.120-</td>
<td>120-</td>
<td>.00</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1234567</td>
<td>1234567</td>
<td>00012b</td>
<td>00012b</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>Y</td>
<td>0/01/20</td>
<td>0/01/20</td>
<td>0/00/00</td>
<td>0/00/00</td>
<td>0/00/00</td>
<td>0/00/00</td>
</tr>
<tr>
<td>Z</td>
<td>1234567</td>
<td>1234567</td>
<td>120</td>
<td>120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Table 8 the b represents a blank. This may occur if a negative zero does not correspond to a printable character.
To use an edit code, code the unshaded columns of the output specifications as shown below:

Columns 23 through 31 can contain conditioning indicators.

Columns 32 through 37 must contain the name of a numeric field.

Column 38 must contain an edit code.

Column 39 can contain B to indicate that the numeric field is to be set to zero after it is printed.

Columns 40 through 43 can contain the end position of the field in the output record.

Columns 45 through 47 can contain one of the following:

- An asterisk * if you want asterisks to replace the leading zeros of the field.
- The currency symbol enclosed in apostrophes if you want a floating currency symbol. The currency symbol will then appear before the first digit in the field.

A fixed currency symbol in columns 45 through 47 must be coded on a line before or following the edit code. The currency symbol remains in the end position specified.

**Note:** You cannot use the X, Y, or Z edit code if you code an asterisk or the currency symbol in columns 45 through 47 of the output specifications.

When you use an edit code to punctuate an entire array, two spaces are skipped before each edited element.
Examples of Using the Currency Symbol with an Edit Code

Suppose you want to print a currency symbol on a report for a field called AMOUNT. An edit code will not put the currency symbol there. You specify this in addition to the edit code you are using.

When you use a floating currency symbol, the currency symbol changes positions so that it prints immediately in front of the first digit. In this case, the AMOUNT field could look like any of the following (N stands for any number):

$NNN.NNN
$NN.NNN
$N.NNN
$.NNN

Note: If the currency symbol is not the dollar sign ($), the currency symbol must be entered in column 18 of the control specification.

See Figure 166 for a coding example of a floating currency symbol.

```
*.. 1 +... 2 +... 3 +... 4 +... 5 +... 6 +... 7 ..*
OName++++DFBASbSaN01N02N03Excptn............................*
OPRINTER H 204 0V
0 OR 1P
0................N01N02N03Field+YBEnd+PConstant/editword+++++++...
0
0 H 2 0V
0 OR 1P
0
0
0
0
0 D 1 01
0*
0* The floating dollar sign is specified by placing '$' in columns 45 through 47 of the same line as the edit code.
0*
0
AMOUNTJ 75 '$'
```

Figure 166. Floating Currency Symbol
When you use a fixed currency symbol, the currency symbol remains in the end position specified on the output specifications. In this case, the AMOUNT field could look like any of the following (N stands for any number):

- \$NNN.NN
- $ NN.NN
- $ N.NN
- $ .NN

The blanks between the first digit and the currency symbol are the result of zero suppression. See Figure 167 for a coding example of a fixed currency symbol.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++++DFBASbSaNO1NO2NO3Excptn....................................*
OPRINTER H 204 OV
0 OR 1P
0.................NO1NO2NO3Field+YBEnd+PConstant/editword+++++++++++++.*
0
068 'WEEKLY SALES REPORT'
0 H 2 OV
0 OR 1P
0
033 'DEPT'
0
51 'SALESMAN'
0
75 'AMOUNT'
0
84 'TOTAL'
0
D 1 01
0*
0* The fixed dollar sign is specified by placing '$' in columns 45 through 47 of the line before the edit code.
0*
0
69 '$'
0 AMOUNTJ 75

Figure 167. Fixed Currency Symbol
```
Example of Using Asterisks with an Edit Code

When you use asterisks to fill the spaces between the currency symbol and the first digit, the AMOUNT field could look like any of the following (N stands for any number):

$NNN.NN
$*NN.NN
$**N.NN
$***.NN

See Figure 168 for a coding example that uses asterisks to punctuate a field.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaNO1N02N03Excptn...........................................

OPRINTER H 204 OV
0 OR 1P

.................NO1N02N03Field+YBEnd+PConstant/editword+++++++++++*

0 68 'WEEKLY SALES REPORT'

0 H 2 OV
0 OR 1P

0 33 'DEPT'
0 51 'SALESMAN'
0 75 'AMOUNT'
0 84 'TOTAL'

0 D 1 01
0 69 '$'

0 To make asterisks fill the empty spaces caused by zero suppression, place '*' in columns 45 through 47 of the same line as the edit code.

0 AMOUNTJ 75 '*'

Figure 168. Punctuating with Asterisks
Edit codes are also used to edit date fields. The edit code for a date field is Y. See Table 9 for various ways to edit a date field.

<table>
<thead>
<tr>
<th>UPDATE</th>
<th>Edit Code</th>
<th>Contents of Column 19</th>
<th>Contents of Column 20</th>
<th>Blank</th>
<th>D</th>
<th>I/J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 30, 1994</td>
<td>Y</td>
<td>Blank</td>
<td>Blank</td>
<td>1/30/94</td>
<td>30/01/94</td>
<td>30-01-94</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>1-30-94</td>
<td>30-01-94</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>Blank</td>
<td>-</td>
<td>1/30/94</td>
<td>1/30-94</td>
<td>1/30-94</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1-30-94</td>
<td>1-30-94</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>Blank</td>
<td>-</td>
<td>30/01/94</td>
<td>30/01-94</td>
<td>30/01-94</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30-01-94</td>
<td>30-01-94</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>Blank</td>
<td>-</td>
<td>90/01/30</td>
<td>90/01-30</td>
<td>90/01-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90-01-30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Any character may be specified in position 20 as the separator character. For the purpose of this example, a dash has been used as the separator character.

Figure 169 shows the effects that the various edit codes have on the same field with a specified end position for output.
Negative Number, 2 Decimal Positions.  
End Position Specified as 10.

<table>
<thead>
<tr>
<th>Edit Code</th>
<th>Output Print Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Unedited</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>J</td>
<td>4</td>
</tr>
<tr>
<td>K</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>4</td>
</tr>
<tr>
<td>M</td>
<td>4</td>
</tr>
<tr>
<td>N</td>
<td>-4</td>
</tr>
<tr>
<td>O</td>
<td>-4</td>
</tr>
<tr>
<td>P</td>
<td>-4</td>
</tr>
<tr>
<td>Q</td>
<td>-4</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y^2</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

^1K represent a negative 2.

^2A field edited by the Y edit code must have zero decimal positions.

Figure 169. Effect of Edit Codes on End Position
Edit Words

Use edit words when you have unusual (special) editing requirements. An edit word allows you to specify directly:

- If commas, decimal points, and zero suppression are needed
- If the negative sign should print
- If a currency symbol and leading asterisks should be used
- If the constant(s) have to be printed.

An edit word gives a pattern for punctuation. When you create an edit word, you are setting up your own editing pattern.

To use an edit word, code the unshaded columns of the output specifications as shown below:

Columns 23 through 31 can contain conditioning indicators.

Columns 32 through 37 must contain the name of a numeric field.

Column 38 (Edit Codes) must be blank.

Column 39 can contain b to indicate that the numeric field is to be set to zero after it is printed.

Columns 40 through 43 can contain the end position of the field in the output record.

Columns 45 through 70 must contain the edit word. The edit word can be up to 24 characters long and must be enclosed in apostrophes. Enter the leading apostrophe in column 45. The actual edit word must begin in column 46.

Editing Considerations

When using an edit word, make sure that there is enough space on the printer form for the edited output field. If the field you want to edit is six characters long, for example, the edited output field might contain more than six characters.

When you compute the length of an edited output field, determine how many of the editing characters are replaceable. A replaceable character is one that will be replaced by a digit from the data field. The number of replaceable characters must equal the length of the field edited.

The following summary provides more information on creating edit words and describes certain characters that have special meaning when used in an edit word.
The **Delta position** is defined as the position in the edit word that is to contain the first, or leftmost, character of the data field. The examples referred to are provided in the section “Examples of Edit Words,” later in this chapter.

**Blanks (b)**
This is a replaceable character. Blanks that are not preceded by an ampersand are always replaced with data from the source field.

**Constants**
Constants are any combination of characters, including commas and decimal points, but not including special uses of 0, *, currency symbols, &, -, or CR symbols.

*When to the right of the Delta position (see examples 11 to 14):* A constant imbedded in replaceable characters will print only if a significant digit from the source data appears to its left in the edited output field. A constant between the last replaceable character and a negative indicator (see “The Minus Sign (-) and the Character “CR”” on page 451) will print only if the source data is negative. Constant(s) at the end of the edit word will always print. Constants are not replaceable characters.

*When to the left of the Delta position (see example 15):* Constants are printed only if preceded by a zero in the edit word.

**The First Zero (0)**

*When to the right of the Delta position (see example 2):* Leading zeros to the left of and including the position in which the zero appears will be suppressed, but leading zeros to the right will not be suppressed. Any constants to the left of the zero will print only if preceded by a significant digit. The zero is a replaceable character.

*When in the Delta position (see example 3):* Leading zeros and constants to the right of the zero will print. The zero will not print in the first position of the output field. If the field contains leading zeros, a blank will print in the first position; otherwise, a significant digit will print.

*When to the left of the Delta position (see example 4):* The results are described in the preceding paragraph, except that if the field contains leading zeros, a zero will print in the first position of the output field. The zero in the edit word is not a replaceable character and does not print.

**Note:** Any zero or asterisk to the right of the first zero is treated as a constant.

**The First Asterisk (*)**

*When to the right of the Delta position (see example 9):* Leading zeros to the left of the asterisk (and the asterisk itself) are replaced by asterisks. Constants will be replaced with asterisks when no significant digits precede the constant(s). Leading zeros to the right of the asterisk will not be suppressed. The asterisk is counted as a replaceable character.

*When in the Delta position (see example 10):* An asterisk will print in the first position of the output field unless there is a significant digit in that position. Leading zeros and constants are not suppressed. The asterisk is counted as a replaceable character.
Note: Any asterisk or zero to the right of the first asterisk is treated as a constant.

The Minus Sign (-) and the Character “CR”
(See examples 5, 6, 17): These symbols are used to identify negative fields on printed output and will print only if the field is negative. If the field is positive, they are replaced by blanks. Only the first - or CR to the right of all of the replaceable characters is treated as a negative indicator; all others are treated as constants. Any constants between the last replaceable character and the negative indicator will print only when the field is negative. Any constants following the negative indicator will always print. The - and CR are not counted as replaceable characters.

Currency Symbols
When to the right of the Delta position (see example 7): A currency symbol followed directly by a zero is said to float: it will print in the position immediately to the left of the first significant digit. The currency symbol may be replaced by a significant digit, but it should not be counted in the total of replaceable characters. A currency symbol not directly followed by a zero will be treated as a constant. The currency symbol is not counted as a replaceable character.

When to the left of the Delta position (see example 8): The currency symbol will always print to the left of the first position of the output field. Leading zeros will be suppressed along with any constants that are not preceded by a significant digit. The currency symbol is not counted as a replaceable character.

The Ampersand (&)
(See example 12): The ampersand must be coded before a blank that is to appear as a blank in the output field.
Examples of Edit Words

For all examples, column 38 (edit codes) of the output specification is blank. The symbol \( \cdot \) indicates where blank spaces would appear in the output result. The symbol \( \Delta \) marks the delta position in each example.

Example 1 – Suppressing decimal point and leading zeros
In the example below all the leading zeros will be suppressed and the decimal point will not print unless there is a significant digit to its left:

```
<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000000012</td>
<td>bbbbbbbbb12</td>
</tr>
<tr>
<td></td>
<td>000000123</td>
<td>bbbbbbbbb1.23</td>
</tr>
</tbody>
</table>
```

Example 2 – Forcing the decimal point
This example causes the decimal point to print even if the field is equal to zero:

```
<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000001</td>
<td>bbbbbbb.01</td>
</tr>
<tr>
<td></td>
<td>00000000</td>
<td>bbbbbbb.00</td>
</tr>
</tbody>
</table>
```

Example 3 – Forcing leading zeros
Leading zeros will print to the left of the first significant digit. Note that a blank, not a zero, prints in the first position:

```
<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000000123</td>
<td>b000001.23</td>
</tr>
</tbody>
</table>
```

Example 4 – Printing a leading zero in the leftmost position of output field
If you want a zero to print in the leftmost position of the output field, the zero must be placed to the left of the Delta position in the edit word (see “Editing Considerations” on page 449). Note that seven blanks were coded to the left of the decimal point, whereas in example 3, only six blanks were coded.
The zero in the edit word will not be printed:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$ \quad$$</td>
<td>00000123</td>
<td>000001.23</td>
</tr>
</tbody>
</table>

**Example 5 – Indicating a negative value**

This example adds a negative value indication. The minus sign will print only when the value in the field is negative. A CR symbol has the same function as a minus sign:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$ \quad$$</td>
<td>00000123</td>
<td>bbbb1.23</td>
</tr>
</tbody>
</table>

**Example 6 – Inserting commas**

Commas are added to separate thousands, millions, and so on. The comma will print only if preceded by a significant digit:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$ \quad$$</td>
<td>123456789</td>
<td>1,234,567.89</td>
</tr>
</tbody>
</table>

**Example 7 – Inserting a floating currency symbol**

A floating currency symbol will print in the position immediately to the left of the first significant digit or to the left of the decimal point if the field is zero or less than 100:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$ \quad$$</td>
<td>00000012</td>
<td>bbbbbb123.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$ \quad$$</td>
<td>000123456</td>
<td>bbbbs1.234.56</td>
</tr>
</tbody>
</table>
Example 8 – Inserting a fixed currency symbol
The currency symbol may also be printed in the position before the first digit of the output field. Constants to the left of the first significant digit are replaced by blanks. See the explanation under the currency symbol in the section “Editing Considerations” on page 449 for limitations regarding the placement of the currency symbol.

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output Record as</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 0 1 2 3 4 5 6</td>
<td>000123456</td>
<td>$1,234.56</td>
</tr>
</tbody>
</table>

Example 9 – Replacing leading blanks/constant by asterisks
Blanks and constants to the left of the first significant digit are replaced by asterisks (asterisk fill):

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output Record as</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 0 1 2 3 4 5 6</td>
<td>000123456</td>
<td>$1,234.56</td>
</tr>
</tbody>
</table>

Example 10 – Leading blank/constant replaced in first position only
An asterisk will be printed in the first position only, unless the field contains a significant digit in the first position. Leading zeros and constants are not suppressed:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output Record as</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 0 1 2 3 4 5 6</td>
<td>000123456</td>
<td>$01,234.56</td>
</tr>
<tr>
<td>* 1 2 3 4 5 6 7 8 9</td>
<td>123456789</td>
<td>1,234.56789</td>
</tr>
</tbody>
</table>

Example 11 – Adding text to negative numbers
Constants between the last replaceable character and the '-' or CR symbol will print only if the field is negative; otherwise, blanks will print in these positions.
Note the use of ampersands to represent blanks:

<table>
<thead>
<tr>
<th>Edit Word</th>
<th>Source Data</th>
<th>Appears in Output Record as:</th>
</tr>
</thead>
</table>
| $ | ** | ** | ** | ** | 00000123 | BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Example 16 – Second occurrences of special characters are treated as constants
Note that any zeros or asterisks (as well as - and CR) as following the first occurrence of either are treated as constants.

Example 17 – Defining a single occurrence of an asterisk or zero as a constant
If an asterisk or a zero is to appear as a constant and there are no other asterisks or zeros preceding it in the edit word, the asterisk or zero must be defined on a separate specification line as a constant ending in the appropriate print position.

Example 18 – Defining a single occurrence of CR as a constant
Note that the CR in the middle of a word may be detected as a negative field value indication. If a word such as SECRET is required, use the coding in the example below, line 3:
Creating Edit Words

The printer spacing chart can help you create edit words. Figure 170 shows how an output line can be created on this chart. The Xs and zeros show field positions. A zero indicates where zero suppression stops. An X indicates that any number can appear in the position. Use blanks in place of the Xs when writing the edit words.

If it is necessary to show a negative number, you must include a sign in the edit word. Use either the minus sign (-) or the letters CR. These print only for a negative number; however, the character positions they require must be included when you enter the end position of the field on the output specifications.

Figure 170 shows an edit word (line 08 of the output specifications) that causes CR to print if the field PERCPL has a negative balance. For example, if the field PERCPL contains -25 (which in storage appears as 2N), the printed output is 25CR. If PERCPL is positive, the CR does not print; the printed output is 25.

Another way to indicate a negative number is to use a minus sign. To leave a space between the number and the negative sign, place an ampersand (&) in the edit word before the minus sign. The PERCPL field then prints as 25 -.
Figure 170. Using the Printer Spacing Chart to Create Edit Words
Chapter 17. Changing the Hexadecimal Value of Characters

Each alphabetic, numeric, and special character is represented in the computer by a separate hexadecimal value. To determine if the value of one character is larger than the value of another character, the computer assigns a sequence to the hexadecimal values of the characters. This sequence is called the normal collating sequence. To collate means to place items in proper sequence or to check that items are in proper sequence. Figure 171 on page 460 shows the normal collating sequence and hexadecimal value of each character.

You can change this normal collating sequence in two ways:

- By temporarily using one character in place of another during a comparison but using the original character at all other times during the program. This method is called changing the collating sequence.

- By using one character in place of another in one or more files whenever the file is used throughout an entire program. This method is called translating a file.
When zones are specified for record identification codes, the & is considered to have a hexadecimal C zone, the - (minus sign) is considered to have a hexadecimal D zone, and the blank is considered to have a hexadecimal F zone.
Changing the Collating Sequence

There are three reasons why you might want to change the normal collating sequence of characters:

- To compare alphameric characters
- To check the sequence of characters
- To check for match fields.

For example, you may want alphabetic characters to follow numeric characters instead of coming before them. Notice in Figure 171 on page 460 that numeric characters come after alphabetic characters in the normal collating sequence. Suppose that a company started with a few departments and assigned each department a number. In their data records, they used only a two-digit field for the department number. When the company grew and the number of departments got larger than 99, the two-digit field was no longer long enough. To avoid having to change the department-number field from two to three characters in every record, the manager changed the collating sequence for that one field to use alphabetic characters after numeric characters. That is, after department 99, the manager named the departments A0, A1, and so on.

Another example is the need in some languages to insert a character such as a $ between A and B in the normal collating sequence.

Coding the Changes

To change the normal collating sequence, you code the control specification and the Translation Table and Alternate Collating Sequence Coding Sheet. Then you use the coding on the Translation Table and Alternate Collating Sequence Coding sheet to create records that actually change the normal collating sequence.

Coding the Control Specification

Column 26 of the control specification must contain S to indicate that you are changing the collating sequence.

Coding the Translation Table and Alternate Collating Sequence Coding Sheet

Figure 172 on page 462 shows the Translation Table and Alternate Collating Sequence Coding Sheet.
### To code a change in the normal collating sequence, follow these steps:

1. In the Graphic column, find the character you want to use to replace another character in the collating sequence. Note the hexadecimal value in the Entry column for the replacing character.

### Translation Table and Alternate Collating Sequence Coding Sheet

<table>
<thead>
<tr>
<th>Code</th>
<th>Graphics</th>
<th>Entry</th>
<th>Replaced By/Name</th>
<th>Place Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000010</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000100</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000101</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001000</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001001</td>
<td></td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001010</td>
<td></td>
<td>GA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001011</td>
<td></td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001100</td>
<td></td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001101</td>
<td></td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001110</td>
<td></td>
<td>J</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001111</td>
<td></td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010000</td>
<td></td>
<td>L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010001</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010010</td>
<td></td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010011</td>
<td></td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010100</td>
<td></td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010101</td>
<td></td>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010110</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010111</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011000</td>
<td></td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011001</td>
<td></td>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011010</td>
<td></td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011011</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011100</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011101</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011110</td>
<td></td>
<td>Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011111</td>
<td></td>
<td>[</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100000</td>
<td></td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100001</td>
<td></td>
<td>]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100010</td>
<td></td>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100011</td>
<td></td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100100</td>
<td></td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100101</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100110</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100111</td>
<td></td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101000</td>
<td></td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101001</td>
<td></td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101010</td>
<td></td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101011</td>
<td></td>
<td>g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101100</td>
<td></td>
<td>h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101101</td>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101110</td>
<td></td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101111</td>
<td></td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110000</td>
<td></td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110001</td>
<td></td>
<td>m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110010</td>
<td></td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110011</td>
<td></td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110100</td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110101</td>
<td></td>
<td>q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110110</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110111</td>
<td></td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111000</td>
<td></td>
<td>t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111001</td>
<td></td>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111010</td>
<td></td>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111011</td>
<td></td>
<td>w</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111100</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111101</td>
<td></td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111110</td>
<td></td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111111</td>
<td></td>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000001</td>
<td></td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000010</td>
<td></td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000011</td>
<td></td>
<td>[</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000100</td>
<td></td>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000101</td>
<td></td>
<td>]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000110</td>
<td></td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000111</td>
<td></td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001000</td>
<td></td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001001</td>
<td></td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001010</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001011</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001100</td>
<td></td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001101</td>
<td></td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001110</td>
<td></td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001111</td>
<td></td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010000</td>
<td></td>
<td>g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010001</td>
<td></td>
<td>h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010010</td>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010011</td>
<td></td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010100</td>
<td></td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010101</td>
<td></td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010110</td>
<td></td>
<td>m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010111</td>
<td></td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011000</td>
<td></td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011001</td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011010</td>
<td></td>
<td>q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011011</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011100</td>
<td></td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011101</td>
<td></td>
<td>t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011110</td>
<td></td>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011111</td>
<td></td>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100000</td>
<td></td>
<td>w</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100001</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100010</td>
<td></td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100011</td>
<td></td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100100</td>
<td></td>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100110</td>
<td></td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100111</td>
<td></td>
<td>[</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101000</td>
<td></td>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101001</td>
<td></td>
<td>]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101010</td>
<td></td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101011</td>
<td></td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101100</td>
<td></td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101101</td>
<td></td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101110</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101111</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110000</td>
<td></td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110001</td>
<td></td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110010</td>
<td></td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110011</td>
<td></td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110100</td>
<td></td>
<td>g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110101</td>
<td></td>
<td>h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110110</td>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110111</td>
<td></td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111000</td>
<td></td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111001</td>
<td></td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111010</td>
<td></td>
<td>m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111011</td>
<td></td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111100</td>
<td></td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111101</td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111110</td>
<td></td>
<td>q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111111</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 172:** Translation Table and Alternate Collating Sequence Coding Sheet

CHANGING THE COLLATING SEQUENCE
3. Code that hexadecimal value in the Replaced By column next to the character being replaced.

For example, if you want to change the normal collating sequence of a blank so that it has the same collating sequence as a zero:

1. Find the zero in the Graphic column.
2. Note that the hexadecimal value in the Entry column for zero is F0.
3. Code F0 in the Replace By column next to the blank.

Figure 173 on page 464 shows this example. The same hexadecimal value is now used for both a blank and zero. Therefore, after you make these changes, the computer treats a blank as equal to zero when it compares alphanumerical characters, checks the sequence of characters, or checks for matching fields.

If you insert a character between two consecutive characters in the normal collating sequence, you must change the collating sequence for every character that is affected by that change. For example, when you insert the dollar sign ($) between A and B, you must also change the collating sequence for every character from B through I. This example is also shown in Figure 173 on page 464.
### Translation Table and Alternate Collating Sequence Coding Sheet

#### Figure 173. Changing the Collating Sequence

<table>
<thead>
<tr>
<th>Code</th>
<th>Graphic</th>
<th>Entry</th>
<th>Required By/Take Place Of</th>
<th>Code</th>
<th>Graphic</th>
<th>Entry</th>
<th>Required By/Take Place Of</th>
<th>Code</th>
<th>Graphic</th>
<th>Entry</th>
<th>Required By/Take Place Of</th>
<th>Code</th>
<th>Graphic</th>
<th>Entry</th>
<th>Required By/Take Place Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td></td>
<td></td>
<td></td>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Notes:
- Blank and zero considered equal.
- $ takes B’s position.
- (no printable character)
- C takes D’s position.

System/36 Compatible RPG II User’s Guide and Reference
Coding the Records That Change the Collating Sequence

The changes to the normal collating sequence must be coded in records that you can enter into the computer after all the RPG specifications in your source program and after the records that translate files. Chapter 3, "Compiling an RPG II Program" explains how to enter your specifications.

These records are actually a kind of table. Unlike other tables, however, they do not need to be coded on the file description or extension specifications. Instead, they must be coded as data records.

The first record must contain ** (asterisk asterisk blank) in positions 1 through 3. You can use the remaining positions of this record for comments.

The second record, and any additional records needed to code the alternate collating sequence, must contain specific entries in the following record positions:

<table>
<thead>
<tr>
<th>Record Position</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>ALTSEQ</td>
</tr>
<tr>
<td>7-8</td>
<td>Leave these positions blank.</td>
</tr>
<tr>
<td>9-10</td>
<td>Enter the hexadecimal value of the character whose normal collating sequence is being changed. This entry is the same as the value in the Entry column on the Translation Table and Alternate Collating Coding Sheet.</td>
</tr>
<tr>
<td>11-12</td>
<td>Enter the hexadecimal value of the character that is replacing another character in the normal collating sequence. This entry is the same as your entry in the Replaced By column on the Translation Table and Alternate Collating Sequence Coding Sheet.</td>
</tr>
<tr>
<td>13-16, 17-20, 21-24, . . .</td>
<td>Use these positions in the same way as positions 9 through 12. The first two positions contain the hexadecimal value of the character replaced. The next two positions contain the hexadecimal value of the replacing character. You can use as many four-position entries as the record can hold. Do not leave any blank positions between the four-position entries. The first blank position ends the record.</td>
</tr>
</tbody>
</table>

If you are changing the collating sequence of many characters, you can use more than one record.

A record with ** (asterisk asterisk blank) in positions 1 through 3 must follow the last record that changes the normal collating sequence.
Example of a Record That Changes the Collating Sequence

To change the normal collating sequence by inserting the dollar sign ($) between A and B, as shown in Figure 173 on page 464, code the record as follows:

Table 11. Example of Data Record Entries to Change Collating Sequence

<table>
<thead>
<tr>
<th>Record Position</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>ALTSEQ</td>
</tr>
<tr>
<td>7-8</td>
<td>Blanks</td>
</tr>
<tr>
<td>9-12</td>
<td>5BC2 ($ replaces B)</td>
</tr>
<tr>
<td>13-16</td>
<td>C2C3 (B replaces C)</td>
</tr>
<tr>
<td>17-20</td>
<td>C3C4 (C replaces D)</td>
</tr>
<tr>
<td>21-24</td>
<td>C4C5 (D replaces E)</td>
</tr>
<tr>
<td>25-28</td>
<td>C5C6 (E replaces F)</td>
</tr>
<tr>
<td>29-32</td>
<td>C6C7 (F replaces G)</td>
</tr>
<tr>
<td>33-36</td>
<td>C7C8 (G replaces H)</td>
</tr>
<tr>
<td>37-40</td>
<td>C8C9 (H replaces I)</td>
</tr>
<tr>
<td>41-44</td>
<td>C9CA (I replaces an unprintable character)</td>
</tr>
</tbody>
</table>

Translating a File

Translating a file means changing the hexadecimal value of one or more characters throughout an entire program. If the character is in an input file, the computer translates (changes) the hexadecimal value when it reads the file into main storage. If the character is in an output file, the computer translates it before writing the file. If the character is in an update or combined file, the computer translates the character when it reads the file and again before it writes the file.

The usual reason for translating a file is security. You can translate input or output data to protect classified information.

Coding the Translation

To translate a file, you code the control specification and the Translation Table and Alternate Collating Sequence Coding Sheet. Then you use the coding on the Translation Table and Alternate Collating Sequence Coding Sheet to create records that actually change the characters.

Coding the Control Specification
Column 43 of the control specification must contain F to indicate that you are translating a file.

Coding the Translation Table and Alternate Collating Sequence Coding Sheet
Figure 172 on page 462 shows the Translation Table and Alternate Collating Sequence Coding Sheet.
To code a character for translation, follow these steps:

1. In the Graphic column, find the character you want to use as the translation for another character.
2. Note the hexadecimal value in the Entry column for the character used as the translation.
3. Code that hexadecimal value in the Replaced By column next to the character being translated.

**Coding the Records That Translate a File**

To tell the computer the files to translate, you must code records that you can enter into the computer after all the RPG specifications in your source program but before the records that change the normal collating sequence. Chapter 3, “Compiling an RPG II Program” explains how to enter your specifications.

These records for translating a file are actually a kind of table. Unlike other tables, however, they do not need to be coded on the file description or extension specifications. Instead, they must be coded as data records.

The first record must contain **b (asterisk asterisk blank) in positions 1 through 3. You can use the remaining positions of this record for comments.

The second record, and any additional records needed to code the translation, must contain specific entries in the following record positions:

<table>
<thead>
<tr>
<th>Record Position</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6 (to translate all files)</td>
<td>Enter *FILES to tell the compiler to translate all input, output, update, and combined files. Leave positions 7 and 8 blank.</td>
</tr>
<tr>
<td>1-8 (to translate a specific file)</td>
<td>Enter the name of the specific file translated.</td>
</tr>
<tr>
<td>9-10</td>
<td>Enter the hexadecimal value of the character that is being translated. This entry is the same as the value in the Entry column on the Translation Table and Alternate Collating Sequence Coding sheet.</td>
</tr>
<tr>
<td>11-12</td>
<td>Enter the hexadecimal value of the character that is translating another character. This entry is the same as your entry in the Replaced By column on the Translation Table and Alternate Collating Sequence Coding Sheet.</td>
</tr>
<tr>
<td>13-16, 17-20, 21-24, . . . , 93-96</td>
<td>Use these positions in the same way as positions 9 through 12. The first two positions contain the hexadecimal value of the character translated. The next two positions contain the hexadecimal value of the translating character. You can use as many four-position entries as the record can hold. Do not leave any blank positions between the four-position entries. The first blank position ends the record. If you need more positions to code the translations, you can use more than one record.</td>
</tr>
</tbody>
</table>
Example of File Translation

A department store uses sales slips that contain the wholesale and retail price of each item. To keep the wholesale prices confidential, the store translates the numbers into letters. In output files, it uses the letters in the code name BUCKINGHAM to represent the numbers 1 through 9 and 0. In input files, it translates the letters back into numbers so that the computer can do calculations on the wholesale prices. Figure 174 on page 469 shows how to code the file translation coding sheet for this example.
Chapter 17. Changing the Hexadecimal Value of Characters

The record to translate these files looks like this:

```
843EDdgq
```

This is the hexadecimal value of the character to be translated.

```
0010110 66
```

This is the hexadecimal value of the character that will be substituted for the character that is to be translated.
### Table 13. Example of Data Record Entries that Translate a File

<table>
<thead>
<tr>
<th>Record Position</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>*FILES (All files are translated.)</td>
<td></td>
</tr>
<tr>
<td>7-8</td>
<td>Blanks</td>
<td></td>
</tr>
<tr>
<td>9-12</td>
<td>C2F1</td>
<td>(B is translated into 1 at input. 1 is translated into B at output.)</td>
</tr>
<tr>
<td>13-16</td>
<td>E4F2</td>
<td>(U is translated into 2 at input. 2 is translated into U at output.)</td>
</tr>
<tr>
<td>17-20</td>
<td>C3F3</td>
<td>(C is translated into 3 at input. 3 is translated into C at output.)</td>
</tr>
<tr>
<td>21-24</td>
<td>D2F4</td>
<td>(K is translated into 4 at input. 4 is translated into K at output.)</td>
</tr>
<tr>
<td>25-28</td>
<td>C9F5</td>
<td>(I is translated into 5 at input. 5 is translated into I at output.)</td>
</tr>
<tr>
<td>29-32</td>
<td>D5F6</td>
<td>(N is translated into 6 at input. 6 is translated into N at output.)</td>
</tr>
<tr>
<td>33-36</td>
<td>C7F7</td>
<td>(G is translated into 7 at input. 7 is translated into G at output.)</td>
</tr>
<tr>
<td>37-40</td>
<td>C8F8</td>
<td>(H is translated into 8 at input. 8 is translated into H at output.)</td>
</tr>
<tr>
<td>41-44</td>
<td>C1F9</td>
<td>(A is translated into 9 at input. 9 is translated into A at output.)</td>
</tr>
<tr>
<td>45-48</td>
<td>D4F0</td>
<td>(M is translated into 0 at input. 0 is translated into M at output.)</td>
</tr>
</tbody>
</table>
Chapter 18. Control Specification

A control specification describes your program and the computer system to the RPG compiler. One control specification is required for each source program. The control specification should always be the first specification in the program. Write the control specification on the first line of the RPG Control and File Description Specifications sheet (see Figure 175).

If you omit the control specification from the source program, the compiler creates a blank control specification.

See the individual column descriptions for the meaning of blank entries.

Figure 175. RPG Control and File Description Specifications
Columns 1-2 (Page)

Entry | Explanation
--- | ---
Blank | No page number is used.
01-99 | Page number.

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

Columns 3-5 (Line)

Entry | Explanation
--- | ---
Blank | No line number is used.
Any numbers | Line numbers.

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

The line numbers used need not be consecutive, but should be in ascending order.

Column 6 (Form Type)

An H must appear in column 6 to identify this line as the control (header) specification.

Column 7 (Comments)

Entry | Explanation
--- | ---
* | Comment line

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.

Columns 7-9

Columns 7 through 9 (the former Size to Compile) have no relevance on an AS/400 system. Leave them blank.
Columns 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing. The /EJECT specification is not printed on the compiler listing.</td>
</tr>
</tbody>
</table>

Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74. A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information. The /TITLE statement causes an eject to the next page before the title is printed. The /TITLE statement is not printed on the compiler listing.</td>
</tr>
</tbody>
</table>

Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SPACEbn</td>
<td>Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed. One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced. /SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.</td>
</tr>
</tbody>
</table>

Column 10

Column 10 (the former Object Output) has no relevance on the AS/400 system. Leave it blank.
Column 11

Column 11 (the former Listing Options) has no relevance on the AS/400 system. Leave it blank.

Columns 12-14

Columns 12 to 14 (the former Size to Run) have no relevance on the AS/400 system. Leave them blank.

Column 15 (Debug)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>A DEBUG operation is not processed. Compiler-generated symbols are not placed in the symbol table.</td>
</tr>
<tr>
<td>1</td>
<td>A DEBUG operation is processed. Compiler-generated symbols are placed in the symbol table.</td>
</tr>
</tbody>
</table>

Use column 15 to indicate if the DEBUG operation is to be processed. To process a DEBUG operation:

- A 1 must appear in column 15 when the source program is compiled.
- The DEBUG operation code must be used in the calculation specifications.

The DEBUG entry also controls the contents of the symbol table that is produced with the program. The symbol table is printed with the program dump. If position 15 contains a 1, the compiler-generated symbols, starting with a period (.), are placed in the symbol table. If position 15 is blank, the compiler-generated symbols are not placed in the symbol table. You can then use compiler-generated symbols to debug a program.

See “DEBUG (Debug)” on page 740 for more information.

Columns 16-17

Columns 16 and 17 are not used. Leave them blank.
Column 18 (Currency Symbol)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Defaults to $ as the currency symbol.</td>
</tr>
<tr>
<td>Any other character</td>
<td>This character is used as the currency symbol. If you want a fixed or floating currency symbol, you must code this symbol in edit words or with the edit codes you want to use. Any character may be specified as the currency symbol except the following characters, which have a special meaning in edit words or edit codes: 0 (zero) * (asterisk) , (comma) &amp; (ampersand) . (decimal point) - (minus sign) C (letter C) R (letter R)</td>
</tr>
</tbody>
</table>

Columns 19-20 (Date Option)

Column 19 (Date Format)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>If column 21 is blank, the default is month/day/year. If column 21 contains a D, I, or J, the default is day/month/year.</td>
</tr>
<tr>
<td>M</td>
<td>Month/day/year.</td>
</tr>
<tr>
<td>D</td>
<td>Day/month/year.</td>
</tr>
<tr>
<td>Y</td>
<td>Year/month/day.</td>
</tr>
</tbody>
</table>

Use column 19 to match the date format for UDATE. The date format entered in column 19 should be the same format as the program date. For example, if columns 19 and 21 are blank, the program date is mm/dd/yy. If column 19 is blank and column 21 contains a D, the program date is dd/mm/yy. If you specify the date in mm/dd/yy format and the program date in the system is in dd/mm/yy format, you will work with the wrong date.

If data containing the UDATE field is sent to, or used by, another system, the UDATE format must be yy/mm/dd.

For a description of the program date, see the System Reference for the System/36 Environment.

Column 20 (Date Edit)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>A period (.) is assumed when column 19 contains a Y or a blank and column 21 contains I or J. Otherwise, a slash (/) is assumed.</td>
</tr>
</tbody>
</table>
& A blank separates the date field.
Any other character The character entered separates the edited date field.

Use column 20 to specify the type of edited output that appears for the Y edit code, which is specified on the output specifications. For an example of how the entries in columns 19 through 21 affect the editing of date fields, see “Column 38 (Edit Codes)” on page 617 in Chapter 25, “Output Specifications.”

### Column 21 (Inverted Print)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Decimal periods are used for numeric literals and editing. UPDATE format is mmdyy if column 19 is blank. If columns 19 and 20 are blank, a slash (/) is used for the Y edit code.</td>
</tr>
<tr>
<td>I</td>
<td>Decimal commas are used for numeric literals and editing. UPDATE format is ddmmyy if column 19 is blank. If columns 19 and 20 are blank, a period (.) is used for the Y edit code.</td>
</tr>
<tr>
<td>J</td>
<td>J is the same as I except zero is written to the left of the decimal comma when the field contains a fraction. Nondecimal edited fields print with a zero in the low-order (units) position.</td>
</tr>
<tr>
<td>D</td>
<td>D is the same as blank except the UPDATE format is ddmmyy if column 19 is blank.</td>
</tr>
</tbody>
</table>

Use column 21 to specify the constants used with RPG edit codes that are entered on the output specifications. Decimal period means that numbers are edited with a period before the fraction (183.55) and with a comma denoting thousands (1,435). Decimal comma means that numbers are edited with a comma before the fraction (183,55) and with a period denoting thousands (1.435).

For information on how the entries in column 21 are used to format numeric data, see “Column 38 (Edit Codes)” on page 617.

### Columns 22-25

Columns 22 through 25 are not used. Leave them blank.

### Column 26 (Alternate Collating Sequence)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Normal collating sequence is used.</td>
</tr>
<tr>
<td>S</td>
<td>Alternate collating sequence is used.</td>
</tr>
</tbody>
</table>

Use column 26 only to alter the normal collating sequence for alphanumerical compare operations, sequence checking, or match fields. For more information, see Chapter 17, “Changing the Hexadecimal Value of Characters.”
Columns 27-36

Columns 27 through 36 are not used. Leave them blank.

Column 37 (Inquiry)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank or I</td>
<td>The program, when interrupted, will not allow the person using the display station to enter new procedures or commands (does not allow option 1 for an inquiry request).</td>
</tr>
<tr>
<td>B</td>
<td>The program, when interrupted, will allow the person using the display station to enter new procedures or commands (does allow option 1 for an inquiry request).</td>
</tr>
</tbody>
</table>

Use column 37 to specify if a running program can be interrupted to allow another program to run. The person using the display station requests an interruption (called an inquiry request) by pressing the Sys Req key. The procedure or command statements for the interrupting program must be entered from the display station after selecting option 1 (system request display).

The program loaded following an inquiry request (the interrupting program) can have an I, B, or blank in column 37 of its control specification. However, even if it has a B in column 37, the interrupting program cannot be interrupted to allow another program to run.

If column 37 contains a B, the inquiry function of the System/36 Environment allows the person using the display station to interrupt a program that is currently using the display station and to enter new procedures or commands. If column 37 contains any of the valid entries, the person can set the inquiry latch for the inline inquiry subroutine (SUBR95), cancel a single requester terminal (SRT) program that the operator started, or release the display station from a multiple requester terminal (MRT) program.

For more information on inquiry, including restrictions on the use of system utilities in inquiry mode, see the *System Reference for the System/36 Environment*.

File Sharing

An inquiry program can get active input, update, and add files. However, an inquiry program cannot get indexed sequential add file types or output files. The DISP-SHR parameter must be specified on the OCL FILE statement for each file shared in both the interrupted and the inquiry programs.

For a description of the valid file-sharing combinations, see the *Concepts and Programmer’s Guide for the System/36 Environment*.

Inline Inquiry Subroutine (SUBR95)

The IBM-written subroutine SUBR95 can be used to process an inquiry if the RPG program is not a MRT program (that is, if the MRTMAX parameter on the control language COMPILE statement was 0 when the program was compiled). Column 37 can be blank or contain an I or B. See the *Concepts and Programmer’s Guide for the System/36 Environment* for restrictions on the inquiry function.
The linkage to SUBR95 must be specified on the calculation specifications at every point in the program where a check is made for an inquiry request. The EXIT SUBR95 operation must be followed by only one RLABL (see Figure 176 on page 478). The indicator specified in columns 45 and 46 of the RLABL operation must be an RPG indicator. For a detailed discussion of this linkage, see “Calling External Subroutines or Programs” on page 721.

When SUBR95 is called, it checks to determine if the inquiry latch was set. If it was (that is, the operator selected option 20, Set inquiry condition for S/36 program, in response to the System Request display), the indicator specified in the RLABL operation is turned on, and the inquiry request is reset. This indicator can then be used to condition further calculation and output operations.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C EXIT SUBR95
C RLABL INXX

Figure 176. Linkage for SUBR95

Columns 38-40

Columns 38 through 40 are not used. Leave them blank.

Column 41 (1P Forms Position)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>First line is printed only once.</td>
</tr>
<tr>
<td>1</td>
<td>First line can be printed repeatedly.</td>
</tr>
</tbody>
</table>

Use column 41 only when the first output line is written to a PRINTER file. If the program contains more than one PRINTER file, the first-page indicator (1P) entry in column 41 applies to each PRINTER file that has first-page (1P) output.

When forms are first put in the printer, they may not be in the right position. Sometimes several lines must be printed to determine the correct position of the form. If 1P forms position is specified, the system prints the first line of output and issues a message. The person using the display station can then line up the forms and select the option to try printing the line again or to continue printing. The 1P forms specification is also valid if the output is spooled. The page counter is not increased until the forms have been positioned correctly.

The 1P forms position specification can be overridden on the OCL PRINTER statement, or forms alignment can be specified on the PRINTER statement.
Column 42

Column 42 is not used. Leave it blank.

Column 43 (File Translation)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No file translation is needed.</td>
</tr>
<tr>
<td>F</td>
<td>Input, output, update, or combined files are to be translated.</td>
</tr>
</tbody>
</table>

Use column 43 only when information contained in an input, output, update, or combined file is in a character code different from the character code used by System/36.

For more information, see “Translating a File” on page 466 in Chapter 17, “Changing the Hexadecimal Value of Characters.”

Column 44

Column 44 is not used. Leave it blank.

Column 45 (Nonprint Characters)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The program halts if the last line printed contained an unprintable character.</td>
</tr>
<tr>
<td>1</td>
<td>The program does not halt for unprintable characters.</td>
</tr>
</tbody>
</table>

Use column 45 to bypass halts for unprintable characters. This column applies only to PRINTER files.

All characters are represented in the system by a hexadecimal value, which is a numeric code. If a hexadecimal value is formed during a calculation that is not in the printer character set and that character is printed, the program halts after printing the line. In the printed line, the unprintable characters are replaced with blanks.

To bypass this halt, enter a 1 in column 45. An unprintable character is then replaced with a blank, and no halt occurs. Note, however, that your output is not correct, and, by bypassing the halt, the incorrect output may not become known (for example, when a packed key field is printed or when a nonprintable field is built by calculation specifications).

Columns 46-54

Columns 46 through 54 are not used. Leave them blank.
**Column 55**

Column 55 (the former Subprogram option) has no relevance on the AS/400 system. Leave it blank.

**Column 56**

Column 56 is not used. Leave it blank.

**Column 57 (Transparent Literal)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No transparent literals or constants are present in the program.</td>
</tr>
<tr>
<td>1</td>
<td>Transparent literals or constants can be present in the program.</td>
</tr>
</tbody>
</table>

The transparent literal option must be specified if your program uses double-byte data. A transparent literal or constant is one that begins with an apostrophe followed immediately by the shift-out (S/O) character (hexadecimal 0E), and ends with the shift-in (S/I) character (hexadecimal 0F) followed immediately by an apostrophe.

If the transparent literal option is specified and a literal or constant is found that begins with an apostrophe immediately followed by the S/O character, the RPG compiler checks for a valid transparent literal or constant. The following conditions diagnose a literal or constant as an invalid transparent literal or constant:

- A second S/O character is found before the S/I character.
- An odd number of one-byte characters are found between the S/O and S/I characters.
- The S/I character is not immediately followed by the ending apostrophe.

If a literal or constant is an invalid transparent literal or constant, it is rechecked as an alphanumeric literal or constant.

Transparent literals and constants are not checked for embedded apostrophes.

For more information about double-byte data, see Chapter 28, “Using Double-Byte Data.”

**Columns 58-74**

Columns 58 through 74 are not used. Leave them blank.

**Columns 75-80 (Program Identification)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to RPGOBJ.</td>
</tr>
<tr>
<td>Any valid program</td>
<td>The first character of the program identification must be alphabetic and</td>
</tr>
<tr>
<td>name</td>
<td>cannot be #, $, or @. The remaining characters must be alphanumeric; however, no special</td>
</tr>
</tbody>
</table>
characters can be used, and blanks must not appear between characters.

Use columns 75 through 80 to assign a name to your program. The compiler uses the program name to catalog the program in the library directory.

If the program contains a CONSOLE device, the compiler also uses this program identification to name the display file for the program. The display file is created by RPG only for CONSOLE files; however, the name is created for both CONSOLE and WORKSTN files. This name is used by RPG; therefore, you must create your own display file with this or an alternative name for WORKSTN files. For the display file name, the compiler uses the name specified as the value of the FMTS continuation-line option. If the FMTS continuation-line option is not specified, the compiler uses the characters specified in columns 75 through 80 of the control specification (the program identification) and adds the characters FM to the end of the program identification. FM is added to the end of the program identification regardless of its length, and the resulting name contains no blanks.

If a cross-reference listing is created for the program, this program identification is also used to identify the listing.

**Note:** Columns 75 to 80 are used by the RPGC and AUTOC procedures running in the System/36 Environment.

In normal AS/400 system mode, the program name is taken from columns 75 through 80 only if PGM(*CTLSPEC) has been specified in the CL commands CRTS36RPG or CRTS36RPT. If the PGM parameter of the CRTS36RPG or the CRTS36RPT command contains *CTLSPEC, and columns 75 to 80 contain blanks, the name of the program defaults to RPGOBJ.
Chapter 19. File Description Specifications

File description specifications describe each file used by a program. One file description specification is required for each file, and a maximum of 50 files can be described for each program.

Write the file description specifications on the Control and File Description Specifications sheet (see Figure 177).

**Figure 177. RPG Control and File Description Specifications**

File Description Charts

Figure 178 through Figure 189 show the file description specification entries for DISK files (which are presented by file organization and processing method), WORKSTN files, PRINTER files, SPECIAL files, CONSOLE files, KEYBOARD files, CRT files, and BSCA files. When you use the charts, keep the following in mind:

- The entries in the chart must be made for the processing method and type of file described on that line.
- The shaded columns must be blank for the file described on that line.
• The unshaded columns with no entries may be required or optional, but cannot be indicated on the chart because the entries represent information that changes from program to program.

How to Use the Charts

As an example, if you are updating an indexed DISK file using the CHAIN operation code, see Figure 180 on page 486, for indexed DISK files, and refer to random processing by CHAIN operation code. Then choose the chained or full-procedural update file with or without record addition.

In this example, the following columns are required but may change from one program to another: file name, record length, length of key field, and key field starting location. Optional entries are line, block length, and file condition.

<table>
<thead>
<tr>
<th>Type of Processing</th>
<th>Consecutive</th>
<th>Random</th>
<th>Consecutive and/or Random</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
<tr>
<td>by CHAIN (delete-capable)</td>
<td>by Addrout</td>
<td>by Addrout</td>
<td>by READ, READP and/or CHAIN (delete-capable)</td>
<td>Add records only</td>
</tr>
</tbody>
</table>
To insert change records in a direct file, define the file as an update file processed consecutively or as an update file processed randomly by the CHAIN operation code.

Figure 179. Processing Methods for Direct DISK Files
### Figure 180. Processing Methods for Indexed DISK Files (Using the Index)

---

1. Sequential processing by key or limits must use the file index, which is always arranged in ascending sequence. When an indexed file is processed record by record from beginning to end, the sequential by key method is used to process the file through the index.

2. If chained files are processed by key, column 31 should contain an A; however, if chained files are processed by relative record number, columns 31 and 32 must be blank.
Record address files containing relative record numbers can be associated with indexed, sequential, or direct DISK files.

Record address files containing key-field limits can be associated only with indexed DISK files, but can be a DISK or CONSOLE file. (See chart for CONSOLE files.)

**Figure 181. Record Address Files Located on Disk**

**Figure 182. WORKSTN Files**

**Figure 183. PRINTER Files**
**Figure 184. SPECIAL Files**

Input file records are displayed on the display screen when keyed into the program.

**Figure 185. CONSOLE Files**
If a KEYBORD file is specified as a primary input file, no other input files in the program can be specified as primary or secondary files.

Input data entered from the KEYBORD device must be defined in calculation specifications for a KEY operation.

No input specifications can be used for KEYBORD files.

*Figure 186. KEYBORD Files*

---

*Figure 187. CRT Files*
### FILE DESCRIPTION CHARTS

#### Figure 188. BSCA Files

<table>
<thead>
<tr>
<th>Line</th>
<th>File Type</th>
<th>Filename</th>
<th>Mode of Processing</th>
<th>File Description</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>Name of Label Exit</th>
<th>Extent Exit for DAM</th>
<th>Storage Index</th>
<th>Labels S/N/E/M</th>
<th>Option Entry</th>
<th>Form Type</th>
<th>Record Type</th>
<th>Overflow Indicator</th>
<th>Type Rewind</th>
<th>Number of Extents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>P F</td>
<td>BSCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>I F</td>
<td>BSCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>D F</td>
<td>BSCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>0 F</td>
<td>BSCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 189. File Description Specifications for IBM-Supplied Subroutines

<table>
<thead>
<tr>
<th>Line</th>
<th>File Type</th>
<th>Filename</th>
<th>Mode of Processing</th>
<th>File Description</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>Name of Label Exit</th>
<th>Extent Exit for DAM</th>
<th>Storage Index</th>
<th>Labels S/N/E/M</th>
<th>Option Entry</th>
<th>Form Type</th>
<th>Record Type</th>
<th>Overflow Indicator</th>
<th>Type Rewind</th>
<th>Number of Extents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>I F</td>
<td>SPECIAL SUBRO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

490 System/36-Compatible RPG II User’s Guide and Reference
### Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

### Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any</td>
<td>Line numbers</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run your program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

The line numbers used need not be consecutive, but should be in ascending order.

### Column 6 (Form Type)

An F must appear in column 6 to identify this line as a file description specification.

### Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.
### Columns 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.

### Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74.</td>
</tr>
</tbody>
</table>

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information.

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

### Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SPACEbn</td>
<td>Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed.</td>
</tr>
</tbody>
</table>

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.

### Columns 7-14 (Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid filename</td>
<td>Every file used in a program must have a separate name. The first character must be alphabetic. The remaining characters can be any combination of alphabetic and numeric characters; however, special characters are not allowed. Blanks cannot appear between characters in the filename. The filename can be from one to eight characters long and must begin in column 7.</td>
</tr>
</tbody>
</table>
Use columns 7 through 14 to assign a unique name to every file used in your program, with the following exceptions:

- Compile-time tables and arrays do not require a filename.
- If multiple tables or arrays are read in at prerun time from the same device, multiple filenames are required.

For naming tables and arrays, see “Columns 27-32 (Array or Table Name)” on page 521 in Chapter 20, “Extension Specifications.”

**Column 15 (File Type)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Input file</td>
</tr>
<tr>
<td>O</td>
<td>Output file</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you code an O in column 15, column 16 must be left blank.</td>
</tr>
<tr>
<td>U</td>
<td>Update file</td>
</tr>
<tr>
<td>C</td>
<td>Combined (input and output) file</td>
</tr>
</tbody>
</table>

The file type indicates how a program will use a file (for input or for output). You code the file type in column 15 of the file description specifications. The total number of files of all types used in a program cannot exceed 50.

Table 14 page=no. shows the types of files that each device type can use.

<table>
<thead>
<tr>
<th>Device Type</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>Input, Output, Update</td>
</tr>
<tr>
<td>WORKSTN</td>
<td>Combined</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Output</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Input, Output, Update, Combined</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Input</td>
</tr>
<tr>
<td>KEYBORD</td>
<td>Input</td>
</tr>
<tr>
<td>CRT</td>
<td>Output</td>
</tr>
<tr>
<td>BSCA</td>
<td>Input (receive), Output (transmit)</td>
</tr>
</tbody>
</table>
COLUMNS 16 (FILE DESIGNATION)

Input Files
An input file contains records that the program reads.

Output Files
An output file contains records that the program writes.

Update Files
An update file is both an input file and an output file. The program reads a record from an update file, changes the data in some fields in the record, and writes the record back to the same place in the same file from which it was read. When an update file is processed, the output records contain both the changed and the unchanged fields from the input records.

Combined Files
A combined file is also both an input file and an output file. However, when a combined file is processed, the output records contain only the fields described on the output specifications. That is, the output records do not always contain the same fields as the input records.

Column 16 (File Designation)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Primary file</td>
</tr>
<tr>
<td>S</td>
<td>Secondary file</td>
</tr>
<tr>
<td>F</td>
<td>Full-procedural file</td>
</tr>
<tr>
<td>C</td>
<td>Chained file</td>
</tr>
<tr>
<td>R</td>
<td>Record address file</td>
</tr>
<tr>
<td>T</td>
<td>Table file (prerun-time arrays or tables)</td>
</tr>
<tr>
<td>D</td>
<td>Demand file</td>
</tr>
<tr>
<td>Blank</td>
<td>Output file.</td>
</tr>
</tbody>
</table>

Use column 16 to further identify the use of input, combined, and update files. Leave column 16 blank for all output files (files that have an O coded in column 15).

Figure 190 on page 495 shows the file designations that each device type and file type can use.
<table>
<thead>
<tr>
<th>Device Type</th>
<th>File Type</th>
<th>File Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>Input</td>
<td>Primary, Secondary, Chained, Record address, Table or array, Demand, Full-procedural</td>
</tr>
<tr>
<td></td>
<td>Output</td>
<td>Chained or blank</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Primary, Secondary, Chained, Demand, Full-procedural</td>
</tr>
<tr>
<td>WORKSTN</td>
<td>Combined</td>
<td>Primary, Demand</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Input</td>
<td>Primary, Secondary, Demand</td>
</tr>
<tr>
<td></td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Primary, Secondary, Demand</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>Primary, Secondary, Demand</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Input</td>
<td>Primary, Secondary, Demand, Record address</td>
</tr>
<tr>
<td>KEYBOARD</td>
<td>Input</td>
<td>Primary, Demand</td>
</tr>
<tr>
<td>CRT</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>BSCA</td>
<td>Input</td>
<td>Primary, Secondary, Demand</td>
</tr>
<tr>
<td></td>
<td>Output</td>
<td></td>
</tr>
</tbody>
</table>

Figure 190. File Designations That Each Device Type and File Type Can Use

**Primary Files**

The primary file is the main file from which the program reads input records. A program can have no more than one primary file, but it does not have to have any.

The primary file can be an input, update, or combined file. It can use any device except CRT or PRINTER.
Secondary Files

Secondary files are used only in programs that use multifile processing. For more information on record selection for multifile processing, see Chapter 12, “Primary/Secondary/Multifile Processing.”

A secondary file can be an input, update, or combined file. It can use the DISK, CONSOLE, SPECIAL, or BSCA device. Secondary files are processed in the order in which they are coded on the file description specifications. If no primary file is specified and one or more secondary files are specified, the first secondary file is assigned as the primary file. If WORKSTN is specified as the device for a primary file, no files in the program can be specified as secondary files.

Full-Procedural Files

A full-procedural file is a combination of a chained file and a demand file. A full-procedural file does not use the normal RPG cycle for input. A program can read records from a full-procedural file only when a CHAIN, READ, READE or READP operation occurs in calculations.

A full-procedural file can be an input or an update file, and it must use the DISK device.

Chained Files

A chained file does not use the normal RPG program cycle for input. Instead, input occurs only when the program uses the CHAIN operation in calculations. The CHAIN operation reads input records randomly (that is, in no particular order) or loads a direct file that does not allow deletions. A chained file can be an input, output, or update file. It can use only the DISK device.

Record Address Files

A record address file contains either key-field limits or relative record numbers of records in a DISK file. (Key-field limits and relative record numbers are explained later in this chapter.) By providing these key-field limits or relative record numbers to the program, a record address file tells the program the records to read from the DISK file and the order in which to read them. A program can use no more than one record address file. Record address files must be further defined on extension specifications.

A record address file must be an input file. If it contains key-field limits, it must use the DISK or CONSOLE device and can be used only with indexed files. If it contains relative record numbers, it must use only the DISK device and can be used with sequential, direct, or indexed files. Record address files that contain relative record numbers are called addrout (address output) files, and they are produced by a sort program.

Array or Table Files

An array or table file is an input file that contains prerun-time array or table entries. Array or table files must be sequential files and must use the DISK device. When array or table files are read while the program is running, the program reads all the entries from the array or table before it begins to process records.

For more information about loading prerun-time arrays or tables, see Chapter 14, “Using Arrays and Tables.”
Demand Files

A demand file can be an input, update, or combined file. It can use any device except CRT and PRINTER.

A demand file does not use the normal RPG program cycle for input. Instead, the program reads a demand file only when the READ operation occurs in calculations (or when the KEY operation occurs if the records come from a KEYBORD device).

Column 17 (End Of File)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The program can end before all records from the file are processed. However, if column 17 is blank for all files, all records from every file must be processed before the program can end. This column must be blank for WORKSTN or KEYBORD files.</td>
</tr>
<tr>
<td>E</td>
<td>All records from the file must be processed before the program can end. This entry is not valid for files processed by record address files.</td>
</tr>
</tbody>
</table>

Use column 17 to indicate if the program can end before all records from the file are processed. Column 17 applies only to primary and secondary files.

Column 17 can be used only for input, update, or combined files used as primary, secondary, or record address files. The devices associated with column 17 are DISK and CONSOLE. End of file for CONSOLE files occurs when the person using the display station presses function key 12.

A program that does processing with more than one input file could reach the end of one file before reaching the end of the others. Therefore, an entry in column 17 indicates if the program is to continue reading records from the other files or is to end.

If the records from all files must be processed, column 17 must be blank or contain E's for all files.

Note: An entry cannot be made in column 17 for files assigned to the KEYBORD and WORKSTN devices. To end the program with a primary file assigned to the KEYBORD device, the last record (LR) indicated must be set on by calculation specifications.

Column 18 (Sequence)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No sequence checking is done. This column must be blank for a WORKSTN file.</td>
</tr>
<tr>
<td>A</td>
<td>Sequence checking is done. Records in the file are in ascending order.</td>
</tr>
<tr>
<td>D</td>
<td>Sequence checking is done. Records in the file are in descending order.</td>
</tr>
</tbody>
</table>

Use column 18 to indicate if the program is to check the sequence of records. Column 18 applies to input, update, or combined files used as primary or secondary files. Sequence checking can be done for DISK files (except those proc-
essed randomly) and for CONSOLE files. Use columns 61 and 62 of the input
specifications to identify the record fields containing the sequence information.

Sequence checking is required when match fields are used in the records from the
file. When a record from a matching input file is out of sequence, error message
RPG9032 File [file name] contains a record not in sequence, is displayed. The
operator has three options:

- Bypass the record out of sequence and read the next record from the same
  file.
- Bypass the record out of sequence, turn on the last-record (LR) indicator, and
  process all end-of-job and final-total procedures.
- Cancel the entire program.

If column 18 contains an entry and matching records are specified, the entry in
column 18 must be the same for all files. If column 18 is left blank and matching
records are specified, then ascending order is assumed for a primary file, and the
sequence of the primary file is assumed for all secondary files.

---

### Column 19 (File Format)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>F or blank</td>
<td>Fixed-length records</td>
</tr>
</tbody>
</table>

An F in column 19 indicates that all records in the file are of the same length. If
this column is blank, F is assumed.

---

### Columns 20-23

Columns 20 to 23 (the former Block Length) have no relevance on an AS/400
system. Leave them blank.²

---

### Columns 24-27 (Record Length)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9999</td>
<td>Record length for DISK or SPECIAL files.</td>
</tr>
<tr>
<td>1-9999</td>
<td>Length of largest input or output record for a WORKSTN file.</td>
</tr>
</tbody>
</table>
| 1-225 | Length of largest output record for PRINTER files. (Entries from 133
  through 225 should only be used for printers with 225 print positions.) |
| 2-1518 | Record length for CONSOLE files. |
| 1-79 | Length of largest field keyed for KEYBORD files. |
| 1-79 | Length of largest output record for CRT files. |

---

² These fields must be specified on the appropriate configuration descriptions for the case of the BSCA file on the AS/400 system.
These fields are syntax-checked for compatibility with System/36, but are ignored on the AS/400 system. For more information
see page 315.
2-58 Twice the record address field length for a record address file assigned to the CONSOLE device.

1-4075 Record length for BSCA files.

Use columns 24 through 27 to indicate the length of the records in a file. An entry must be made for each file, and the entry depends on the device named for the file. Entries in these columns must end in column 27, and leading zeros can be omitted (see Figure 191).

All records in one file must be the same length. (For update files, the length of the record after the record is updated must be the same as it was before the record was updated.) The maximum length allowed depends upon the device assigned to the file (see Figure 191). The record length specified can be shorter than the maximum length allowed for the device but not longer.

The record length for KEYBOARD files should be the length of the largest field typed in (that is, the record length equals the largest field length specified in columns 49 through 51 of the calculation specifications when the KEY operation code is used). If the KEY operation is used to display a message, you must also consider the length of the message when you specify the record length for the KEYBORD file. The maximum alphameric field length is 79 characters, and the maximum numeric field length is 15 characters. If the record length specified for a KEYBORD file is 40 or less, a display of six lines with 40 characters per line is centered both vertically and horizontally on the display screen. If the record length is greater than 40, the display consists of 24 lines with 79 characters per line.\(^5\)

<table>
<thead>
<tr>
<th>Columns 40 through 46 (Device)</th>
<th>Columns 24 through 27 (Record Length)</th>
<th>Maximum Record Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>Record length</td>
<td>9999</td>
</tr>
<tr>
<td>WORKSTN</td>
<td>Length of longest input or output record</td>
<td>9999</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Record length</td>
<td>225</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Record length</td>
<td>9999</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Record length</td>
<td>1518</td>
</tr>
<tr>
<td></td>
<td>Record length (RAF file)</td>
<td>58</td>
</tr>
<tr>
<td>KEYBOARD</td>
<td>Length of largest field to be keyed</td>
<td>79 (alphameric)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 (numeric)</td>
</tr>
<tr>
<td>CRT</td>
<td>Length of longest output record</td>
<td>79</td>
</tr>
<tr>
<td>BSCA</td>
<td>Record length</td>
<td>4075</td>
</tr>
</tbody>
</table>

*Figure 191. Record Length Entries*
### Column 28 (Mode of Processing)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Blank | Consecutive  
Sequential by key field  
Random by relative record number  
Random by key field  
Random by address output file |
| L     | Sequential within key-field limits |
| R     | Random by key field  
Random by address output file  
Direct file load (random load) |

Use column 28 to indicate the method by which records are read from the file, or to indicate that a direct file load (random load) is to take place.

For DISK files specified as primary, secondary, demand, chained, or full-procedural, the possible processing methods depend upon the organizations of the files. For the other types of files, consecutive processing is the only possible method.

Column 31 further identifies the access method for the program. See “Column 31 (Record Address Type)” on page 502.
<table>
<thead>
<tr>
<th>File Organization</th>
<th>Possible Processing Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary, Secondary, or Demand Files</strong></td>
<td></td>
</tr>
</tbody>
</table>
| Sequential | Consecutively  
Randomly by address output file |
| Direct | Consecutively  
Randomly by address output file (except demand files) |
| Indexed | Sequentially by key field  
Sequentially within key-field limits  
Randomly by address output file  
Consecutively (not using the index) |
| **Chained Files** | |
| Sequential | Randomly by relative record number |
| Direct | Randomly by relative record number |
| Indexed | Randomly by key field  
Randomly by relative record number (not using the index) |
| **Full-Procedural Files** | |
| Sequential | Consecutively  
Randomly by address output file  
Randomly by relative record number |
| Direct | Consecutively  
Randomly by address output file  
Randomly by relative record number |
| Indexed | Sequentially by key field  
Sequentially within key-field limits  
Randomly by address output file  
Randomly by key field  
Consecutively (not using the index)  
Randomly by relative record number (not using the index) |

*Figure 192. Possible Processing Methods for DISK Files*

**Columns 29-30 (Length of Key Field or Record Address Field)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Records are read consecutively</td>
</tr>
<tr>
<td>1-99</td>
<td>Length of key field or relative record number</td>
</tr>
</tbody>
</table>
COLUMNS 31 (RECORD ADDRESS TYPE)

Use columns 29 and 30 to indicate:

- The length in bytes of the key fields in indexed files and record address files
- The total length in bytes of the noncontiguous key fields if a noncontiguous key is being used
- The length in characters of the relative record numbers in address output files, which is always three.

Columns 29 and 30 apply only to indexed files and record address files.

All of the key fields in the records in an indexed file must be the same length.

The maximum length of a key field is 99 positions unless an indexed file is being processed sequentially within key field limits using a CONSOLE device, in which case the maximum length is 29 positions. Key fields in packed decimal format can be up to eight positions in length.

**Column 31 (Record Address Type)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Blank | Relative record numbers are used in processing sequential, direct and indexed files.  
A sequential or direct file is being loaded.  
Records are read consecutively.  
Key fields in the record address file are in the same format as key fields in the indexed files.  
Relative record numbers from the address output file are used to process this file (for full-procedural files only). |
| A     | Key fields in zoned-decimal format are used in processing or loading indexed files and processing record address files. |
| I     | Relative record numbers from the address output file are used to process the file, or the file is an address output file consisting of relative record numbers. |
| P     | Key fields in packed-decimal format are used in processing or loading indexed files and processing record address files. |

Use column 31 to indicate how records in a DISK file are identified. Column 31 applies to DISK files specified as input, update, or chained output files. Together, columns 28 and 31 indicate:

- The method by which records are read from the file
- A direct file load.

The specifications for retrieving records are shown in Figure 193 on page 503.
<table>
<thead>
<tr>
<th>Processing Method</th>
<th>Entry in Column 28</th>
<th>Entry in Column 31&lt;sup&gt;1&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary, Secondary, or Demand Files</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consecutive</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>By address output&lt;sup&gt;2&lt;/sup&gt;(except demand files)</td>
<td>R</td>
<td>I</td>
</tr>
<tr>
<td>Sequential by key field</td>
<td>Blank</td>
<td>A or P</td>
</tr>
<tr>
<td>Sequential within key-field limits</td>
<td>L</td>
<td>A or P</td>
</tr>
<tr>
<td><strong>Chained Files</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random by relative record number</td>
<td>R</td>
<td>Blank</td>
</tr>
<tr>
<td>Random by key field</td>
<td>R</td>
<td>A or P</td>
</tr>
<tr>
<td>Direct file load (random load)</td>
<td>R</td>
<td>Blank &lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Full-Procedural Files</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consecutive</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>By address output&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>Sequential by key field</td>
<td>Blank</td>
<td>A or P</td>
</tr>
<tr>
<td>Sequential within key-field limits</td>
<td>L</td>
<td>A or P</td>
</tr>
<tr>
<td>Random by relative record number</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>Random by key field</td>
<td>Blank</td>
<td>A or P</td>
</tr>
</tbody>
</table>

*Figure 193. Specifications for Retrieving Records*

<sup>1</sup>When creating a file with key fields in packed-decimal format (P in column 31), you must specify the key field as packed in your output specifications.

<sup>2</sup>For address output files, column 31 must contain an I, indicating that binary relative record numbers are used in processing. For full-procedural files processed by address output files, column 31 must be blank.

<sup>3</sup>For files that do not allow deletions, a direct file load requires an O in column 15 and a C in column 16. For files that allow deletions, a direct file load requires an O in column 15 and a blank in column 16.
COLUMNS 35-38 (KEY FIELD STARTING LOCATION)

Column 32 (File Organization)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Sequential file, direct file, or indexed file processed consecutively or randomly, by relative record number. The program uses one input/output area for the file.</td>
</tr>
<tr>
<td>I</td>
<td>Indexed file processed sequentially or randomly, by key.</td>
</tr>
<tr>
<td>T</td>
<td>Address output file.</td>
</tr>
<tr>
<td>1-9</td>
<td>These entries (the former Additional Input/Output Area) have no relevance on an AS/400 system. They are syntax-checked and ignored.</td>
</tr>
</tbody>
</table>

Use column 32 to identify the organization of all DISK files except address output files, and to identify address output files. If columns 35-38 contain EXTK, then column 16 (file designation) must contain P, S, or F.5

Columns 33-34 (Overflow Indicator)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No overflow indicator is used.</td>
</tr>
<tr>
<td>OA-OG, OV</td>
<td>The overflow indicator you specify will be printed when overflow occurs.</td>
</tr>
</tbody>
</table>

Use columns 33 and 34 to specify an overflow indicator to condition the lines in each PRINTER file that will be printed when overflow occurs.

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

For more information on overflow processing, see Chapter 8, “Using a PRINTER File.”

Columns 35-38 (Key Field Starting Location)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Records are processed consecutively.</td>
</tr>
<tr>
<td>EXTK</td>
<td>The file uses noncontiguous fields as its key.</td>
</tr>
<tr>
<td>1-9999</td>
<td>Record position in which the key field begins if the key has only one field.</td>
</tr>
</tbody>
</table>

Use columns 35 through 38 to identify the beginning record position of the key field for an indexed file that only uses one field as its key, or to indicate that an index file uses noncontiguous fields as its key. Columns 35 through 38 apply only to indexed DISK files, and an entry must be made in these columns for an indexed DISK file.

Note: If columns 35-38 contain EXTK, then column 16 (file designation) must contain P, S, or F.

The key field of a record contains the information that identifies the record. This information is used in the index portion of the file. The key field must be in the
same location in all of the records in the file. The entry in these columns must end in column 38. Leading zeros can be omitted.

Key fields can contain hexadecimal FF characters.

**Column 39 (Extension Code)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No extension or line counter specifications are used.</td>
</tr>
<tr>
<td>E</td>
<td>Extension specifications further describe the file.</td>
</tr>
<tr>
<td>L</td>
<td>Line counter specifications further describe the file.</td>
</tr>
</tbody>
</table>

Use column 39 to indicate if the file is further described on the extension specifications or line counter specifications. Column 39 applies only to (1) prerun-time array and table files, (2) record address files, and (3) output files assigned to the printer. Describe PRINTER files on the line counter specifications, and describe array, table, and record address files on the extension specifications.

**Columns 40-46 (Device)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>Disk</td>
</tr>
<tr>
<td>WORKSTN</td>
<td>Display station or ICF file</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Printer</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Used for a device not supported directly by RPG</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Console data file or console record address file</td>
</tr>
<tr>
<td>KEYBORD</td>
<td>Keyboard</td>
</tr>
<tr>
<td>CRT</td>
<td>Display screen</td>
</tr>
<tr>
<td>BSCA</td>
<td>Binary synchronous communications adapter</td>
</tr>
</tbody>
</table>

Use columns 40 through 46 to identify the input/output device used for the file.

All entries must begin in column 40. The devices and the associated file types that can be used with each device are shown in Table 15 on page 506.
<table>
<thead>
<tr>
<th>Device</th>
<th>Form of Data</th>
<th>File Type</th>
<th>Column 15</th>
<th>Column 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>Disk</td>
<td>Primary input</td>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Secondary input</td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Record address file containing key-field limits</td>
<td>I</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Record address file containing relative record numbers (address output file)</td>
<td>I</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Full-procedural</td>
<td>I</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Chain input</td>
<td>I</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Demand</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Array or table (prerun-time only)</td>
<td>I</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Update (primary, secondary, full-procedural, chained, or demand)</td>
<td>U</td>
<td>P, S, F, C, or D</td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Output</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disk</td>
<td>Direct file that does not allow deletions</td>
<td>O</td>
<td>C</td>
</tr>
<tr>
<td>WORKSTN</td>
<td>Typed in by operator</td>
<td>Demand</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>Typed in by operator</td>
<td>Combined primary</td>
<td>C</td>
<td>P</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Printed lines</td>
<td>Output</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>Form of Data</td>
<td>File Type</td>
<td>Column 15</td>
<td>Column 16</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Primary input</td>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Secondary input</td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Demand</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Update (primary, secondary, or demand)</td>
<td>U.</td>
<td>P, S, or D</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Combined (primary, secondary, or demand)</td>
<td>C</td>
<td>P, S, or D</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Special device</td>
<td>Output</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Typed in by operator</td>
<td>Primary input</td>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Typed in by operator</td>
<td>Secondary input</td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Typed in by operator</td>
<td>Demand</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Typed in by operator</td>
<td>Record address files containing key-field limits</td>
<td>I</td>
<td>R</td>
</tr>
<tr>
<td>KEYBORD</td>
<td>Typed in by operator</td>
<td>Primary input</td>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>KEYBORD</td>
<td>Typed in by operator</td>
<td>Demand</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>CRT</td>
<td>Display lines</td>
<td>Output</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>BSCA</td>
<td>Data communications line</td>
<td>Primary input</td>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>BSCA</td>
<td>Data communications line</td>
<td>Secondary input</td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td>BSCA</td>
<td>Data communications line</td>
<td>Demand</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>BSCA</td>
<td>Data communications line</td>
<td>Output</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
Figure 194 shows the columns that can be used for the devices named.

### Device Types

The device type coded specifies the type of file that is used by the program. The file type dictates the type of device that will be used. The device type is coded in columns 40 through 46 of the file description specifications. The eight device types are: DISK, WORKSTN, PRINTER, SPECIAL, CONSOLE, KEYBOARD, CRT, and BSCA.

**DISK**

The DISK device is an input/output device that allows RPG programs to process data stored on a magnetic disk drive (disk). Data stored on a disk is called a DISK file. A DISK file can be used for input, for output, or for both.

Chapter 6, "Using a DISK File," explains how to code an RPG program that uses a DISK file.

**WORKSTN**

The WORKSTN device is an input/output device that allows RPG programs to communicate with one or more display stations. WORKSTN stands for work station, which is another name for display station. A display station consists of a display screen on which data is displayed and an attached keyboard from which data is entered. Use the System/36 display format specifications to define input and output for a WORKSTN file. The WORKSTN file allows programs to communicate with a remote system using ICF.

A program can use no more than one WORKSTN file, and a program that uses a WORKSTN file cannot use CONSOLE, KEYBOARD, or CRT files.
Chapter 7, “Using a WORKSTN File,” explains how to code a program that uses a WORKSTN file.

**PRINTER**
A PRINTER file can be used only for output. A program can use up to eight PRINTER files.

Chapter 8, “Using a PRINTER File,” explains how to code an RPG program that uses a PRINTER file.

**SPECIAL**
A SPECIAL file is handled by a device not directly supported by RPG. To use a SPECIAL file, you must use a subroutine to transfer data between the SPECIAL device and main storage.

Chapter 9, “Using a SPECIAL File,” explains how to code an RPG program that uses a SPECIAL file.

**CONSOLE**
The CONSOLE device is an input device that allows RPG programs to read data records directly from a display station. CONSOLE is another word for display station. A CONSOLE file can be used as a record address file or as an input data file. If used as a record address file, the file must be further defined on extensions specifications. A CONSOLE file can be used only as an input file. It cannot be used to display the records in a file.

Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File,” explains how to code an RPG program that uses a CONSOLE file.

**KEYBORD**
The KEYBORD device is an input device that allows RPG programs to receive data with the KEY and SET operation codes. Input specifications are not used for KEYBORD files. Instead, the input data is defined in the KEY and SET operations.

Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File,” explains how to code an RPG program that uses a KEYBORD file.

**CRT**
The CRT device is an output device that allows RPG programs to write data to a display station. CRT stands for cathode ray tube, which means the display screen of a display station. A CRT file can be used only as an output file to display information on the screen. A person using the display station cannot change this displayed information.

Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File,” explains how to code an RPG program that uses a CRT file.

**BSCA**
A BSCA file allows an RPG program to send and receive binary synchronous data on a data communications network.

Chapter 11, “Using a BSCA File,” explains how to code an RPG program that uses a BSCA file.
Columns 47-52

Columns 47 through 52 are not used. Leave them blank.

Column 53 (Continuation Lines–K)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Continuation record</td>
</tr>
</tbody>
</table>

Use column 53 to indicate that a continuation record provides additional information about the DISK file, WORKSTN file, or SPECIAL file being defined. Only one continuation record can be specified for each DISK file or each SPECIAL file; however, several continuation records can be specified for a WORKSTN file. When you specify a continuation record for a SPECIAL device, columns 54 through 59 (continuation-line option) must be coded. When you specify a continuation record for a DISK or WORKSTN device, columns 54 through 65 must be coded. Figure 195 shows an example of the coding necessary for a continuation line on the file description specifications for a SPECIAL file.

*...12...3...4...5...6...7...*
filenameIPEAFBlenRlenLK1AI0vKlocEDevice+Exit+++A...U+.*
FILE1 SPECIAL SUBRXX KARRAY1

Figure 195. Specifications for a SPECIAL Device

Columns 54-59

Name of Label Exit

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBRxx</td>
<td>Name of the user-written or IBM-written subroutine that processes the input/output operation for a SPECIAL device. (x = any alphabetic or numeric character.)</td>
</tr>
</tbody>
</table>

Use columns 54 through 59 to specify the subroutine that processes the input/output operations for a file assigned to a SPECIAL device. Columns 54 through 59 must contain an entry for each file assigned to a SPECIAL device. The subroutine name entered in columns 54 through 59 can be from four to six characters long. For a user-written subroutine the first four characters must be SUBR; the remaining characters can be any alphabetic or numeric character.

Continuation-Line Option for DISK File

The RECNO option is used to randomly add records to, or to load, a DISK file that allows deletion of records.
### Option Value

**RECNO** Name of a numeric field that is seven digits long with zero decimal positions. The name must be coded in the leftmost of columns 60 through 65. This field name must be specified if records are to be added randomly to a direct or sequential file that allows deletions. This field name is also required for a direct file load of a file that allows deletions.

You must place in the RECNO field the relative record number of the record added to the file. It must be the relative record number of a deleted record. RPG uses the relative record number in the RECNO field to determine where a record is loaded (direct file load) or added (ADD on output specifications).

**Note:** If the program successfully reads a record from a sequential or direct file by a CHAIN or READ operation, RPG places the relative record number of this record in the RECNO field.

### Continuation-Line Options for WORKSTN File

The following options can be specified for a WORKSTN file if more than one device is attached to a program or if you want to specify the WORKSTN file information data structure (INFDS) or the WORKSTN exception/error-processing subroutine (INFSR). The NUM keyword is required if the program attaches more than one device to a file at the same time. Enter the keyword in columns 54 through 59 and the value in columns 60 through 65 (columns 60 through 67 can be used if the FMTS option is specified).

**Note:** For WORKSTN files, a device can be either a display station or an ICF session.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NUM</strong></td>
<td>Number of devices that can be attached to this file at one time (Maximum is 251). The number specified must be coded in the rightmost of columns 60 through 65. If a number is not specified, 1 is assumed. If a number is specified, NUM must be greater than or equal to the number of requesters specified by the MRTMAX parameter when the program is compiled plus the number of acquired devices (those specified on the OCL WORKSTN statement or in the ACQ operation). The number specified on the MRTMAX parameter is reserved for requesters. The difference between the MRTMAX value and the NUM value is the maximum number of acquired devices that can be attached to the program at one time by using OCL statements or the ACQ operation. For example, if the MRTMAX value is 5 and the NUM value is 6, only one acquired device can be attached to the program, even if only one requester is currently signed on.</td>
</tr>
</tbody>
</table>

| SAVDS | Name of a data structure that is to be saved and restored for each device attached to this file. The name must be coded in the leftmost of columns 60 through 65. This data structure cannot be a display station local data area, and it cannot contain a compile-time array or a preruntime array. If SAVDS is not specified, no data area swapping is done. |

| IND | Number of indicators, beginning with 01, that are saved and restored by display the station. Prior to reading from a WORKSTN file, RPG saves indicators 01 through the number specified, for the device from which the last input record was read. After the read, indicators are restored from the save |
area belonging to the current device. No store is done before the first read. Therefore, after the first read, the indicators are restored from a blank save area, thereby setting them all off.

If IND is not specified, no indicator swapping is done. The entry must be coded in the rightmost of columns 60 through 65.

**Note:** For SAVDS and IND, only one copy of the data structure and indicators is available at a time. The indicators and data structure that are available are those associated with the device from which the last input was read.

The data structure and indicators that are available change each time an input operation (either a primary file input or a demand file read) is processed. On an input operation, the present copy of the data structure and indicators in the program is written to a save area for the device from which the previous input was read. The data structure and indicators for the device now being read from are then written from the save area associated with the device to the program SAVDS and IND areas. After the first input operation for each device, all the restored indicators will be off and all the fields in the SAVDS data structure will be blank. (For more information, see Chapter 7, "Using a WORKSTN File."

**SLN** Name of a 2-digit numeric field whose value determines the first line on the display screen where the display format is to begin if a variable starting line number (V in column 17 of the display format S specification) was specified in the format. The name must be coded in the leftmost of columns 60 through 65. If SLN is not specified, all formats having a variable starting line number begin on line 1.

**FMTS** *NONE.* indicates that there are only ICF formats present in this program.

Name of the display file that contains the display formats. The compiler uses the name specified here as the name of the display file. The name entered can be from one to eight characters in length and must be coded in the leftmost of columns 60 through 67. If a name is not entered, the compiler assumes that the name of the display file is the program name with FM added to the end of the name.

**ID** Name of a two-character, self-defining alphamer field that contains the identification of the device that supplied the record being processed in this file. The name must be coded in the leftmost of columns 60 through 65. The ID field is updated whenever a record is read from the WORKSTN file. Therefore, it always contains the identification of the device from which the last record was read (unless your program moves a different identification into the ID field). This field is considered self-defining because it need not be specified as an input or result field. For a multiple device file, you can direct an operation to a device other than the one currently being processed by changing the value in the ID field to the symbolic ID of another device in the file before processing the output operation.

The display station identifications are assigned at system configuration time. Display station identifications are in the form AX, where A is any alphabetic character (A-Z, #, @, or $) and X is any character. If the OCL WORKSTN statement exists for the display station, the identifica-
tion is identical to the value of the SYMID parameter as long as the program is not placed in the job queue or evoked.

ICF session identifications can be in two formats. They are either NN where N is numeric (0-9), or NA where N is numeric and A is alphabetic (A-Z, #, @, or $). If the format is NA, the OCL SESSION statement must be specified with a SYMID parameter whose value is also in an NA format.

**INFSR**
Name of the user-written calculation subroutine designated as the WORKSTN exception/error-processing subroutine. The name must be coded in the leftmost of columns 60 through 65. Control may be passed to this subroutine if an exception/error occurs during the following operations: ACQ, REL, NEXT, POST, input (READ or primary input), or output (EXCPT operation or normal cycle output). If INFSR is not specified, the program halts if an exception/error occurs. See “Handling Exceptions and Errors” on page 191 in Chapter 7, “Using a WORKSTN File,” for more information on INFSR.

**INFDS**
Name of the data structure that contains the identification of the type of exception/error condition and an indication of the WORKSTN operation that was running when the exception/error condition occurred. The name must be coded in the leftmost of columns 60 through 65. If INFDS is not specified, this information is not available to the RPG program.

**CFILE**
Name of an ICF user-defined communications file that associates a WORKSTN file with a communication format file. For more information about creating ICF files, see *ICF Programmer’s Guide*.

### Continuation-Line Option for SPECIAL Device

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array/table</td>
<td>Name of array or table used by the user-written subroutine name.</td>
</tr>
</tbody>
</table>

### Columns 60-65 (Storage Index)

Columns 60 through 65 are not used. Leave them blank if this is not a continuation line.

### Column 66 (File Addition)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>New records are added to the file.</td>
</tr>
<tr>
<td>U</td>
<td>U was used to specify that records are loaded for an indexed file in unordered sequence (random sequence). U is not valid on the AS/400 system because records of an indexed file are always loaded in unordered sequence. The entry will be syntax checked and ignored.</td>
</tr>
</tbody>
</table>

Use column 66 to indicate that the program is to add new records to the file. Records can be added at detail, total, or exception time during the program cycle.

Column 66 applies to direct, sequential, and indexed DISK files.
**Note:** Adding records to a file also requires a corresponding ADD entry in columns 16 through 18 of the output specifications.

For more information about adding records to a DISK file, see Chapter 6, “Using a DISK File.”

---

**Columns 67-70**

Columns 67 through 70 are not used. Leave them blank.

---

**Columns 71-72 (File Condition)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The file is not conditioned by an external indicator.</td>
</tr>
<tr>
<td>U1-U8</td>
<td>The file is conditioned by the specified external indicator.</td>
</tr>
</tbody>
</table>

Use columns 71 and 72 to indicate if the file is conditioned by an external indicator. Columns 71 and 72 apply to input (excluding table input files and KEYBORD files), update, and output files. A file conditioned by an external indicator is used only when the indicator is on. When the indicator is off, the file is treated as though the end of the file is reached; that is, no records can be read from or written to the file.

All operations involving a conditioned file should be conditioned to the same external indicator.

The external indicators are normally set prior to processing by the OCL SWITCH statement or by a previous RPG program. Their setting can be changed during processing, allowing the program to alter the status of these indicators. However, if an external indicator conditions a file, that indicator must be set on when the program is loaded in order to use the file in the program. For information about how to save and restore the external indicators for each display station attached to a WORKSTN file, see Chapter 7, “Using a WORKSTN File.”

If a file is conditioned by an external indicator, any operations that are not done when the file is not used should also be conditioned by the same indicator.

---

**Columns 73-74**

Columns 73 and 74 are not used. Leave them blank.

---

**Columns 75-80 (Program Identification)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
</tbody>
</table>

Any valid program name Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphanemic with no imbedded blanks. No special character can be used.
Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** For compatibility with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 20. Extension Specifications

Extension specifications describe: all record address files; compile-time or prerun-time tables; and compile-time, prerun-time, or run-time arrays used in a program.

Write these specifications on the RPG Extension and Line Counter Specifications sheet (see Figure 196). Record address files require entries in columns 11 through 26. Prerun-time arrays and tables require entries in columns 11 through 45. Compile-time arrays and tables require entries in columns 19 through 45. Run-time arrays require entries in columns 27 through 32 and in columns 36 through 45. If you want to specify an alternating array or table with the array or table described in columns 11 through 45 or 19 through 45, the alternating array or table must be described in columns 46 through 57 of the same line. A maximum of 200 arrays or tables can be used in a program.

Figure 196. RPG Extension and Line Counter Specifications

Figure 197 on page 518 shows possible extension specifications. See Chapter 14, “Using Arrays and Tables,” for a complete discussion of arrays and tables.
For tables and all arrays except run-time arrays, columns 19 through 26 are optional. For all arrays and tables, columns 46 through 57 are optional.

Run-time arrays are loaded by input and/or calculation specifications.

For record address files, columns 11 through 26 must have entries.

Figure 197. Possible Entries for Extension Specifications

Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.
Column 6 (Form Type)
An E must appear in column 6 to identify this line as an extension specifications statement.

Column 7 (Comments)
<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.

Columns 7-10
Columns 7 through 10 are not relevant to the AS/400 system. Leave them blank.

Columns 7-12 (/EJECT)
<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing.</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.

Columns 7-12 (/TITLE)
<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74.</td>
</tr>
</tbody>
</table>

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information.

The /TITLE statement causes an eject to the next page before the title is printed. The /TITLE statement is not printed on the compiler listing.
Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SPACEbn</td>
<td>Line spacing occurs at this point in the compiler listing</td>
</tr>
<tr>
<td></td>
<td>Valid entries for n are 1 to 12. The number must be left justified.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify n, 1 is assumed.</td>
</tr>
</tbody>
</table>

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.

Columns 11-18 (From Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Array or table is loaded at compilation time if there is an entry in columns 33 through 35.</td>
</tr>
<tr>
<td></td>
<td>Array is loaded at run time (by input and/or calculation specifications)</td>
</tr>
<tr>
<td></td>
<td>if there is no entry in columns 33 through 35.</td>
</tr>
<tr>
<td>Record address filename</td>
<td>Name of the record address file.</td>
</tr>
<tr>
<td>Array or table filename</td>
<td>Name of the array or table file loaded at prerun time.</td>
</tr>
</tbody>
</table>

Use columns 11 through 18 to name an array file, table file, or record address file. Filenames must begin in column 11. The record address filename must always be entered in these columns and also on the file description specifications. The filename of every prerun-time array or table used in the program must be entered in these columns and on the file description specifications. Leave columns 11 through 18 blank for compile-time arrays or tables and for arrays loaded by input and/or calculation specifications (run-time arrays).

When an array or table is loaded at compilation time, it is compiled along with the source program and included in the program object. Such an array or table does not need to be loaded separately every time the program is run. Only those arrays and tables that contain constant data should be compiled with the program.

When arrays or tables are compiled with the program, the array or table records must always follow the RPG source program. A record with **b (b = blank) in positions 1 through 3 must separate the RPG source program from the array or table records. Arrays or tables must be separated from each other by records with **b in positions 1 through 3. Because **b in positions 1 through 3 indicates the start of an
array or table, **b must not be specified in positions 1 through 3 of the array or table input records.

Short tables (tables that contain blank entries) can be compiled with the program. See “Columns 36-39 (Number of Entries per Table or Array)” on page 525 for a discussion of short tables.

Columns 19-26 (To Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of an input or update file</td>
<td>File processed by the record address file named in columns 11 through 18.</td>
</tr>
<tr>
<td>Name of an output file</td>
<td>Output file to which an array or table is to be written at end of job.</td>
</tr>
</tbody>
</table>

Use columns 19 through 26 to define the relationship between a file named in these columns and a file named in columns 11 through 18 or to name the file to which an array or table is to be written at end of job. Filenames must begin in column 19.

If a record address file is named in columns 11 through 18, the name of the input or update file that contains the data records to be processed must be entered in columns 19 through 26. Do not enter the record address filename in these columns.

If an array or table is written at end of job (that is, after last record processing), enter the filename of the output file in columns 19 through 26. This output file must be named previously in the file description specifications. An array or table can be written to only one output device. Leave columns 19 through 26 blank if the array or table is not written.

If an array or table is assigned to an output file, it is automatically written after all other records are written. The array or table is written in the same format in which it was entered.

Because there is no program control over the output format when an entry is made in columns 19 through 26, those cases where formatting is required should be provided for in the program through the output specifications or by the EXCPT operation that writes one item at a time. For more information, see Chapter 27, “Operation Codes.” Arrays or tables should be written only after all records are processed (last-record indicator LR is on).

Columns 27-32 (Array or Table Name)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array or table name</td>
<td>Name of array or table used in the program</td>
</tr>
</tbody>
</table>

Use columns 27 through 32 to name the array or table. No two arrays or tables can have the same name. The rules for forming array and table names are discussed in the following text.
COLUMNS 27-32 (ARRAY OR TABLE NAME)

Array Name

Each array used in a program must be given a separate name that does not begin with the letters TAB. The name can be from one to six characters long and must begin with an alphabetic character. This array name is used throughout the program. To refer to the entire array, use the array name alone. To refer to a single element of the array, use the array name plus an index. See “Using an Array Name and Index” on page 407 in Chapter 14, “Using Arrays and Tables” for more information on array names and on referencing array entries.

Table Name

Every table used in your program must have a name that begins with the letters TAB. The entire table name can be from three to six characters long.

After the letters TAB, one to three alphabetic or numeric characters can be used (no special characters are allowed). Blanks cannot appear between characters in the table name. Any name in columns 27 through 32 that does not begin with TAB is considered an array name.

The table name entered in columns 27 through 32 is used throughout the program. However, different results can be obtained depending upon how the table name is used. When the table name is used in factor 2 or the result field of the calculation specifications with a LOKUP operation, the name refers to the entire table. When the table name is used with any other operation code, the name refers to the table entry last selected from the table by a LOKUP operation (see Chapter 27, “Operation Codes,” and Chapter 14, “Using Arrays and Tables”).

Table files are processed in the order they are specified on the extension specifications. If you have more than one table file, the files should be loaded in the same order as they appear on the extension specifications.

If two tables are in alternating format in one file, the table whose entry appears first must be named in columns 27 through 32. The second table is named in columns 46 through 51 (see Figure 198).
Two tables (TABA and TABB) are described in alternating format. An item for TABA appears first. Therefore, TABA is named in columns 27 through 32 of the extension specifications sheet (see Part 2 of this figure); TABB is named in columns 46 through 51.

<table>
<thead>
<tr>
<th>Table A (Account Number)</th>
<th>Table B (Amount Due)</th>
</tr>
</thead>
<tbody>
<tr>
<td>03240</td>
<td>39.00</td>
</tr>
<tr>
<td>03648</td>
<td>156.72</td>
</tr>
<tr>
<td>15632</td>
<td>17.98</td>
</tr>
<tr>
<td>28887</td>
<td>2.97</td>
</tr>
<tr>
<td>29821</td>
<td>280.98</td>
</tr>
<tr>
<td>30001</td>
<td>579.95</td>
</tr>
</tbody>
</table>

The account number and the amount due for that account number are corresponding table items.

**Note:** The decimal points shown in Table B are only for illustration purposes. Decimal points are not a part of array or table input data.

The corresponding items from the tables are entered in the system in alternating format. Corresponding items from the two tables are considered as one entry.

*Figure 198 (Part 1 of 2). Related Tables*
Table entries for the two tables, A and B, are entered in alternating format. A1 and B1, the corresponding items in tables A and B, are considered one entry. Even though 14 table items are listed, there are only seven table entries.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromfileTofile++ArrnamEntParrLenPDSAltnamLenPDSComments+++++++++
E*
E* Table TABA whose items are loaded first is named in columns 27
E* through 32.
E*
E* The second entry 7 indicates the number of table entries in each
E* input record. Remember, corresponding items from the two tables are
E* considered as one entry.
E*
E* Table TABB whose items are loaded second is named in columns
E* 46 thorough 51.
E*
```

```
TABA    7    7    5    TABB    7    2
```

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

*Figure 198 (Part 2 of 2). Related Tables*

### Columns 33-35 (Number of Entries per Record)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-999</td>
<td>Number of array or table entries in each array or table input record</td>
</tr>
</tbody>
</table>

Use columns 33 through 35 to indicate the exact number of array or table entries in each array or table input record. The number must end in column 35. Every array or table input record except the last must contain the same number of entries as indicated in columns 33 through 35. The last record can contain fewer entries than indicated, but not more. Comments can be entered on table input records in the positions following the table entries.

If two arrays or tables are in alternating format in one file, each array or table input record must contain the corresponding entries from each array or table. The corresponding entries from the two arrays or tables are considered one entry and must be on the same record.
When columns 27 through 32 contain an array name, the following rules apply to the use of columns 11 through 18 and 33 through 35:

- For a prerun-time array, columns 11 through 18 must contain a filename and columns 33 through 35 must have an entry.
- For a compile-time array, columns 11 through 18 must be blank and columns 33 through 35 must have an entry.
- For a run-time array, columns 11 through 18 and columns 33 through 35 must be blank.

### Columns 36-39 (Number of Entries per Table or Array)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9999</td>
<td>Maximum number of array or table entries</td>
</tr>
</tbody>
</table>

Use columns 36 through 39 to indicate the maximum number of entries that can be contained in the array or table named in columns 27 through 32. This number applies to one array or table or to two arrays or tables in alternating format. The number entered must end in column 39.

Because the number of entries for two arrays or tables written in alternating format must be the same, the number in these columns also gives the number of entries in the second array or table specified in columns 46 through 51.

If the array or table is full, these columns give the exact number of entries in it. However, if the array or table is not full, these columns give the number of entries that can be put into it (see Figure 199 on page 526). An array or table that is not full is one that contains unused entries and is known as a **short array** or **table**.

A compile-time array or table should be full. However, if it is not full (a short array or table), the array or table is compiled with the program. In storage, the unused entries in a short array or table are filled with blanks or zeros (for alphameric or numeric arrays or tables, respectively).

A prerun-time array or table need not be full.
<table>
<thead>
<tr>
<th>TABPRT (Part Number)</th>
<th>TABAMT (Price)</th>
<th>TABPRT (Part Number)</th>
<th>TABAMT (Price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>127.62</td>
<td>001</td>
<td>127.62</td>
</tr>
<tr>
<td>002</td>
<td>198.32</td>
<td>002</td>
<td>198.32</td>
</tr>
<tr>
<td>003</td>
<td>0.27</td>
<td>003</td>
<td>0.27</td>
</tr>
<tr>
<td>004</td>
<td>0.01</td>
<td>004</td>
<td>0.01</td>
</tr>
<tr>
<td>005</td>
<td>1.98</td>
<td>005</td>
<td>1.98</td>
</tr>
<tr>
<td>006</td>
<td>3.79</td>
<td>006</td>
<td>3.79</td>
</tr>
<tr>
<td>007</td>
<td>5.67</td>
<td>007</td>
<td>5.67</td>
</tr>
<tr>
<td>008</td>
<td>2.33</td>
<td>008</td>
<td>2.33</td>
</tr>
<tr>
<td>009</td>
<td>2.33</td>
<td>009</td>
<td>2.33</td>
</tr>
<tr>
<td>010</td>
<td>2.33</td>
<td>010</td>
<td>2.33</td>
</tr>
<tr>
<td>011</td>
<td>2.33</td>
<td>011</td>
<td>2.33</td>
</tr>
<tr>
<td>012</td>
<td>2.33</td>
<td>012</td>
<td>2.33</td>
</tr>
<tr>
<td>013</td>
<td>2.33</td>
<td>013</td>
<td>2.33</td>
</tr>
<tr>
<td>014</td>
<td>2.33</td>
<td>014</td>
<td>2.33</td>
</tr>
<tr>
<td>015</td>
<td>2.33</td>
<td>015</td>
<td>2.33</td>
</tr>
<tr>
<td>016</td>
<td>2.33</td>
<td>016</td>
<td>2.33</td>
</tr>
<tr>
<td>017</td>
<td>2.33</td>
<td>017</td>
<td>2.33</td>
</tr>
<tr>
<td>018</td>
<td>2.33</td>
<td>018</td>
<td>2.33</td>
</tr>
<tr>
<td>019</td>
<td>2.33</td>
<td>019</td>
<td>2.33</td>
</tr>
<tr>
<td>020</td>
<td>2.33</td>
<td>020</td>
<td>2.33</td>
</tr>
<tr>
<td>021</td>
<td>2.33</td>
<td>021</td>
<td>2.33</td>
</tr>
<tr>
<td>022</td>
<td>2.33</td>
<td>022</td>
<td>2.33</td>
</tr>
<tr>
<td>023</td>
<td>2.33</td>
<td>023</td>
<td>2.33</td>
</tr>
<tr>
<td>024</td>
<td>2.33</td>
<td>024</td>
<td>2.33</td>
</tr>
<tr>
<td>025</td>
<td>2.33</td>
<td>025</td>
<td>2.33</td>
</tr>
<tr>
<td>026</td>
<td>2.33</td>
<td>026</td>
<td>2.33</td>
</tr>
<tr>
<td>027</td>
<td>2.33</td>
<td>027</td>
<td>2.33</td>
</tr>
<tr>
<td>028</td>
<td>2.33</td>
<td>028</td>
<td>2.33</td>
</tr>
<tr>
<td>029</td>
<td>2.33</td>
<td>029</td>
<td>2.33</td>
</tr>
<tr>
<td>030</td>
<td>2.33</td>
<td>030</td>
<td>2.33</td>
</tr>
<tr>
<td>031</td>
<td>2.33</td>
<td>031</td>
<td>2.33</td>
</tr>
<tr>
<td>032</td>
<td>2.33</td>
<td>032</td>
<td>2.33</td>
</tr>
<tr>
<td>033</td>
<td>2.33</td>
<td>033</td>
<td>2.33</td>
</tr>
<tr>
<td>034</td>
<td>2.33</td>
<td>034</td>
<td>2.33</td>
</tr>
<tr>
<td>035</td>
<td>2.33</td>
<td>035</td>
<td>2.33</td>
</tr>
<tr>
<td>036</td>
<td>2.33</td>
<td>036</td>
<td>2.33</td>
</tr>
<tr>
<td>037</td>
<td>2.33</td>
<td>037</td>
<td>2.33</td>
</tr>
<tr>
<td>038</td>
<td>2.33</td>
<td>038</td>
<td>2.33</td>
</tr>
<tr>
<td>039</td>
<td>2.33</td>
<td>039</td>
<td>2.33</td>
</tr>
<tr>
<td>040</td>
<td>2.33</td>
<td>040</td>
<td>2.33</td>
</tr>
<tr>
<td>041</td>
<td>2.33</td>
<td>041</td>
<td>2.33</td>
</tr>
<tr>
<td>042</td>
<td>2.33</td>
<td>042</td>
<td>2.33</td>
</tr>
</tbody>
</table>

Note: The decimal points shown in these tables are only for illustration purposes. Decimal points are not part of table input data.

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

E....FromfileTofile++ArrnamEntParrLenPDSAltnamLenPDSComments++++++++

E* The third entry (20) indicates that TABPRT and TABAMT can both have E* a maximum of 20 entries.

E*

E

**TABPRT 12 20 3 0 TABAMT 5 2**

*Figure 199. Table Entries (Number per Table)*

**Columns 40-42 (Length of Entry)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-15</td>
<td>Length of a numeric entry</td>
</tr>
<tr>
<td>1-256</td>
<td>Length of an alphameric entry</td>
</tr>
</tbody>
</table>

Use columns 40 through 42 to specify the length of each entry in the array or table named in columns 27 through 32. The number entered must end in column 42. For numeric arrays or tables in packed-decimal format, enter the zoned-decimal
length in columns 40 through 42. For numeric arrays or tables in binary format, enter the number of digits required in storage for the binary field. For a two-position binary field, the entry in columns 40 through 42 is 4; for a four-position binary field, the entry is 9.

All array or table entries must have the same number of characters. It is almost impossible, however, for every item to be the same length. Therefore, add leading zeros for numeric entries and add blanks after alphameric entries to make them the same length as shown below.

<table>
<thead>
<tr>
<th>Table 16. Table Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Months</strong></td>
</tr>
<tr>
<td>JANUARY</td>
</tr>
<tr>
<td>FEBRUARY</td>
</tr>
<tr>
<td>MARCH</td>
</tr>
<tr>
<td>APRIL</td>
</tr>
<tr>
<td>MAY</td>
</tr>
<tr>
<td>JUNE</td>
</tr>
<tr>
<td>JULY</td>
</tr>
<tr>
<td>AUGUST</td>
</tr>
<tr>
<td>SEPTEMBER</td>
</tr>
<tr>
<td>OCTOBER</td>
</tr>
<tr>
<td>NOVEMBER</td>
</tr>
<tr>
<td>DECEMBER</td>
</tr>
</tbody>
</table>

All entries must have the same length. Those items that are not as long as the longest item must be padded with blanks (b).

If two arrays or tables are entered in alternating format, the specification in columns 40 through 42 applies to the array or table whose entry appears first in the record (see Figure 200).

See Chapter 14, “Using Arrays and Tables,” for more information.
Two tables are entered in alternating format, TABCOD and TABAMT. Each item in TABCOD is three characters long; each item in TABAMT is six characters long. Because TABCOD is entered in the system first, its length, 3, is specified in columns 40 through 42. The length of items in TABAMT is in columns 52 through 54.

**Note:** The decimal points shown in these tables are only for illustration purposes. Decimal points are not a part of table input data.

<table>
<thead>
<tr>
<th>TABCOD (Code)</th>
<th>TABAMT (Amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>021</td>
<td>217.43</td>
</tr>
<tr>
<td>022</td>
<td>93.06</td>
</tr>
<tr>
<td>023</td>
<td>8.14</td>
</tr>
<tr>
<td>040</td>
<td>2166.58</td>
</tr>
<tr>
<td>041</td>
<td>39.23</td>
</tr>
<tr>
<td>060</td>
<td>1741.78</td>
</tr>
<tr>
<td>117</td>
<td>83.33</td>
</tr>
<tr>
<td>118</td>
<td>5.12</td>
</tr>
<tr>
<td>143</td>
<td>72.03</td>
</tr>
<tr>
<td>352</td>
<td>253.96</td>
</tr>
</tbody>
</table>

Figure 200. Length of Corresponding Table Items
### Column 43 (Packed or Binary Field)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Data for array or table is in zoned-decimal format or is alphameric.</td>
</tr>
<tr>
<td>P</td>
<td>Data for array or table is in packed-decimal format on disk.</td>
</tr>
<tr>
<td>B</td>
<td>Data for array or table is in binary format on disk.</td>
</tr>
</tbody>
</table>

Use column 43 to indicate that a numeric field in a prerun-time array or table file is in packed or binary format. Leave column 43 blank if the field is in zoned-decimal format. See “Column 43 (Packed-Decimal or Binary Field)” on page 571 under “Field Description Entries” on page 571 in Chapter 23, “Input Specifications,” for more information on packed or binary format.

### Column 44 (Decimal Positions)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Alphameric array or table</td>
</tr>
<tr>
<td>0-9</td>
<td>Number of positions to the right of the decimal in numeric array or table items</td>
</tr>
</tbody>
</table>

Use column 44 to indicate the number of decimal positions in a numeric array or table entry. Column 44 must always have an entry for a numeric array or table. If the entries in an array or table have no decimal positions, enter a 0.

If two arrays or tables are entered in alternating format, the specification in this column applies to the array or table containing the entry that appears first on the record.

### Column 45 (Sequence)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No particular order</td>
</tr>
<tr>
<td>A</td>
<td>Ascending order</td>
</tr>
<tr>
<td>D</td>
<td>Descending order</td>
</tr>
</tbody>
</table>

Use column 45 to describe the sequence (either ascending or descending) of the data in an array or table.

When an entry is made in column 45, the array or table is checked for the specified sequence. If a compile-time array or table is out of sequence, a terminal error occurs, and the compiler halts. If a prerun-time array or table is out of sequence, an error occurs, and the program halts immediately. The program can be restarted from the point where it halted if you do not want to correct the out-of-sequence condition; however, if you do correct the out-of-sequence condition, program processing must be restarted from the beginning.

**Ascending order** means that the array or table entries start with the lowest data entry (according to the collating sequence) and proceed to the highest.

**Descending order** means that the array or table entries start with the highest data entry and proceed to the lowest.
If two arrays or tables are entered in alternating format, the entry in column 45 applies to the array or table containing the entry that appears first on the record. When the LOKUP operation is used to search an array or table for an entry to determine if the entry is high or low compared with the search word, the array or table must be in either ascending or descending order. See “LOKUP (Lookup)” on page 774 for more information.

A run-time array (built by input and/or calculation specifications) is not sequence-checked. However, an A or D entry must be specified if a high or low LOKUP operation is processed.

### Columns 46-57

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array or table name and description</td>
<td>Name and description of the alternating array or table</td>
</tr>
</tbody>
</table>

Use columns 46 through 57 only to describe a second array or table that is entered in an alternating format with the array or table specified in columns 27 through 45. All fields in this section have the same significance and require the same entries as the fields with corresponding titles in columns 27 through 45. See the previous discussion on those columns for information about correct specifications. Leave these columns blank for a single array or table.

### Columns 58-74 (Comments)

Columns 58 through 74 can be used for comments to document the purpose of each specification line.

### Columns 75-80 (Program Identification)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphabetic with no imbedded blanks. No special characters can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** To be compatible with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 21. Line Counter Specifications

Line counter specifications indicate the line at which overflow occurs and the length of the form used in the printer. Both of these entries can be specified on the RPG Extension and Line Counter Specifications sheet (see Figure 201).

You use line counter specifications for each PRINTER file in your program. If no line counter specifications exist, the form length used is the form length specified on the OCL PRINTER statement. (See the LINES parameter of the OCL PRINTER statement in the System Reference for the System/36 Environment for a description of the defaults for the form length.) In this instance, assume the overflow line is six lines less than the specified form length.

Figure 201. RPG Extension and Line Counter Specifications
### Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

### Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

### Column 6 (Form Type)

An L must appear in column 6 to identify this line as a line counter specifications statement.

### Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to help document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.

### Column 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing.</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.
### Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74. A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information. The /TITLE statement causes an eject to the next page before the title is printed. The /TITLE statement is not printed on the compiler listing.</td>
</tr>
</tbody>
</table>

### Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| /SPACEb
| Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed. |

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines to be spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.

### Columns 7-14 (Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid filename</td>
<td>Filename of the printer output file as previously defined on the file description specifications sheet. The filename must begin in column 7.</td>
</tr>
</tbody>
</table>

Use columns 7 through 14 to identify the output file printed on the printer.

### Columns 15-17 (Line Number–Number of Lines per Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-112</td>
<td>Number of printing lines available is 2 to 112.</td>
</tr>
</tbody>
</table>

Use columns 15 through 17 to specify the exact number of lines available on the page you want to use. The entry must end in column 17. Leading zeros can be omitted.
**Columns 18-19 (Form Length)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>Form length</td>
</tr>
</tbody>
</table>

Use columns 18 and 19 to indicate that the entry in columns 15 through 17 is the form length. Columns 18 and 19 must contain the entry FL.

**Columns 20-22 (Line Number–Overflow Line)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-112</td>
<td>The line number specified is the overflow line.</td>
</tr>
</tbody>
</table>

Use columns 20 through 22 to specify the line number that is the overflow line. The entry must end in column 22. Leading zeros can be omitted. The entry must be less than or equal to the form length specified in columns 15 through 17. When the line that is specified as the overflow line is printed, the overflow indicator turns on. When the overflow indicator is on and fetch overflow is not specified, the following occurs before forms advance to the next page:

1. Detail lines are printed (if this part of the program cycle has not already been completed).
2. Total lines are printed (if conditions are met).
3. Total lines conditioned by the overflow indicator are printed.

Because all these lines are printed on the page after the overflow line, specify the overflow line high enough on the page to allow all these lines to print. See “Handling Overflow” on page 265 and “Overflow Indicators” on page 355 for more information.

**Note:** If the number of lines per page entry equals the overflow line entry, no overflow occurs.

**Columns 23-24 (Overflow Line)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL</td>
<td>Overflow line</td>
</tr>
</tbody>
</table>

Use columns 23 and 24 to indicate that the entry in columns 20 through 22 is the overflow line. Columns 23 and 24 must contain OL.

**Columns 25-74**

Columns 25 through 74 are not used. Leave them blank.
**Columns 75-80 (Program Identification)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphanemic with no imbedded blanks. No special character can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** To be compatible with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 22. Telecommunications Specifications

Telecommunications specifications describe the information necessary to establish and maintain the batch binary synchronous communications (BSCA) link. Each BSCA file defined on the file description specifications must have a corresponding specification on the RPG Telecommunications Specifications sheet. (See Figure 202 on page 538.)

RPG data communications programming enables you to send and receive binary synchronous data by means of a data communications network. RPG data communications support processes all the functions necessary to establish the line connections, exchange identification sequences, send and receive data, and run the correct end or disconnect procedures.

---

**Note**

This chapter describes the System/36 RPG II Telecommunications support. Many of the parameters documented below are syntax-checked for compatibility with System/36, but are ignored on the AS/400 system. These parameters will be identified with *** prior to the title. In these cases the parameters documented are syntax-checked for compatibility with System/36. The value specified in the AS/400 system communications configuration is used. Refer to Chapter 11, “Using a BSCA File” on page 313.
RPG batch BSC permits the AS/400 system to function as any of the following station types:

- Receive only (receive input data from a remote station)
- Send only (send data to a remote station)
- Send and receive, but no conversational reply. Three modes of operation are possible:
  - Sending a file, and then receiving another file
  - Receiving a file, and then sending another file
  - Sending records of one file interspersed with receiving records of another file.

BSC is a flexible form of line control that provides a set of rules for communications between devices. For a description of the basic characteristics and operational concepts of BSC, a description of the RPG interface to BSC, and a complete description of RPG data communications programming, see Chapter 11, “Using a BSCA File.”

**Note:** Telecommunications specifications are used only for RPG data communications programming (batch BSC).
### Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

### Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to run an RPG program successfully. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

### Column 6 (Form Type)

A T must appear in column 6 to identify this line as a telecommunications specification.

### Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. You can use any character in a comment line.

### Columns 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing.</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.
### Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74.</td>
</tr>
</tbody>
</table>

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information.

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

### Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SPACEbn</td>
<td>Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed.</td>
</tr>
</tbody>
</table>

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.

### Columns 7-14 (File name)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid file name</td>
<td>File name previously defined on the file description specifications for the BSC device.</td>
</tr>
</tbody>
</table>

### **Column 15 (Configuration)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P or blank</td>
<td>This is a point-to-point nonswitched line.</td>
</tr>
<tr>
<td>M</td>
<td>This is a multipoint line where the control station selects the tributary station through polling or addressing. (The System/36 and also the AS/400 system cannot be the control station.)</td>
</tr>
<tr>
<td>S</td>
<td>This is a point-to-point switched line.</td>
</tr>
</tbody>
</table>

If this column contains an M, column 17 must contain a T.
### Column 16 (Type of Station)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>This station sends (transmits) information from the file named in columns 7 through 14. The file must be designated as an output file on the file description specifications and must be defined on the output specifications.</td>
</tr>
<tr>
<td>R</td>
<td>This station receives information in the file named in columns 7 through 14. The file must be designated as an input file on the file description specifications and must be defined on the input specifications.</td>
</tr>
</tbody>
</table>

**Note:** This entry is independent of the entry in column 20.

### ***Column 17 (Type of Control)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Polling is not used.</td>
</tr>
<tr>
<td>T</td>
<td>This is a tributary station on a multipoint network. Column 17 must contain a T if column 15 contains an M.</td>
</tr>
</tbody>
</table>

The AS/400 system cannot be the control station.

### ***Column 18 (Type of Code)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A or U</td>
<td>ASCII (formerly referred to as USASCII) transmission control characters are used. An A or U entry does the necessary file translation on System/36. Refer to “ASCII-EBCDIC Character Translation” on page 319 for file translation on the AS/400 system.</td>
</tr>
<tr>
<td>E or blank</td>
<td>EBCDIC transmission control characters are used. ASCII and EBCDIC characters are listed in the ICF Programmer’s Guide.</td>
</tr>
</tbody>
</table>

If your BSC program halts because of an invalid ASCII character in your data, check your data and the ASCII translation table.

### ***Column 19 (Transparency)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>EBCDIC transparency is used. The data being transferred can contain transmission control characters and/or packed numeric or alphameric characters. Column 18 must be E or blank.</td>
</tr>
<tr>
<td>N or blank</td>
<td>EBCDIC transparency is not used. Zoned-decimal numeric or alphameric data is sent and received. The data being transferred cannot contain transmission characters.</td>
</tr>
</tbody>
</table>
***Column 20 (Switched)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>This is not a switched line.</td>
</tr>
<tr>
<td>M</td>
<td>The operator using this program makes the connection by dialing the number (manual dial).</td>
</tr>
<tr>
<td>A</td>
<td>This program uses autoanswer.</td>
</tr>
<tr>
<td>B</td>
<td>This program uses manual answer.</td>
</tr>
</tbody>
</table>

**Note:**
1. This entry is independent of the entry in column 16.
2. If you are using an autocall line, the switch type specified has no effect. If no phone list is specified in the OCL SESSION statement, the switch type specified here is established.

Columns 21-31

Columns 21 through 31 are not used. Leave them blank.

***Column 32 (Location of Identification—This Station)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No identification is used for this station.</td>
</tr>
<tr>
<td>S</td>
<td>This station’s identification is at the position specified by the symbolic name in columns 33 through 39. This entry applies only to switched lines.</td>
</tr>
<tr>
<td>E</td>
<td>The entry in columns 33 through 39 is this station’s identification. This entry applies only to switched lines.</td>
</tr>
</tbody>
</table>

***Columns 33-39 (Identification—This Station)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic characters</td>
<td>When column 40 contains an E, this entry is the actual identification sequence of the remote station (minimum of two characters). When column 40 contains an S, this entry is the symbolic name of the location of the remote station’s identification.</td>
</tr>
</tbody>
</table>

If columns 33 through 39 contain a symbolic name, it must not be an array name. If the BSCA file is a primary or secondary file, the symbolic name must refer to the first entry of a table (the table might have only one entry) to make sure that the station identification is in storage before the communications line is opened.

The station identification referred to by the symbolic name can be from 2 to 15 characters long, but it must not contain a transmission control character. The station identification is translated if the BSCA files are translated.
### **Column 40 (Location of Identification—Remote Station)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No identification is used for the remote station.</td>
</tr>
<tr>
<td>S</td>
<td>The remote station's identification is at the position specified by the symbolic name in columns 41 through 47. This entry applies only to switched lines.</td>
</tr>
<tr>
<td>E</td>
<td>The entry in columns 41 through 47 is the remote station's identification. This entry applies only to switched lines.</td>
</tr>
</tbody>
</table>

### **Columns 41-47 (Identification—Remote Station)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphameric characters</td>
<td>When column 40 contains an E, this entry is the actual identification sequence of the remote station (minimum of two characters). When column 40 contains an S, this entry is the symbolic name of the location of the remote station's identification.</td>
</tr>
</tbody>
</table>

If columns 41 through 47 contain a symbolic name, it must not be an array name. If the BSCA file is a primary or secondary file, this symbolic name must refer to the first entry in a table (the table might have only one entry) to make sure that the station identification is in storage before the communications line is opened.

The station identification referred to by the symbolic name can be from 2 to 15 characters long, but must not contain a transmission control sequence character. The station identification is translated if the BSCA files are translated. The identification received from the remote station is compared with this entry. The session continues only if the identification matches.

### Columns 48-51

Columns 48 through 51 are not used. Leave them blank.

### **Column 52 (ITB)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Intermediate block checking is not used.</td>
</tr>
<tr>
<td>I</td>
<td>Intermediate block checking is used.</td>
</tr>
</tbody>
</table>

Intermediate block checking (ITB) can be used only if the records are blocked. ITB and EBCDIC transparency cannot both be specified for the same BSCA output file.
Columns 53-54 (Permanent-Error Indicator)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No permanent-error indicator is specified. If a permanent error occurs, a system halt occurs and the program cannot be restarted.</td>
</tr>
<tr>
<td>01-99, L1-L9, LR, H1-H9</td>
<td>A permanent-error indicator can be specified for every BSCA file. The indicator does not have to be unique for each file. You should specify a permanent-error indicator when the system is running and no one is attending to the system.</td>
</tr>
</tbody>
</table>

Use columns 53 and 54 to specify a permanent-error indicator for every BSCA file. When a permanent error occurs, the specified error indicator and the identification indicator of the record causing the error turn on. No hardware diagnostics are done. The permanent-error indicator can then be used to condition the appropriate programming response, such as printing a message or processing a controlled cancel.

Do not attempt to send information while the permanent-error indicator is on. This restriction includes attempts to send more than one record during detail, total, or exception output. Further attempts to send information can be prevented if each record sent is conditioned with the not-permanent-error indicator. Indicate Nn, where n is the permanent-error indicator, in columns 9 through 11 of the calculation specifications or columns 23 through 31 of the output specifications.

To retry an operation after a permanent error occurs, turn off the permanent-error indicator. The RPG program can then access the BSCA file on which the error occurred. If an error occurs on the retried operation, the permanent-error indicator is turned on again. If no errors occur on the retried operation, processing continues.

Consider the following points when retrying an operation:

- The permanent-error indicator is the only indication to the RPG program that an error occurred. A BSC information message describing the type of error is displayed. If a halt (H1 through H9) is not issued as part of the permanent-error routine, the BSC information message may not be preserved on the display screen. You can see more detailed messages by running the HISTORY procedure.

- Any data in the BSC buffers at the time of an error is lost. The record in your buffers is not the same as the record in the BSC buffers. Therefore, retrying the last operation will still result in lost data.

- Switched lines are not turned off when an error occurs unless a disconnect sequence is received or the hardware detects disconnect.

- Any data sent while the permanent-error indicator is on is invalid. Unless your program is designed to recognize all data, the error condition can cause an unidentified record halt.

- The RPG program imposes a limit on the number of times an error can occur before the program is stopped.

- The occurrence of a permanent error indicates the end of processing of a file, but not the end of file.
Note: Avoid using H1 through H9 as permanent-error indicators if you are going to condition operations on the permanent-error indicator being off. Because H1 through H9 are reset at the end of the detail logic cycle, they can be set off before the program cycle in which the error occurred is completed. If H1 through H9 are used as permanent-error indicators, the H1 through H9 display can preempt the system halt display. If the H1 through H9 display appears before the system display, the person using the display station should take the 0 option to prompt the system halt display.

***Columns 55-57 (Wait Time)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The system convention for time-out, 180 seconds, is used.</td>
</tr>
<tr>
<td>Numeric</td>
<td>The length of time in seconds, 1 to 999, that BSC waits with no messages being sent or received before a permanent error occurs.</td>
</tr>
</tbody>
</table>

A permanent error is recognized by the system whenever the wait time on an idle line elapses. Therefore, when determining the wait time, consider the time the person using the display station might require to respond to halts and other processing interruptions, as well as the time the program might require for special operations such as table searches and computing square roots.

The wait-time limit specified applies only to delays caused by the System/36 program and does not apply to the remote device. In addition, the time limit applies only during the transmission or reception of a file, not between file transmissions.

Columns 58-59 (Record-Available Indicator)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-99, L1-L9, LR, H1-H9</td>
<td>A record-available indicator should be specified if a reverse interrupt (RVI) is received. This indicator turns on whenever a reverse interrupt (RVI) is received.</td>
</tr>
</tbody>
</table>

Columns 60 (Last File)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>This BSCA file may not be the last input file processed.</td>
</tr>
<tr>
<td>L</td>
<td>This BSCA file is processed only after all other input files are processed.  All secondary files should have L in column 60.</td>
</tr>
</tbody>
</table>

The entry in column 60 does not affect demand files.
***Columns 61-62 (Polling Characters)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>This station is not part of a multipoint network.</td>
</tr>
<tr>
<td>Alphameric characters</td>
<td>The polling identification of this station is required if this station is</td>
</tr>
<tr>
<td></td>
<td>part of a multipoint network and the BSCA file is a sending (output) file.</td>
</tr>
<tr>
<td></td>
<td>Polling and addressing characters must be used in pairs.</td>
</tr>
</tbody>
</table>

Columns 63-64 (Addressing Characters)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>This station is not part of multipoint network.</td>
</tr>
<tr>
<td>Alphameric characters</td>
<td>The addressing identification of this station is required if this station is part of a multipoint network and the BSCA file is a receiving (input) file. Polling and addressing characters must be used in pairs. Enter polling and addressing characters in EBCDIC; the compiler converts the characters to the form required by the code specified in column 18. (If ASCII was specified, enter uppercase addressing characters; they are converted to lowercase ASCII characters.)</td>
</tr>
</tbody>
</table>

Columns 65-74

Columns 65 through 74 are not used. Leave them blank.

Columns 75-80 (Program Identification)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphabetic with no imbedded blanks. No special character can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source listing.

**Note:** The specifications sheets show only 80 positions for each statement in order to be compatible with other RPG systems. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 23. Input Specifications

Input specifications describe the data files, records, and fields of the records used in the program. All input files are described on the input specifications except files assigned to the device KEYBORD, record address files, address output files, and table files. KEYBORD files are described on the calculation specifications when the KEY operation code is used. Record address files, address output files, and table files are described on the extension specifications.

Input specifications are also used to describe data structures.

The input specifications are divided into two categories:

- File and record-type identification entries (columns 7 through 42) describe the input record and its relationship to other records in the file.
- Field description entries (columns 43 through 74) describe the fields in the records. These specifications must start on the line below the file and record-type identification specifications.

Write these specifications on the RPG Input Specifications sheet (see Figure 203).
File and Record-Type Identification Entries

Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

Column 6 (Form Type)

An I must appear in column 6 to identify this line as an input specification.

Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to help document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to the RPG II program; they only document your program.

Columns 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing.</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.
Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74.</td>
</tr>
</tbody>
</table>

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information.

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

Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACEbn</td>
<td>Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed.</td>
</tr>
</tbody>
</table>

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the three blank lines that occur between specification types.
## Columns 7-14 (Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid filename or data structure name</td>
<td>Same filename that appears on the file description specifications for the input file or the name of a data structure.</td>
</tr>
</tbody>
</table>

If a data structure is specified (DS in columns 19 and 20), columns 7 through 14 can contain:

- Blanks
- A name up to six characters long
- A name previously referenced in columns 53 through 58 of the input specifications.

Data structure entries must be the last statements on the input specifications.

## Columns 14-16

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND or OR</td>
<td>AND/OR indicates a relationship between record-identifying indicators or record types. The entry must begin in column 14.</td>
</tr>
</tbody>
</table>

See “Columns 21-41 (Record Identification Codes)” on page 567 and “Columns 53-58 (Field Name)” on page 576 for more information on the AND/OR relationship.

## Columns 15-16 (Sequence)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any two alphabetic characters</td>
<td>Program does not check for special sequence. Alphabetic characters must be used for full-procedural files, chained files, demand files (except CONSOLE demand files), WORKSTN files, and lookahead records.</td>
</tr>
</tbody>
</table>

01-99 | Program checks for special sequence. |

Use an entry (01 through 99) in columns 15 and 16 to assign a special sequence number to different record types in a file. The first sequence number must be 01. Gaps in sequence numbers are allowed, but the numbers must be in ascending order.
If the types of records do not need to be in any special order, use two alphabetic characters (see Figure 204). Within one file, all record types having alphabetic entries in columns 15 and 16 must be described before those types with numeric entries.

File RECORDA has two types of records (part number and item number) that can appear in any order. Because they are not checked for sequencing, they are assigned two alphabetic characters in columns 15 and 16 (AA and BC, respectively) instead of numbers.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.................................................................*
IRECORDA AA 01 1 CP
I........................................PFromTo++DField+L1M1FrPoNeEq...*
I   BC 02 1 CI
```

*Figure 204. Unsequenced Record Types in a File*
Assigning Sequence Numbers

Enter a numeric character in columns 15 and 16 if one record type (identified by a record identification code) must be read before another record type in a sequenced group. To specify sequence checking, each record type must have a record identification code, and the record types must be numbered in the order they should appear. The program checks this order as the records are read (see Figure 205 on page 553). If a record type is out of sequence, the program stops and error message RPG903I File [file name] contains a record not in sequence is displayed. You can continue the program by selecting option 0 and pressing an entry function key. The program bypasses the record that caused the halt and reads the next record from the same file.

Sequence numbers make sure that all records of the lowest record type come before the records of the next highest record type. The sequence numbers do not make sure that records within a record type are in any certain order. Sequence numbers are unrelated to control levels and do not provide for checking data in fields of a record for a special sequence (see Figure 206 on page 554). Use columns 61 and 62 to indicate that data in fields of a record be checked for a special sequence.

Records in an OR or AND line cannot have a sequence entry in these columns. The entry in columns 15 and 16 on the previous line also applies to the OR or AND line.
This file contains four different kinds of records. The records are arranged in groups according to a customer name control field. The name record is first in each group and is assigned sequence number 01. Street record is next and is assigned 02. City/state record is 03. (Remember, gaps are allowed). Item number record is 07. More than one item number record can be present (N in column 17).

*.. 1 +.. 2 +.. 3 +.. 4 +.. 5 +.. 6 +.. 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.................................*
ICUST  011  1 CN
I........................................PFromTo++DField+L1M1FrPoNeEq...*
I                  5 25 NAME
I      0210  1 CS
I      2 26 STREET
I      031  1 CC
I      5 21 CTYST
I      07NO  1 CI
I              10 16 ITEM
I              18 23 QTY

Figure 205. Sequence Checking of Record Type
Each group is in proper sequence according to the assigned sequence numbers (01, 02, 03, and 07). Notice, however, that the city/state record for customer 3 is in the group for customer 2 and vice versa. The sequence entry that you specify in columns 15 and 16 does not catch this mistake because the sequence entry does not cause the data on the record to be checked. See Figure 23-3 for the coding of this example.

*Figure 206. Correct Record Sequence (Incorrect Data within Groups)*
## Column 17 (Number)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program does not check record types for a special sequence (columns 15 and 16 have alphabetic entries).</td>
</tr>
<tr>
<td>1</td>
<td>Only one record of this type can be present in the sequenced group.</td>
</tr>
<tr>
<td>N</td>
<td>One or more records of this type can be present in the sequenced group.</td>
</tr>
</tbody>
</table>

Use column 17 only if columns 15 and 16 contain a numeric entry specifying sequence checking (see Figure 207 on page 556).

OR lines (columns 14 and 15 contain OR) and AND lines (columns 14 through 16 contain AND) should not have an entry in this column. The entry in column 17 on the previous line also applies to the OR or AND line. See “Columns 53-58 (Field Name)” on page 576 for more information on OR lines.
Customer 2
Record types 02 and 07 are optional as indicated by 0 in column 18.

Customer 1
Only one record of types 01, 02, and 03 can be present as indicated by 1 in column 17; however, any number of record type 07 can be present as indicated by N in column 17.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME=SQNORiPos1NCCPos2NCCPos3NCC.................................*
ICUST  011  1 CN
I........................................PFromTo++DField+L1M1FrPoNeEq...*
I
I   0210  1 CS
I
I   031  1 CC
I
I   07NO  1 CI
I
I   10  16 ITEM
I
I   18  23 QTY
```

*Figure 207. Sequenced Record File*
### Column 18 (Option)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Record type must be present if sequence checking is specified.</td>
</tr>
<tr>
<td>O</td>
<td>Record type is optional (that is, it may or may not be present) if sequence checking is specified.</td>
</tr>
<tr>
<td>U</td>
<td>The program uses the data structure defined on this specification line as a display station local data area.</td>
</tr>
</tbody>
</table>

Use column 18 only if columns 15 and 16 contain a numeric entry specifying sequence checking, or if the data structure defined on the following specification line is used as a display station local data area.

If sequence checking is specified and all record types are optional, no sequence error is found.

OR and AND lines should not have an entry in this column. The entry in column 18 on the previous line also applies to the OR or AND line. See "Columns 53-58 (Field Name)" on page 576 for more information on OR lines.

### Columns 19-20 (Record-Identifying Indicator, **, DS)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-10</td>
<td>Record-identifying indicator for CONSOLE files. Record-identifying indicators 01 through 10 for CONSOLE files correspond to function keys 1 through 10.</td>
</tr>
<tr>
<td>01-99</td>
<td>Record-identifying indicator.</td>
</tr>
<tr>
<td>L1-L9</td>
<td>Control-level indicator used for a record-identifying indicator when a record type rather than a control field signals the start of a new control group.</td>
</tr>
<tr>
<td>LR</td>
<td>Last-record indicator.</td>
</tr>
<tr>
<td>H1-H9</td>
<td>Halt indicator used for a record-identifying indicator when checking for a record type that causes an error condition.</td>
</tr>
<tr>
<td>**</td>
<td>Look-ahead field Look-ahead can be used only with input or update files; however, these files cannot be full-procedural, chained, or demand files. Look-ahead fields are not valid with CONSOLE files or WORKSTN files.</td>
</tr>
<tr>
<td>DS</td>
<td>Data structure. A data structure is considered to be alphabetic data and can be from 1 to 9999 characters in length. Data structure entries must be the last entries on the input specifications. For more information about data structures, see Chapter 15, “Using Data Structures.”</td>
</tr>
</tbody>
</table>
Look-Ahead

A look-ahead field allows you to:

- Determine when the last record of a control group is being processed
- Extend the RPG matching-record capability.

Because an RPG program processes one record at a time, normally only the information from the record being processed is available for use. However, look-ahead makes information available from records that follow the one currently being processed. This information can then be used to determine the operation that should be done next.

Any or all of the fields in a file can be described as look-ahead fields. The description applies to all records in the file regardless of their type. Look-ahead fields can be described before or after the field descriptions for any of the records in the file. The line that signals that look-ahead fields are described must contain an alphabetic entry in columns 15 and 16 and must contain ** in columns 19 and 20. All the other columns must be blank. Remember that specifications with an alphabetic sequence in columns 15 and 16 must precede specifications with a numeric sequence in columns 15 and 16.

Look-ahead fields are described on the lines immediately following the line that contains ** in column 19 and 20 (see Figure 208 on page 559). Make the following entries for each look-ahead field description line:

- Columns 44-51: Identify the record positions in which the field is located.
- Column 52: If the field is numeric, enter the number of digits to the right of the decimal point in column 52. If there are no decimal positions, enter a 0. If the field is alphameric, leave this column blank.
- Columns 53-58: Enter the name of the look-ahead field. If the field is also one of the normal fields in the record, use a different name for the look-ahead field.

The program reads records from two disk files. The primary file is named PRIMARY; the secondary file, SECONDARY. If a record from the primary file matches one from the secondary file, the information in positions 1 through 10 of the secondary file record is placed in positions 31 through 40 of the primary file record. When there is no match, a 6 is placed in position 1 of the primary file record. The 6 indicates an unmatched record in the primary file.

Because the primary file record is processed first when it matches a secondary file record, the information from the secondary file can be made available only by a look-ahead field.
Figure 208. Look-Ahead Fields
For input files, look-ahead fields always apply to the next record in the file, provided the file is not an update file. Therefore, if the information is used both before and after the record is selected for processing, describe the field twice, once as a look-ahead field and once as a normal field. See Figure 209 on page 561 for an example of how records are selected for processing from two input files when look-ahead fields are used.

For update files, the look-ahead fields apply to the next record in the file only if the record currently selected for processing was read from another file.

Therefore, when the program is reading from only one file and that file is an update file, look-ahead fields always apply to the current record and contain the same information as a normal field. See Figure 210 on page 564 for an example of how records are selected for processing from an update file and an input file when look-ahead fields are used.

As the last record from a file is processed, every look-ahead field for the file is automatically filled with 9s. For example, a look-ahead field that is three characters long will contain 999. The 9s remain in the field until the job ends. The blank-after option (B in column 39 of the output specifications) cannot be used with look-ahead fields.
Chapter 23. Input Specifications

Figure 209 (Part 1 of 3). Available Records: Two Input Files
1. Select second record from primary file. for processing.

2. Read third record from primary file.

Figure 209 (Part 2 of 3). Available Records: Two Input Files
Figure 209 (Part 3 of 3). Available Records: Two Input Files
Figure 210 (Part 1 of 3). Available Records: One Input File, One Update File
1. Read second record from update file.

1. Select second record from update file for processing.

1. Select second record from update file for processing.

Figure 210 (Part 2 of 3). Available Records: One Input File, One Update File
1. Read third record from update file.
2. Select first record from secondary file for processing.
3. Read second record from secondary file.

Records Available for Look-Ahead

<table>
<thead>
<tr>
<th>Processed Records</th>
<th>Records Being Processed</th>
<th>Records Available for Look Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>U1 and S1</td>
<td></td>
</tr>
<tr>
<td>U2</td>
<td>U2 and S1</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>U3 and S2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 210 (Part 3 of 3). Available Records: One Input File, One Update File
Columns 21-41 (Record Identification Codes)

Use columns 21 through 41 to describe the information that identifies a record type. If all records are processed alike regardless of their type, or if there is only one record type, leave columns 21 through 41 blank.

**Note:** Only columns 21 through 34 are valid for CONSOLE files (see Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File,” for more information).

When one file contains more than one record type, each record type is identified by a code consisting of a character or a combination of characters in certain positions in the record. If different operations are done for each record type, this code must be described in columns 21 through 41 so that the program can determine the type of record selected for processing. Only one type of record is selected for processing during a program cycle, and the record-identifying indicator for that record turns on at the time of selection.

Seven columns are used for the description of one character in the record identification code. Each specification line contains three sets of seven columns: columns 21 through 27, 28 through 34, and 35 through 41. Each set consists of four fields: Position, Not, C/Z/D, and Character. Coding is the same for all three sets.

**Note:** Any record that is read by the system and is not described by a record identification code in columns 21 through 41 causes the program to halt. The person using the display station can continue, however, by selecting the appropriate option. The record that causes the halt is not processed, and the next record in that file is read.

**Position (Columns 21-24, 28-31, and 35-38)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No record identification code is needed.</td>
</tr>
<tr>
<td>1-9999</td>
<td>Record position of one character in the record identification code.</td>
</tr>
</tbody>
</table>

Use these columns to give the location in the record of every character in the identification code. These entries must end in columns 24, 31, and 38 respectively.

**Not (N) (Columns 25, 32, and 39)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Character is present in the specified record position.</td>
</tr>
<tr>
<td>N</td>
<td>Character should not be present in the specified record position (not valid for CONSOLE files; see Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File”).</td>
</tr>
</tbody>
</table>

Use these columns to indicate that a certain character should not be present in a specified position.
C/Z/D (Columns 26, 33, and 40)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Entire character. C must be used for CONSOLE files.</td>
</tr>
<tr>
<td>Z</td>
<td>Zone portion of character.</td>
</tr>
<tr>
<td>D</td>
<td>Digit portion of character.</td>
</tr>
</tbody>
</table>

Use these columns to indicate the portion of a character that is used as part of the record identification code. Only the zone portion, only the digit portion, or both portions (the whole character) can be used (see Figure 211). When establishing record identification codes, remember that many characters have either the same zone or the same digit portion. For a list of characters that have identical zone or digit portions, see Figure 212 on page 569.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenamesurveyPos1NCCPos2NCCPos3NCC.........................................
I*
I* The character 5 must be present in position 1, the zone portion
I* of the character T in position 94, the character 9 in position
I* 95, and the digit portion of the character E in position 96.
I* However, the digit portion of the character 9 must not be present
I* in column 93. Only the digit portions of 9 and E are checked,
I* and only the zone portion of character T is checked.
I*
I* The AND must be used to describe the last 2 characters
I* of a 5-character code (95 C9 96 DE).
I*
IPAYROLL BC 12 1 C5 93ND9 94 ZT
I AND 95 C9 96 DE
I*
I* Record type 15 can be identified by two different codes: a 5 in
I* position 1 and a 6 in position 2, or a 6 in position 1.
I*
IHRSWRK DF 15 1 C5 2 C6
I OR 1 C6
```

Figure 211. Record Identification Codes
**Character Grouping by Zone (Z)**

<table>
<thead>
<tr>
<th>Zone 4</th>
<th>Zone 8</th>
<th>Zone D</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>a</td>
<td>- (minus)</td>
</tr>
<tr>
<td>è</td>
<td>b</td>
<td>)</td>
</tr>
<tr>
<td>.</td>
<td>c</td>
<td>J</td>
</tr>
<tr>
<td>&lt;</td>
<td>d</td>
<td>K</td>
</tr>
<tr>
<td>(</td>
<td>e</td>
<td>L</td>
</tr>
<tr>
<td>+</td>
<td>f</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g</td>
</tr>
<tr>
<td>i</td>
<td>h</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zone 5</th>
<th>Zone 9</th>
<th>Zone E</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>j</td>
<td>\</td>
</tr>
<tr>
<td>!</td>
<td>k</td>
<td>S</td>
</tr>
<tr>
<td>$</td>
<td>l</td>
<td>T</td>
</tr>
<tr>
<td>}</td>
<td>m</td>
<td>U</td>
</tr>
<tr>
<td>;</td>
<td>n</td>
<td>V</td>
</tr>
<tr>
<td>)</td>
<td>o</td>
<td>W</td>
</tr>
<tr>
<td>?</td>
<td>p</td>
<td>X</td>
</tr>
<tr>
<td>&amp;</td>
<td>q</td>
<td>Y</td>
</tr>
<tr>
<td>)</td>
<td>r</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zone 6</th>
<th>Zone A</th>
<th>Zone F</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (minus)</td>
<td>s</td>
<td>blank</td>
</tr>
<tr>
<td>[</td>
<td>t</td>
<td>0</td>
</tr>
<tr>
<td>, (comma)</td>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>)</td>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>(underscore)</td>
<td>w</td>
<td>3</td>
</tr>
<tr>
<td>&gt;</td>
<td>x</td>
<td>4</td>
</tr>
<tr>
<td>?</td>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zone 7</th>
<th>Zone C</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>À</td>
</tr>
<tr>
<td>:</td>
<td>@</td>
</tr>
<tr>
<td>&amp;</td>
<td>Á</td>
</tr>
<tr>
<td>#</td>
<td>À</td>
</tr>
<tr>
<td>(apostrophe)</td>
<td>'</td>
</tr>
<tr>
<td>=</td>
<td>À</td>
</tr>
</tbody>
</table>

**Character Grouping by Digit (D)**

<table>
<thead>
<tr>
<th>Digit 0</th>
<th>Digit 6</th>
<th>Digit C</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>f</td>
<td>– – –</td>
</tr>
<tr>
<td>&amp;</td>
<td>o</td>
<td>* * *</td>
</tr>
<tr>
<td>- (minus)</td>
<td>w</td>
<td>% % %</td>
</tr>
<tr>
<td>)</td>
<td>F</td>
<td>0 0 0</td>
</tr>
<tr>
<td>\</td>
<td>D</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digit 1</th>
<th>Digit 7</th>
<th>Digit D</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>g</td>
<td>( (</td>
</tr>
<tr>
<td>a</td>
<td>p</td>
<td>) (apostrophe)</td>
</tr>
<tr>
<td>J</td>
<td>x</td>
<td>\</td>
</tr>
<tr>
<td>G</td>
<td>P</td>
<td>%</td>
</tr>
<tr>
<td>A</td>
<td>J</td>
<td>w</td>
</tr>
<tr>
<td>J</td>
<td>X</td>
<td>%</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digit 2</th>
<th>Digit 8</th>
<th>Digit E</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>h</td>
<td>– – –</td>
</tr>
<tr>
<td>k</td>
<td>q</td>
<td>: : :</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
<td>&gt; &gt; &gt;</td>
</tr>
<tr>
<td>B</td>
<td>H</td>
<td>– – –</td>
</tr>
<tr>
<td>K</td>
<td>Q</td>
<td>Y</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digit 3</th>
<th>Digit 9</th>
<th>Digit F</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>i</td>
<td>– – –</td>
</tr>
<tr>
<td>l</td>
<td>r</td>
<td>– – –</td>
</tr>
<tr>
<td>t</td>
<td>z</td>
<td>– – –</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>– – –</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>– – –</td>
</tr>
<tr>
<td>T</td>
<td>Z</td>
<td>– – –</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digit 4</th>
<th>Digit A</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>¢</td>
</tr>
<tr>
<td>m</td>
<td>„</td>
</tr>
<tr>
<td>u</td>
<td>:</td>
</tr>
<tr>
<td>D</td>
<td>M</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digit 5</th>
<th>Digit B</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>’</td>
</tr>
<tr>
<td>n</td>
<td>’</td>
</tr>
<tr>
<td>v</td>
<td>’</td>
</tr>
<tr>
<td>E</td>
<td>’</td>
</tr>
<tr>
<td>N</td>
<td>’</td>
</tr>
<tr>
<td>V</td>
<td>’</td>
</tr>
<tr>
<td>s</td>
<td>’</td>
</tr>
</tbody>
</table>

---

**Figure 212. Characters Interpreted as Having the Same Zone or Digit**

**Character (Columns 27, 34, and 41)**

In these columns, enter the alphabetic character, special character, or numeric character that is used in the record as the identification code or part of the code.

**Character Grouping by Zone or Digit**

When characters are used for record identification purposes on a digit- or zone-only basis, all characters having the same zone or digit are selected by the system as meeting record identification requirements. When a character is read into the system, it is converted into an eight-bit code. The program tests this eight-bit code to determine if the character meets the requirements of the record identifying character in the input specifications.
Figure 212 lists the characters that have identical zones or digits. For example, if column 26 contains D, which specifies digit-only, and column 27 contains A, all records having a slash (/), A, J, or 1 in the specified column are selected as having the correct record identification code. If column 26 contains Z and column 27 contains A, all records containing & or A through I are selected as having the correct code.

The following three special cases are exceptions:

- The hexadecimal representation of an & (ampersand) is 50. However, when the ampersand is coded in the character entry, it is treated as though its hexadecimal representation were C0, that is, as if it had the same zone as the characters A through I. An ampersand in the input data satisfies two zone checks, for either a hexadecimal 5 zone or a hexadecimal C zone.

- The hexadecimal representation of a - (minus sign) is 60. However, when the minus sign is coded in the character entry, it is treated as though its hexadecimal representation were D0, that is, as if it had the same zone as the characters J through R. A minus sign in the input data satisfies two zone checks, for either a hexadecimal 6 zone or a hexadecimal D zone.

- The hexadecimal representation of a blank is 40. However, when the blank is coded in the character entry, it is treated as though its hexadecimal representation were F0, that is, as if it had the same zone as the numeric characters 0 through 9. A blank in the input data satisfies two zone checks, for either a hexadecimal 4 zone or a hexadecimal F zone.

**AND Relationship**

A maximum of three identifying characters can be described in one specification line. If the identification code consists of more than three characters, an AND line must be used to describe the additional characters. Code the word AND in columns 14 through 16 to indicate an AND line (see Figure 211 on page 568).

Any number of AND lines can be used to describe the record identification code for a record sequence. The record must contain all the characters indicated as its record identification code before the record-identifying indicator turns on. AND lines are not allowed on CONSOLE files used for interactive data entry.

**OR Relationship**

If a particular record type can be identified by two different codes, OR lines must be used to indicate that either of the codes can be present to identify the record. There is no limit to the number of OR lines that can appear for each record sequence. Code the word OR in columns 14 and 15 to indicate an OR line (see Figure 211 on page 568).

**Note:** If AND lines and OR lines are combined any number of AND lines and OR lines can be used.

**Column 42**

Column 42 is not used. Leave it blank.
Field Description Entries

The field description entries (columns 43 through 74) must begin one line below the file and record identification entries (columns 7 through 42) for each file.

Column 43 (Packed-Decimal or Binary Field)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Field is in zoned-decimal format or is alphanemic.</td>
</tr>
<tr>
<td>P</td>
<td>Field named in columns 53 through 58 is in packed-decimal format.</td>
</tr>
<tr>
<td>B</td>
<td>Field named in columns 53 through 58 is in binary format.</td>
</tr>
</tbody>
</table>

Use column 43 to indicate the format of a numeric field in a record.

Packed or binary input to arrays should have a P or B in this column. The From and To columns should then define the positions the array occupies in the record in the packed or binary format. The zoned-decimal length of each array element is defined on the extension specifications.

Zoned-Decimal Format (Blank)

Zoned-decimal format means that each byte can contain one character. That character can be a decimal number or an alphabetic or special character. In the zoned-decimal format, each byte is divided into a four-bit zone portion and a four-bit digit portion. The zoned-decimal format looks like this:

```
0 ──── 7 0 ──── 7 0 ──── 7 0 ──── 7 0 ──── 7
```

The zone portion of the low-order byte indicates if the decimal number is positive or negative. A positive value is indicated by a hexadecimal F, and a negative value is indicated by a hexadecimal D. 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. The decimal number 8191 looks like this in zoned-decimal format:

```
8 ──── 1 ──── 9 ──── 1
```

The AS/400 system RPG II compiler uses all data internally as zoned, and verifies the zone portion if the data is numeric. You can use the FIXDECDTA parameter of the CRTS36RPG command to specify how the program will handle bad decimal data.
If you specify FIXDECDTA(*NO), bad decimal data will cause error message RPG9078 to display or print, and the program to end.

If you leave the parameter blank, or specify FIXDECDTA(*YES), the system will fix the bad decimal data when it moves it from the input buffer to the fields on input, or when it moves it to the result field on an arithmetic or MOVE operation. The system changes the zone portion of the byte to ‘F’ if it is not ‘F’, and the digit portion of the byte to ‘0’ if it is not in the range of ‘0’ through ‘9’. The zone portion of the rightmost byte is changed to ‘F’ if it is not ‘F’ or ‘D’.

**Note:** The FIXDECDTA parameter of the CRTS36AUTO command is always set to ‘*YES’.

For more efficient use of disk storage, you may want to enter your numeric data (decimal numbers) in packed-decimal or binary format.

## Packed-Decimal Format (P)

Packed-decimal format means that each byte (except for the low-order byte) can contain two digits. Because many of the fields in a DISK file contain decimal numbers, you can conserve storage by using the packed-decimal format.

In the packed decimal format, each byte, except the low-order byte, is divided into two four-bit digit portions. The rightmost portion of the low-order byte contains the sign (plus or minus) for that field. The packed-decimal format looks like this:

```
   0   7  0   7
```

```
+----+----+----+----+
| Digit | Digit | Digit | Sign |
+----+----+----+----+
```

The sign portion of the low-order byte indicates if the numeric value represented in the digit portions is positive or negative. A positive value is indicated by a hexadecimal F, and a negative value is indicated by a hexadecimal D. In the packed-decimal format, the sign is included for each decimal number; however, the zone portion is not given for each digit in the number. Compare how the decimal number 8191 is represented in packed-decimal format with its zoned-decimal representation shown before (see Figure 213 on page 575).

Because processing requires the zoned-decimal format once it is inside the computer, you must indicate when input fields are in another format. Entering a P in column 43 indicates that the input field is in the packed-decimal format and that the system must convert this field to the required zoned-decimal format.

When a packed-decimal field is converted to a zoned-decimal field, the zoned-decimal field always contains an odd number of bytes. If a zoned-decimal field with an even number of bytes is converted to a packed-decimal field and then converted back to a zoned-decimal field, the resulting zoned-decimal field also contains an odd number of bytes.

Packed-decimal fields can be up to eight bytes long. The following chart shows the packed-decimal equivalents for zoned-decimal fields up to 15 bytes long, which is the maximum length.
Table 17. Packed-Decimal Equivalents for Zoned-Decimal Fields

<table>
<thead>
<tr>
<th>Zoned-Decimal Length in Bytes</th>
<th>Packed-Decimal Length in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Binary Format (B)

Binary format means that two bytes can contain a four-digit number, and that four bytes can contain a nine-digit number. The binary format allows you to save even more disk storage space than you can save using the packed-decimal format. In the binary format, each field on disk must be either two or four bytes long.

Each two-byte binary field consists of a one-bit sign followed by a 15-bit numeric value. In binary format, a decimal number as high as 9999 requires only two bytes of disk storage. For each two-byte binary field, the RPG compiler automatically sets aside four bytes of storage to accommodate the field when it is unpacked. A two-byte field in binary format looks like this:

```
<table>
<thead>
<tr>
<th>Sign</th>
<th>Number</th>
</tr>
</thead>
</table>
```

Each four-byte binary field consists of a 1-bit sign followed by a 31-bit numeric value. In binary format, a decimal number as high as 999 999 999 requires only four bytes of disk storage. For each four-byte binary field stored on disk, the RPG compiler automatically sets aside nine bytes of storage to accommodate the field when it is converted. A four-byte field in binary format looks like this:

```
<table>
<thead>
<tr>
<th>Sign</th>
<th>Number</th>
</tr>
</thead>
</table>
```

---

Chapter 23. Input Specifications  573
In each case, the sign portion of the high-order byte indicates if the numeric value is positive (sign bit off) or negative (sign bit on). Positive numbers are represented in true binary notation with a 0 bit in the sign position. Negative numbers are represented in twos-complement notation with a 1 bit in the sign position. The bits between the sign position and the leftmost significant bit of the integer are always the same as the sign bit. When the number is positive, all bits to the left of the most significant bit, including the sign bit, are 0s. When the number is negative, all bits to the left of the most significant bit, including the sign bit, are 1s. Notice that, in the binary format, the zone position of the decimal number is not given.

Compare how the decimal number 8191 is represented in binary format with packed-decimal and with zoned-decimal representation (see Figure 213 on page 575).

Because processing requires the zoned-decimal format once it is inside the computer, you must indicate when input fields are in another format. Entering a B in column 43 indicates that the input field is in the binary format and that the system must convert this field to the required zoned-decimal format.
To obtain the numeric value of a positive binary number, add the values of the bits that are on (1); the sign bit is not included. To obtain the numeric value of negative binary number, add the values of the bits that are off (0) plus one; the sign bit is not included (twos-complement notation).

If 8191 is read into storage as a zoned-decimal field, it occupies four bytes. However, if it is converted to packed-decimal format, it occupies three bytes; then when it is converted back to zoned-decimal format, it occupies five bytes.

Figure 213. Binary, Packed-Decimal, and Zoned-Decimal Representation of 8191

Columns 44-51 (Field Location)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9999</td>
<td>Beginning of a field (From) or end of a field (To). See CONSOLE file considerations. For a WORKSTN file, the From and To positions refer to the location of the fields in the input record and not to their location in the display format. Use columns 44 through 51 to describe the location on the record of the field named in columns 53 through 58. Enter the number of the record position in which the field begins in columns 44 through 47. Enter the number of the record position in which the field ends in columns 48 through 51. The entries must end in columns 47 and 51. Leading zeros can be omitted. Define a single-position field by entering the same number in both the From (columns 44 through 47) and To (columns 48 through 51) positions. If a field of more than one position is defined, the number entered in columns 44 through 47 must be smaller than the number entered in columns 48 through 51.</td>
</tr>
</tbody>
</table>
The maximum field length for a zoned-decimal numeric field is 15 positions (eight if
the field is packed-decimal and four if it is binary). The maximum field length for an
alphameric field is 256 characters, and the maximum length for a data structure is
9999 characters.

Column 52 (Decimal Positions)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Alphameric field</td>
</tr>
<tr>
<td>0-9</td>
<td>Number of decimal positions in numeric field</td>
</tr>
</tbody>
</table>

Use column 52 to indicate the number of decimal positions in any numeric field
named in columns 53 through 58. Column 52 must contain an entry when the field
named in columns 53 through 58 is numeric. To define a field as numeric with no
decimal position, enter a 0. If a field is used in arithmetic operations or is edited, it
must be numeric. If the number of decimal positions specified for a field exceeds
the length of that field, the number of decimal positions is assumed equal to the
length of the field.

Columns 53-58 (Field Name)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>alphameric</td>
</tr>
<tr>
<td>PAGE,</td>
<td>Field name, array name, or array element</td>
</tr>
<tr>
<td>PAGE1-</td>
<td>characters</td>
</tr>
<tr>
<td>PAGE7</td>
<td>Special words</td>
</tr>
</tbody>
</table>

Use columns 53 through 58 to name a field, array, or array element found on your
input records. When referencing an array, additional entries may be needed in
these columns (see “Using an Array Name and Index” on page 407).

Use this name throughout the program whenever you refer to this field. Indicate
the names of the fields for all types of records using a separate line for each field.
However, name only the fields that you use. For example, if you use only the first
10 positions of a record that is 96 positions long, define positions 1 through 10 on
the input specifications.

For CONSOLE files, whole array names must be entered in one of the following
ways:

- Define the whole array as a subfield within a field.
- Define each element of the array with an index and place this entry in columns
  53 through 58 of the input specifications. The index must be an integer value.
Field Names

A field name can be from one to six characters long and must begin in column 53. The first character must be an alphabetic character. The remaining characters can be any combination of alphabetic and numeric characters (special characters are not allowed). Blanks cannot appear between characters in the name.

All fields in one type of record should have different names. If two or more fields on the same record type have the same name, only the field described last is used. However, fields from different record types can have the same name if the fields are the same length and contain the same type of data. This applies even if the fields are in different locations in each record type.

Numeric fields can have a maximum length of 15 digits. alphanumerical fields can have a maximum length of 256 characters (66 for CONSOLE files). A data structure can have a maximum length of 9999 characters. Subfields can have a maximum length of 256 characters for alphanumerical subfields and 15 digits for numeric subfields.

If a data structure subfield is specified in columns 53 through 58, only field-record-relation indicators (columns 63 and 64) can be specified. Entries for control-level indicators (columns 59 and 60), match field values (columns 61 and 62), and field indicators (columns 65 through 70) are not allowed. A data structure name cannot be specified as a subfield in a data structure.

Fields that are used in arithmetic operations (see Chapter 27, “Operation Codes”) or fields that are edited or zero-suppressed (see “Column 38 (Edit Codes)” on page 617 and “Columns 45-70 (Constant or Edit Word)” on page 619 in Chapter 25, “Output Specifications”) must be defined as numeric. Therefore, column 52 must have a decimal position entry (0 through 9).

Field Names in OR Relationship

If two or more record types contain identical fields, you must describe each field. To eliminate duplicate coding of identical fields from different record types, use the OR relationship (see Figure 212 on page 569). Any number of OR lines can be used for each record sequence group.

An OR relationship means that the fields named can be found in either of the record types. You can use OR lines when:

- Two or more record types have the same fields in the same positions (see Figure 214 on page 578).
- Two or more record types have some fields that are identical and some fields that differ in location, length, or type of data. See “Columns 63-64 (Field Record Relation)” on page 581 for sample coding of such record types.

Code OR in columns 14 and 15 to indicate an OR line. If there are several AND or OR lines, field description lines start after the last record identification line.

Special Words (PAGE, PAGE1-PAGE7)

If a printed report has several pages that are numbered, use the special word PAGE to indicate that page numbering is done. When you use a PAGE entry on the output specifications, page numbering automatically starts with 1 (see “Columns 32-37 (Field Name)” on page 612 in Chapter 25, “Output Specifications”).
To start at a page number other than 1, enter that page number in a field of an input record and name that field PAGE in columns 53 through 58. The number entered in the PAGE field should be one less than the starting page number. If numbering starts with 24, enter a 23 in the PAGE field. The PAGE field can be 1 to 15 digits long, but must have zero decimal positions (see Figure 215 on page 579). If a PAGE field is used but it is not defined, the PAGE field is assumed to be four digits long with zero decimal positions. Any entry in the PAGE field should be coded in the rightmost columns, such as 0023.

Page numbering can be restarted during a program run when a number is specified in a PAGE field of any input record. The PAGE field can be defined as a numeric field, 1 to 15 digits in length, with zero decimal positions, and used in calculations like any other field.

The eight possible PAGE entries (PAGE, PAGE1, PAGE2, PAGE3, PAGE4, PAGE5, PAGE6, and PAGE7) are provided for numbering different page types in the output file or for numbering the pages for different PRINTER files.

*... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC................................*  
ISALES AA 14 1 C5
I....................................PFromTo++DField+L1MLFrPoNeEq...*  
I 5 8 DEPT  
I 9 14 EMPNO  
I 46 500ITEM  
I 66 700COST  
I BB 15 1 C6  
I 5 8 DEPT  
I 9 14 EMPNO  
I 46 500ITEM  
I 66 700COST  
I*  
I* To eliminate duplicate coding, use the OR relationship.  
I*  
ISALES AA 14 1 C5  
I OR 15 1 C6  
I 5 8 DEPT  
I 9 14 EMPNO  
I 46 500ITEM  
I 66 700COST  

Figure 214. Record Types with Identical Fields
**Columns 59-60 (Control Level)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-L9</td>
<td>Any control-level indicator. Control-level indicators cannot be used with full-procedural, chained, demand, or WORKSTN files or with a data structure.</td>
</tr>
</tbody>
</table>

Use columns 59 and 60 to assign control-level indicators to input fields. Use control-level indicators to specify when calculation or output operations are processed. For more information, see Chapter 13, “Using Indicators.”

**Columns 61-62 (Matching Fields)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1-M9</td>
<td>Any matching level</td>
</tr>
</tbody>
</table>

Use columns 61 and 62 to specify match fields and sequence checking. Match fields and sequence checking cannot be specified for chained files, full-procedural files, demand files, WORKSTN files, or a data structure.

An entry in columns 61 and 62 indicates:

- Match fields and sequence checking when you have two or more input or update files with match fields
- Sequence checking only when you have just one input or update file

The match levels are ranked in order of importance, with M1 being the least significant.

**Match Fields**

In processing more than one input file, specify match fields to compare records from two or more input or update files to determine the record that is to be selected for processing. You can use one field, many fields, or an entire record to match records. Whenever the contents of the match field from the primary file record are the same as the contents of the match field from a secondary file record, the matching-record (MR) indicator turns on. The matching-record indicator can then be used to condition those operations that are done only when records match (see “Columns 9-17 (Indicators)” on page 588 in Chapter 24, “Calculation Specifications”; “Columns 23-31 (Output Indicators)” on page 611 in Chapter 25, “Output Specifications”; and Figure 130 on page 380 in Chapter 13, “Using Indicators”).
As many as nine match fields can be indicated when you use the values M1 through M9.

M1 through M9 only identify the fields by which the records are matched; they are not indicators, but they cause the matching-record indicator to turn on.

For a complete description of how to assign match fields and how records are selected for processing, see Chapter 12, “Primary/Secondary/Multifile Processing.”

Sequence Checking

To check the data in the fields of a record in one input or update file for a special sequence, assign a value of M1 through M9 to the field to be checked. As many as nine fields can be checked. The sequence (ascending or descending) of the record file must be specified in column 18 of the file description specifications (see Chapter 19, “File Description Specifications”). See Figure 216 for an example of sequence checking.

To check the sequence of record types in a file, see “Columns 15-16 (Sequence)” on page 550.

An input file called MASTER is sequence-checked through three fields. Data from two records is shown below:

<table>
<thead>
<tr>
<th>First Record</th>
<th>Second Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT 008</td>
<td>DEPT 003</td>
</tr>
<tr>
<td>REGION 051</td>
<td>REGION 025</td>
</tr>
<tr>
<td>DIVSON 003</td>
<td>DIVSON 005</td>
</tr>
</tbody>
</table>

In sequence checking, all fields are treated as one continuous field. Therefore, the match fields look like:

<table>
<thead>
<tr>
<th>M3</th>
<th>M2</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>003</td>
<td>051</td>
<td>008</td>
</tr>
<tr>
<td>005</td>
<td>025</td>
<td>003</td>
</tr>
</tbody>
</table>

The match field from record 1 is compared with the match field from record 2. If the file is specified in ascending sequence, the records are in order because 005025003 is higher than 003051008. However, if the file is specified as having a descending sequence, record 2 is out of order.
Columns 63-64 (Field Record Relation)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Columns must be blank for CONSOLE files.</td>
</tr>
<tr>
<td>01-99</td>
<td>Record-identifying indicator assigned to a record type, or an indicator set on elsewhere in the program.</td>
</tr>
<tr>
<td>L1-L9</td>
<td>Control-level indicator previously used.</td>
</tr>
<tr>
<td>MR</td>
<td>Matching-record indicator.</td>
</tr>
<tr>
<td>U1-U8</td>
<td>External indicator previously set.</td>
</tr>
<tr>
<td>H1-H9</td>
<td>Halt indicator previously used.</td>
</tr>
</tbody>
</table>

Use a record-identifying indicator in columns 63 and 64 to relate a field to a particular record type.

When several record types are specified in an OR relationship, all fields that do not have a field-record-relation indicator in columns 63 and 64 are associated with all record types in the OR relationship. To relate a field to just one record type, enter the record-identifying indicator assigned to that record type in columns 63 and 64 (see Figure 217 on page 582).

Columns 63 and 64 can also be used to specify that the program accept and use data from a particular field only when a certain condition occurs (such as matching records, a control break, or an external indicator is on). Data from the field named in columns 53 through 58 is accepted only when the field-record-relation indicator is on.
The file contains two different types of records, one identified by a 5 in position 1 and the other by a 6 in position 1. FLDC is related by record-identifying indicator 14 to the record type identified by a 5 in position 1. FLDD is related to the record type having a 6 in position 1 by record-identifying indicator 16. This means that FLDC is found on only one type of record (that identified by 5 in position 1) and FLDD is found only on the other type. FLDA is conditioned by indicator 07, which was previously defined elsewhere in the program. FLDB is found on both types because they are not related to any one type by a record-identifying indicator.

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

IFilenameSqNORiPos1NCCPos2NCCPos3NCC.............................................................*
IREPORT AA 14 1 C5
I OR 16 1 C6
I.............................................................PFromTo++DField+L1M1FrPoNeEq...*
I*
I* Indicator 07 is specified elsewhere in the program, and FLDA
I* is made available for processing only when indicator 07 is set on.
I*
I 20 30 FLDB
I 2 10 FLDA 07
I 40 50 FLDC 14
I 60 70 FLDD 16

Figure 217. Field-Record-Relation Indicator

Columns 65-70 (Field Indicators)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-99</td>
<td>Numeric indicator</td>
</tr>
<tr>
<td>H1-H9</td>
<td>Halt indicator (when checking for an error condition in the data)</td>
</tr>
</tbody>
</table>

Use columns 65 through 70 to check the condition of the numeric fields. Use columns 69 and 70 to check the condition of an alphameric field. These columns cannot be used for a data structure. The three conditions are:

- Plus (columns 65 and 66). An indicator entered in columns 65 and 66 turns on if the numeric field named in columns 53 through 58 is greater than zero.
- Minus (columns 67 and 68). An indicator entered in columns 67 and 68 turns on if the numeric field in columns 53 through 58 is less than zero.
- Zero or blank (columns 69 and 70). An indicator entered in columns 69 and 70 turns on if a numeric field named in columns 53 through 58 is all zeros or if an alphabetic field is all blanks. A numeric field that is all blanks turns on an indicator specified for zeros. However, if an alphabetic field is all zeros, the field does not turn on the indicator specified for all blanks.

Columns 65 through 70 must be blank when table or array names are specified in input specifications. However, an entry can be made for an array element.

Field indicators assigned in these columns can also be set on or set off by SETON or SETOF operations in the calculation specifications.

Columns 71-74

Columns 71 through 74 are not used. Leave them blank.

Columns 75-80 (Program Identification)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphabetic with no imbedded blanks. No special characters can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** To be compatible with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 24. Calculation Specifications

Calculation specifications describe the calculations you want done on your data and the order in which you want them done. Each calculation specifications statement can be divided into three parts:

- When the operation is processed (columns 7 through 17). The indicators entered in these columns determine the conditions under which the specified operation is done.
- The kind of operation processed (columns 18 through 53). Entries in these fields describe the kind of operation done and specify the data the operation uses.
- The tests that are made on the results of the operation (columns 54 through 59). The indicators entered in these columns signal the result of the operation and can be used to condition other operations.

Calculation specifications must be specified in the following order: detail, total, subroutine. Write these specifications on the RPG Calculation Specifications sheet (see Figure 218).

Figure 218. RPG Calculation Specifications

© Copyright IBM Corp. 1994

585
### Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

### Columns 3-5 (Lines)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

### Column 6 (Form Type)

A C must appear in column 6 to identify this line as a calculation specification.

### Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to help document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.
## Columns 7-8 (Control Level)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Calculation operation is done at detail calculation time for each program cycle if the indicators in columns 9 through 17 allow it; or if calculation is part of a subroutine.</td>
</tr>
<tr>
<td>L0</td>
<td>Calculation operation is done at total calculation time for each program cycle after total calculation processing has started. <strong>Note:</strong> If no control-level indicators are specified on input specifications, total calculation time processing starts during the second program cycle. If control-level indicators are specified on the input specifications, total calculation time processing starts during the program cycle after the first record containing control fields is processed or at LR time. Totals are always processed at LR time.</td>
</tr>
<tr>
<td>L1-L9</td>
<td>Calculation operation is done when the appropriate control break occurs at total calculation time.</td>
</tr>
<tr>
<td>LR</td>
<td>Calculation operation is done after the last record has been processed.</td>
</tr>
<tr>
<td>SR</td>
<td>Calculation operation is part of a subroutine. A blank entry is also valid for calculations that are part of a subroutine.</td>
</tr>
<tr>
<td>AN, OR</td>
<td>Establishes AND and OR relationships between lines of indicators.</td>
</tr>
</tbody>
</table>

Use columns 7 and 8 to:
- Process total calculation operations when the appropriate control break occurs.
- Process calculation operations that are done only after the last record has been read.
- Indicate that an operation is part of a subroutine. However, columns 7 and 8 can also be blank for calculations that are part of a subroutine.
- Specify that certain lines of indicators are in an AN/OR relationship.

For more information on the two-character entries L0 and L1 through L9, see Chapter 13, “Using Indicators.”

### Subroutine Lines (SR)

An SR entry in columns 7 and 8 indicates that this specification line is part of a subroutine (see “Subroutine Operations” on page 721 in Chapter 27, “Operation Codes”). You do not have to use SR on a calculation specification line that is part of a subroutine; you can leave columns 7 and 8 blank. Subroutine lines must be specified last.

### AN/OR Lines

Use columns 7 and 8 to specify that lines of indicators are in an AN/OR relationship. When you use the AN/OR relationship, many lines of indicators can be grouped together to condition an operation. A maximum of seven AN lines or seven OR lines or any combination thereof can condition an operation. For more information, see Chapter 13, “Using Indicators.”
Columns 7-12 (/EJECT)

Entry          Explanation
/EJECT          The specifications following this entry are to begin on a new page of the
                compiler listing.

The /EJECT specification is not printed on the compiler listing.

Columns 7-12 (/TITLE)

Entry          Explanation
/TITLE          The heading information (such as a title or security classification) that
                follows the /TITLE entry appears at the top of each page of the compiler
                listing. The heading information is entered in columns 14 through 74.

A program can contain more than one /TITLE statement. Each /TITLE statement
provides heading information for the compiler listing until the next /TITLE statement
is read. To print on the first page of the compiler listing, a /TITLE statement must
be the first statement read. Information specified by the /TITLE statement is printed
in addition to compiler heading information.

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

Columns 7-14 (/SPACE)

Entry          Explanation
/SPACE{n}       Line spacing occurs at this point in the compiler listing. Valid
                entries for n are 1 to 12. The number must be left-justified. If you
do not specify n, 1 is assumed.

One blank (b) must come before the value you specify for n. The value you specify
for n indicates the number of blank lines spaced before the next specification line is
printed. If n is greater than the number of lines remaining on the current page, the
next specification line is printed on a new page. If you specify just /SPACE, one
line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line
spacing. The spacing indicated by /SPACE is in addition to the two blank lines that
occur between specification types.

Columns 9-17 (Indicators)

Entry          Explanation
Blank          Operation is processed on every program cycle.
01-99          Field indicators, record-identifying indicators, or resulting indi-
cators assigned elsewhere in the program.
KA-KN, KP-KY   Function-key indicators assigned elsewhere.
L1-L9 Control-level indicators assigned elsewhere. These indicators are on as detail indicators when the first record of a new control group is processed.

LR Last-record indicator.

MR Matching-record indicator.

H1-H9 Halt indicators assigned elsewhere.

U1-U8 External indicators previously set.

OA-OG, OV Overflow indicators previously assigned.

Use columns 9 through 17 to assign indicators that control the conditions under which an operation is done. You can use from one to three separate fields (columns 10 and 11, 13 and 14, and 16 and 17) on each line, one for each indicator. If the indicator must be off to condition the operation, place an N before the appropriate indicator (columns 9, 12, 15).

The indicators specified in columns 9 through 17 on one specification line are in an AND relationship with each other. The indicators on one line or indicators in grouped lines plus the control-level indicator (if used in columns 7 and 8) must all be exactly as specified before the operation is done.

An indicator that is specified in columns 9 through 17 of a calculation specification can also be entered as a resulting indicator on the same line. If the indicator in columns 9 through 17 is on, the calculation is done.

**Relationship between Columns 7-8 and Columns 9-17**

In one program cycle, all operations conditioned by control-level indicators in columns 7 and 8 (total time) are done before operations conditioned by control-level indicators in columns 9 through 17 (see Figure 219).

When a control-level indicator is used in columns 9 through 17 and columns 7 and 8 are not used (detail time), the operation conditioned by the indicator is done only on the record that causes a control break or any higher-level control break.

When a control-level indicator is specified in columns 7 and 8 (total time) and the matching-record indicator (MR) is specified in columns 9 through 17, MR indicates the matching condition of the previous record and not the record just read that caused the control break. After all operations conditioned by the control-level indicators (specified in columns 7 and 8 of the calculation specifications) are done, MR then indicates the matching condition of the record just read.

**Figure 219. Conditioning Operations Using Control Level Indicators**
Assume that indicator 25 represents a record type and that a control level 2 break occurred when record type 25 was read. L1 and L2 are both on. All operations conditioned by the control-level indicators in columns 7 and 8 are processed before operations conditioned by control-level indicators in columns 9 through 17. Therefore, the operation in line 02 occurs before the operation in line 01. The operation in line 01 is done on the first record of the new control group indicated by 25, whereas the operation in line 02 is a total operation done for all records of the previous control group.

The operation in line 02 can be done when the L2 indicator is on provided the other conditions are met. Indicator 10 must be on. The L3 indicator must not be on.

The operation conditioned by both L2 and NL3 is done only when a control level 2 break occurs. These two indicators are used together because this operation is not done when a control level 3 break occurs, even though L2 is also on.

Columns 18-27 (Factor 1)

Use columns 18 through 27 to name the field or to give the actual data (literal) used in the operation processed. See Table 20 on page 713 for a summary of the operation codes.

The entries you can use for factor 1 are:

- The name of any field that has been defined
- Any alphabetic or numeric literal
- Any subroutine, table, array name, or array element
- Any date field name (UDATE, UMONTH, UDAY, UYEAR)
- The special names PAGE, PAGE1, PAGE2, PAGE3, PAGE4, PAGE5, PAGE6, or PAGE7
- The special qualifier *LIKE for the DEFN operation
- Any figurative constant (*BLANK, *BLANKS, *ZERO, *ZEROS)
- A label for a TAG, BEGSR, or ENDSR operation.

The following restrictions apply to entries in factor 1:

- A data structure name can be specified in factor 1 or factor 2, but only for PARM operations.
- A data structure subfield name can be used in factor 1 or factor 2; however, overlapping subfields in a data structure cannot be used in the same calculation. A subfield is considered an overlapping subfield if its From or To position occurs within the From and To positions of another subfield within the same data structure. If factor 1, factor 2, or the result field references a subfield in a data structure that is an array or array element with a variable index, the entire array is used to determine if overlap exists. The same array name can be referenced in the appropriate factors of a calculation specification without violating the overlap rule. See Figure 220 on page 591 for examples of the overlap rule.
- Figurative constants cannot be used with move zone operations, bit operations, or the SET, KEY, SQRT, or DEBUG operation codes.
An entry in factor 1 must begin in column 18.

Entries for factor 1 depend upon the operation code used in columns 28 through 32. Some operations require entries in both factors, some require entries in only one, and some require no entries at all. See “Columns 28-32 (Operation)” on page 596 for more information on operation codes. For information on how to name a subroutine, see “Subroutine Operations” on page 721 in Chapter 27, “Operation Codes.”

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.........................................................*
I*
The data structure DATADS contains subfields and arrays that are defined as overlapping (that is, occupying part of the same area). ARR1 has six elements, and each element is five positions long for a total length of 30. ARR2 has five elements, and each element is six positions long for a total length of 30.
IDATADS DS
I..........................................................PFromTo++DField+LIM1FrPoNeEq...*
I 1  5 ALPHA1
I 6 10 ALPHA2
I 4  8 ALPHA3
I 7  9 ALPHA4
I 6 10 ALPHA5
I 11 140NUM1
I 11 150NUM2
I 11 150NUM3
I 15 201NUM4
I 17 180NUM5
I 21  50 ARR1
I 41  70 ARR2

Figure 220 (Part 1 of 3). Examples of Valid and Invalid Calculations with Overlapping Subfields in a Data Structure
The following individual calculations are valid because the subfields do not overlap.

```
C
MOVE ALPHA1 ALPHA2
C
MOVE ALPHA2 ALPHA1
C
NUM1 ADD NUM4 NUM4
C
Z-ADD1 NUM3
C
Z-ADD5 NUM2
C
Z-ADDNUM5 NUM1
C
MOVE Alpha3
C
MOVE ALPHA1 ARR1,X
```

The following individual calculations are valid because the subfields are determined to be the same area, and running will not cause invalid results.

```
C
MOVE ALPHA2 ALPHA5
C
MOVELALPHA5 ALPHA2
C
NUM2 ADD NUM3 NUM3
C
Z-ADDNUM3 NUM2
```

The following individual calculations are invalid because the subfields occupy part of the same area, and running could cause invalid results.

```
C
MOVE ALPHA1 ALPHA3
C
MOVELALPHA3 ALPHA4
C
NUM1 ADD NUM2 NUM2
C
Z-ADDNUM5 NUM4
C
Z-ADDNUM3 NUM1
```

Figure 220 (Part 2 of 3). Examples of Valid and Invalid Calculations with Overlapping Subfields in a Data Structure
The following individual calculations involve the same array and are valid calculations.

C

    MOVE ARR1,1    ARR1,5
    MOVE ARR1,X    ARR1,2
    MOVE ARR2,X    ARR2,Y

The following individual calculations are valid because the array elements associated with the constant indexes do not overlap.

C

    MOVE ARR1,1    ARR2,1
    MOVE ARR1,2    ARR2,5

The following individual calculations are invalid because the array elements associated with the constant indexes overlap, or variable indexes are specified and the entire array is required to determine overlap.

C

    MOVE ARR2,1    ARR1,5
    MOVE ARR1,X    ARR2,X
    MOVE ARR1,X    ARR2,1
    MOVE ARR1,1    ARR2,X
Literals

A literal is the actual data used in an operation rather than the field name representing that data. A literal can be either alphameric or numeric.

**Alphameric Literals**
Consider the following rules when using an alphameric literal (see Figure 221 on page 595):

- Any combination of characters can be used in an alphameric literal. Blanks are also valid.
- The maximum length of an alphameric literal is eight characters.
- Alphameric literals must be enclosed in apostrophes (').
- An apostrophe required as part of a literal is represented by two apostrophes. For example, the literal O'CLOCK is coded as 'O"CLOCK'.
- Alphameric literals cannot be used for arithmetic operations.

**Numeric Literals**
Consider the following rules when using a numeric literal (see Figure 221 on page 595):

- A numeric literal consists of any combination of the digits 0 through 9. A decimal point or sign can also be included.
- The sign (+ or -), if present, must be the leftmost character. An unsigned literal is treated as a positive number.
- The maximum total length of a numeric literal is 10 characters including the sign and decimal point.
- Blanks cannot appear in a numeric literal.
- Numeric literals must not be enclosed in apostrophes (').
- Numeric literals are used in the same way as a numeric field.
Examples of Alphameric Literals.

'512% DT.'
'FEBRUARY'
'0' CLOK'  
''84'

Examples of Numeric Literals.

12500
12500.00
.001256789
-.01256789

Figurative Constants

The figurative constants *BLANK, *BLANKS, *ZERO, and *ZEROS can be specified as literals. The following rules apply for figurative constants:

- The figurative constants *BLANK and *BLANKS can only be used with alphabetic fields.
- The figurative constants *ZERO and *ZEROS can be used with either alphabetic or numeric fields.
- The length of the figurative constant is assumed equal to the length of the other factor field, if present. Otherwise, the length of the figurative constant is assumed equal to the length of the result field.
- Figurative constants are considered elementary items, and, if used in conjunction with an array, act like a field. For example:

If ARR has four-character elements, each element of ARR contains 0000 after the move is processed.

- The logical placement of a figurative constant in the collating sequence can be altered by specifying an alternative collating sequence.
Columns 28-32 (Operation)

Use columns 28 through 32 to specify the kind of operation to be processed using factor 1, factor 2, and/or the result field. The operation code must begin in column 28. A special set of operation codes must be used to indicate the type of operation to be processed.

Every operation code used requires certain entries on the same specification line. See Table 20 on page 713 for a summary of all the operation codes and the entries required for each code. For further information on the operation codes, see Chapter 27, “Operation Codes.”

The program processes the operations in the order specified on the calculation specifications sheet.

Columns 33-42 (Factor 2)

Use columns 33 through 42 to name the field or to give the actual data (literal) used in the operation processed. See Table 20 on page 713 for a summary of the operation codes.

The entries you can use for factor 2 are:

- The name of any field that has been defined
- Any alphanumerical or numeric literal
- Any subroutine, table, array name, or array element
- Any date field name (UDATE, UMONTH, UDAY, UYEAR)
- The special names PAGE, PAGE1, PAGE2, PAGE3, PAGE4, PAGE5, PAGE6, or PAGE7
- Any figurative constant (*BLANK, *BLANKS, *ZERO, *ZEROS)
- A label for a GOTO or EXSR operation
- A filename for a SETLL, CHAIN, DEBUG, READ, READE, READP, FORCE, ACQ, REL, or NEXT operation
- An EXCPT name for an EXCPT operation
- A subroutine name for an EXIT operation
- An array name for a SORTA operation.

The following restrictions apply to entries in factor 2:

- A data structure name can be specified in factor 1 or factor 2, but only for PARM operations.
- A data structure subfield name can be used in factor 1 or factor 2; however, overlapping subfields in a data structure cannot be used in the same calculation. A subfield is considered an overlapping subfield if its From or To position occurs within the From and To positions of another subfield within the same data structure. If factor 1, factor 2, or the result field references a subfield in a data structure that is an array or array element with a variable index, the entire array is used to determine if overlap exists. The same array name can be referenced in the appropriate factors of a calculation specification.
without violating the overlap rule. See Figure 220 on page 591 for examples of the overlap rule.

- Figurative constants cannot be used with move zone operations, bit operations, or the SET, KEY, SQRT, or DEBUG operation codes.

An entry in factor 2 must begin in column 33.

Entries for factor 2 depend upon the operation code used in columns 28 through 32. Some operations require entries in both factors, some require entries in only one, and some require no entries at all. See “Columns 28-32 (Operation)” on page 596 for more information on operation codes. For information on how to name a subroutine, see “Subroutine Operations” on page 721 in Chapter 27, “Operation Codes.”

### Columns 43-48 (Result Field)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERASE</td>
<td>Delete the CONSOLE file buffer by using the SET operation code.</td>
</tr>
<tr>
<td>Field name, table name, array name, array element, data structure sub-field name, or data structure name</td>
<td>The field specified contains the result of, or is the object of, the operation specified in columns 28 through 32. A data structure name can be specified as a result field only if the operation code in columns 28 through 32 is RLABL, PARM, or POST.</td>
</tr>
<tr>
<td>INxx (xx = any RPG indicator)</td>
<td>The indicator transferred to an external subroutine in an RLABL operation.</td>
</tr>
<tr>
<td>Subroutine name</td>
<td>Name of a subroutine to branch to if the condition specified in xx portion of a CASxx statement is met. A subroutine name can be specified as a result field only if the operation code in columns 28 through 32 is CASxx.</td>
</tr>
</tbody>
</table>

### Erase

Enter ERASE in columns 43 through 48 to blank or delete the entire buffer for the CONSOLE file. The filename of the CONSOLE file must be entered in columns 33 through 42. ERASE indicates to the system that the buffer should be set to blanks just before the system gets a record at the beginning of the next RPG cycle.

### Field Name, Table Name, Array Name, Array Element, or Data Structure

Use columns 43 through 48 to name the field, data structure subfield, table, array, array element, or data structure that holds the result of the operation specified in columns 28 through 32, or that is the field upon which an operation is processed. Use the name of a field, table, array, array element, data structure, or data structure subfield that has already been defined either by the input, extension, or calculation specifications; or define a new field by entering a field name that is not already used. Any field defined in the result field is created when the program is compiled. The result field can be either numeric or alphameric.
A field used in arithmetic operations (see “Columns 28-32 (Operation)” on page 596) or numeric compare operations or a field edited or zero-suppressed by output specifications must be numeric.

A data structure name can be used as the result field only if the operation specified in columns 28 through 32 is RLABL, PARM, or POST. Overlapping subfields in a data structure cannot be used in the same calculation. If factor 1, factor 2, or the result field references a subfield in a data structure that is an array or array element with a variable index, the entire array is used to determine if overlap exists. The same array name can be referenced in the appropriate factors of a calculation specification without violating the overlap rule. See Figure 220 on page 591 for examples of the overlap rule.

The result field name must begin with an alphabetic character in column 43 and contain no blanks or special characters.

If columns 43 through 48 contain the name of a field that is not defined elsewhere, columns 49 through 52 should also contain entries. If the field is defined elsewhere, entries in columns 49 through 52 are not necessary but, if specified, must agree with the previous definition of that field.

<table>
<thead>
<tr>
<th>Columns 49-51 (Field Length)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry</strong></td>
</tr>
<tr>
<td>Blank</td>
</tr>
<tr>
<td>1-256</td>
</tr>
</tbody>
</table>

Use columns 49 through 51 to specify the length of the result field. If the result field is defined elsewhere, no entry is required for the length. However, if the length is specified, it must be the same as the previously defined length, with the same number of decimal positions. If the result field is a new field, consider the form your data is in because the result field must be large enough to hold the largest possible result. If the result field is too small, significant digits can be lost.

For example, to add field A (eight characters long, four decimal positions) to field B (10 characters long, six decimal positions), the result field, field C, must be large enough to contain 11 characters:

```
9999.0000   Field A
0001.111111 Field B
10000.111111 Field C (result field)
```

In this example, field C must be defined as 11 characters long with six decimal positions. Some of the numbers to the right of the decimal could be lost without changing the meaning of the result greatly. However, if field C was defined as 10 characters long with six decimal positions, a significant digit to the left of the decimal would be lost. Field C in this case would be 0000.111111; the meaning of the result has greatly changed.

Figure 222 on page 599 shows how the contents of a result field can change after a multiplication operation, depending on the decimal position (column 52) and field
length (columns 49 through 51) specifications. The result field for a multiply operation should be as long as the sum of the lengths of the two factor fields.

Numeric fields have a maximum length of 15 characters. Alphameric fields can be up to 256 characters long.

If the result field contains the name of a table or array, an entry in these columns is optional. If used, the entry must agree with the length described by the extension specifications.

Multiplication: $98.76 \times 1.234 = 121.86984$

<table>
<thead>
<tr>
<th>Decimal Positions for Result Field (Column 52)</th>
<th>Result Field Length (Columns 49 through 51)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Not permitted

Permitted but inaccurate

Recommended

Figure 222. Result Field Contents Based on Various Field-Length and Decimal-Position Specifications
Column 52 (Decimal Positions)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Alphameric or numeric result field is described elsewhere, or the newly defined result field is alphameric.</td>
</tr>
<tr>
<td>0-9</td>
<td>Number of decimal positions in a newly defined numeric result field.</td>
</tr>
</tbody>
</table>

Use column 52 to indicate the number of positions to the right of the decimal in a numeric result field. If the numeric result field contains no decimal positions, enter a 0 (zero). This column must be blank if the result field is alphameric. This column can be left blank if the result field is numeric but was described by input or calculations specifications. In this case, field length (columns 49 through 51) must also be left blank.

The number of decimal positions must never be greater than the length of the field. The number can, however, be larger or smaller than the number of decimal positions that actually result from an operation. If the number of decimal positions specified is greater than the number of decimal places that actually result from an operation, zeros are filled in to the right. If the number specified is smaller than the number that results from the operation, the rightmost digits are dropped.

Column 53 (Half-Adjust)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Do not half adjust</td>
</tr>
<tr>
<td>H</td>
<td>Half adjust</td>
</tr>
</tbody>
</table>

Use column 53 to indicate that the contents of the result field are half-adjusted (rounded). Half-adjusting is when the single digit to the right of the last decimal position specified is added to the same position in the result field. All decimal positions to the right of the position specified for that field are then dropped (see Figure 223 on page 601).

The half-adjust entry is allowed only with arithmetic operations. See “Columns 28-32 (Operation)” on page 596. However, half-adjusting cannot be specified for a SQRT operation, for an MVR operation, or for a DIV operation followed by an MVR operation.

Note: The algorithm used for half-adjust in System-36 RPG II is not identical to that used on the AS/400.
This calculation line shows a result field being half-adjusted to two decimal positions (2 in column 52 and H in column 53).

Second Position

35.7968 Result of an add operation.
3. Add the digit to the right of the last decimal position specified to the same position in the result field.
35.80xx Drop all decimal positions to the right at the position specified.
35.80 Result after half-adjusting.

Figure 223. Half-Adjust

| Columns 54-59 (Resulting Indicators) |
|-------------------------------|-----------------------------|
| **Entry** | **Explanation** |
| 01-99 | Any two-digit number |
| KA-KN, KP-KY | Any function-key indicator (allowed only with SET or SETOF operation) |
| H1-H9 | Any halt indicator |
| L1-L9 | Any control-level indicator |
| LR | Last-record indicator |
| OA-OG, OV | Any overflow indicator |
| U1-U8 | Any external indicator |
Columns 54 through 59 have two purposes:

- To test the value of the result field after an arithmetic operation or to test the result of a CHAIN, KEY, LOKUP, COMP, READ, READE, READP, CAS, TESTB, TESTZ, ACQ, REL, NEXT, POST, or SHTDN operation. For more information on each specific operation, see Chapter 27, “Operation Codes.”
- To specify the indicators that are turned on or off by the SETON and SETOF operations.

Test Results

You can use an indicator in columns 54 through 59 to test the value of the result field, or to indicate an end-of-file condition, a no-record-found condition, or an exception/error condition. Normally, only the two-character entries 01 through 99 and H1 through H9 are used as resulting indicators for testing. The indicator specified turns on only if the result field satisfies the condition being tested for. If the condition tested for is not met, the indicator is turned off.

You can use three fields (columns 54 and 55, 56 and 57, and 58 and 59) for testing the results. Each field is used to test for different conditions. You can specify testing for any or all conditions at the same time.

For more information on using resulting indicators for testing, see Chapter 13, “Using Indicators.”

Columns 60-74 (Comments)

Use columns 60 through 74 to enter any meaningful comments that will help you understand the purpose of each statement. Comments are not instructions to the RPG program; they serve only as a means of documenting your program.

Columns 75-80 (Program Identification)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphanumeric with no embedded blanks. No special character can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** To be compatible with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 25. Output Specifications

You use output specifications to describe the records and fields in the output file and the conditions under which output operations are to be processed. These specifications can be divided into two general categories:

- File and record identification entries (columns 7 through 37) that describe the output file, the records, and the indicators that condition the output.

- Field description entries (columns 23 through 74) that describe the position and format of data on the output record. These entries must begin one line below the file and record identification entries.

Write these specifications on the RPG Output Specifications sheet (see Figure 224).

Figure 224. RPG Output Specifications

© Copyright IBM Corp. 1994
### Columns 1-2 (Page)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No page number is used.</td>
</tr>
<tr>
<td>01-99</td>
<td>Page number.</td>
</tr>
</tbody>
</table>

Use columns 1 and 2 in the upper right corner of each sheet to number the specifications sheets, in ascending order, for your job. You can use more than one of each type of sheet, but keep all sheets of the same type together.

### Columns 3-5 (Line)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>No line number is used.</td>
</tr>
<tr>
<td>Any numbers</td>
<td>Line numbers.</td>
</tr>
</tbody>
</table>

Use columns 3 through 5 to number the lines on each page. Columns 3 and 4 are preprinted on each sheet so, in most cases, line numbering is already done.

Page and line numbers are optional entries and are not required to successfully run an RPG program. Columns 1 through 5 are checked for ascending order, and RPG prints an S on the RPG listing for any statement that is out of order. Duplicate sequence numbers do not cause a sequence error.

### Column 6 (Form Type)

An O must appear in column 6 to identify this line as an output specification.

### Column 7 (Comments)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Comment line</td>
</tr>
</tbody>
</table>

Use an asterisk in column 7 to identify the line as a comment line. Use comments throughout your program to help document the purpose of a certain section of coding. You can use any character in a comment line. Comments are not instructions to your program; they only document your program.

### Columns 7-12 (/EJECT)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EJECT</td>
<td>The specifications following this entry are to begin on a new page of the compiler listing.</td>
</tr>
</tbody>
</table>

The /EJECT specification is not printed on the compiler listing.
### Columns 7-12 (/TITLE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TITLE</td>
<td>The heading information (such as a title or security classification) that follows the /TITLE entry appears at the top of each page of the compiler listing. The heading information is entered in columns 14 through 74.</td>
</tr>
</tbody>
</table>

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until the next /TITLE statement is read. To print on the first page of the compiler listing, a /TITLE statement must be the first statement read. Information specified by the /TITLE statement is printed in addition to compiler heading information.

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

### Columns 7-14 (/SPACE)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SPACEn</td>
<td>Line spacing occurs at this point in the compiler listing. Valid entries for n are 1 to 12. The number must be left-justified. If you do not specify n, 1 is assumed.</td>
</tr>
</tbody>
</table>

One blank (b) must come before the value you specify for n. The value you specify for n indicates the number of blank lines spaced before the next specification line is printed. If n is greater than the number of lines remaining on the current page, the next specification line is printed on a new page. If you specify just /SPACE, one line is spaced.

/SPACE is not printed on the compiler listing but is replaced by the actual line spacing. The spacing indicated by /SPACE is in addition to the two blank lines that occur between specification types.

### Columns 7-14 (Filename)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid filename</td>
<td>Same filename that appears on the file description specifications for the output, combined, update, or add file</td>
</tr>
</tbody>
</table>

Use columns 7 through 14 to identify the output file you want to describe. The filename must begin in column 7.

The filename should be specified only on the first line. However, if another output file is specified and more specifications are then required for the first file, the first filename must be repeated in columns 7 through 14 (see Figure 225 on page 606).
Figure 225. Specifying Filename

Note: The filename need not be repeated in columns 7 through 14 unless another output file is specified and then further specifications are required for the first file.

Figure 225. Specifying Filename
Columns 14-16 (AND/OR)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND or OR</td>
<td>AND/OR indicates a relationship between lines of output indicators.</td>
</tr>
</tbody>
</table>

Use columns 14 through 16 to specify AND/OR lines for output operations. For an AND relationship, the condition for all indicators must be satisfied before the output operation is done. You can use any number of AND lines for an output operation. For an OR relationship, only one condition is met between several indicators or groups of indicators before the output operation is done. You can use any number of OR lines for an output operation.

You can use AND and OR lines to condition entire output lines, but you must not use them to condition fields. However, you can condition an output field with more than three indicators by using the SETON operation in calculations.

Column 15 (Type)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Heading records</td>
</tr>
<tr>
<td>D</td>
<td>Detail records</td>
</tr>
<tr>
<td>T</td>
<td>Total records</td>
</tr>
<tr>
<td>E</td>
<td>Exception records (lines written during calculation time)</td>
</tr>
</tbody>
</table>

Use column 15 to indicate the type of record written. Column 15 must have an entry for every output record (see Figure 225 on page 606).

Heading Records (H)

Heading records usually contain constants identifying information such as column headings, page number, and date.

Detail Records (D)

Detail records usually contain data that comes directly from the input record or is the result of calculations done on data from the input record.

Total Records (T)

Total records usually contain data that is the end result of specific calculations on several detail records. Total output cannot be specified for primary or secondary update files. Records can be added to indexed primary and secondary files at total time if add is specified (A in column 66) on the file description specifications.

Exception Records (E)

Exception records are written during calculation time. Exception records can be specified only when the operation code EXCPT is used. See Chapter 27, “Operation Codes,” for more information on the EXCPT operation code.
COLUMNS 16-18 (ADD/DEL)

Columns 16-18 (ADD/DEL)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add a record to an indexed, direct, or sequential file defined as an input, output, or update file.</td>
</tr>
<tr>
<td>DEL</td>
<td>Delete the last record read on the identified update file.</td>
</tr>
</tbody>
</table>

**ADD**

When ADD is specified in columns 16 through 18 to add a record to an indexed, direct, or sequential file, column 66 of the file description specifications must contain an A for the file to which records are being added. The output device for this file must be DISK.

The ADD entry must not be used in an OR line. An ADD entry in columns 16 through 18 of the previous line also applies to the record in the OR relationship. For a detailed description of adding records to a file, see "Column 66 (File Addition)" on page 513 in Chapter 19, "File Description Specifications," or "Adding Records to a Sequential File" on page 104 in Chapter 6, "Using a DISK File."

**DEL**

If a record is deleted from a file, the file must be defined as delete-capable when it is built. For more information on defining a delete-capable file, see the OCL FILE statement in the System Reference for the System/36 Environment. If you attempt to delete a record from a file that is not delete-capable, an run-time error message is displayed.

DEL must be specified in columns 16 through 18 of the main output record line. DEL applies to all the OR extensions to the main line. When records are deleted from a file, the file must be defined as an update file (column 15 of the file description specifications contains U).

**Note:** Record deletion is not dependent on the file organization and mode of processing entries.

Records are not physically removed from a file when they are deleted. Instead, deleted records are filled with hexadecimal FFs. That is, all the bits for every character in the deleted record are set on.
Column 16 (Fetch Overflow or Release)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Fetch overflow routine</td>
</tr>
<tr>
<td>R</td>
<td>Release the device (display station or ICF session) after output</td>
</tr>
</tbody>
</table>

Fetch Overflow

Use fetch overflow when printing a particular line causes overflow and not enough space is left on the page to print the remaining detail, exception, or total output lines. 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 and total line.

Use column 16 to specify fetch overflow for a PRINTER file only. Column 16 of each OR line must contain an F if the overflow routine is used for each record in the OR relationship. Fetch overflow cannot be used when an overflow indicator is specified in columns 23 through 31 on the same specification line. If this occurs, the overflow routine is not fetched. Specifying fetch overflow allows you to alter the RPG overflow logic (see “Columns 33-34 (Overflow Indicator)” on page 504 in Chapter 19, “File Description Specifications”). You can advance forms when total, detail, or exception records are printed instead of waiting for the usual time in the program cycle. The fetched overflow routine does not automatically cause forms to advance; that is, the entry in columns 21 and 22 of the output specifications must contain a two-digit entry that is less than the number of the lines that the printer is currently on. Fetching the overflow routine can prevent printing over the page perforation and can use as much of the page as possible. For more information on fetch overflow, see Chapter 8, “Using a PRINTER File.”

Release

You can release a device from your program after output to that device has been written. To release the device, enter an R in column 16. You can specify OR lines; however, column 16 must contain an R for each OR line. The device is released when that output specification is read during the output operations. If you specify a format name on a field description line for the record that contains an R in column 16, the format is written, and then the device is released.

If the WORKSTN file is a primary file and the program does not have a NEP attribute, RPG sets on the last-record (LR) indicator when all devices have been released. If the program has a NEP attribute, RPG sets on the last-record (LR) indicator when all devices have been released and the system operator enters a command to stop the system.

Note: For WORKSTN files, a device can be either a display station or an ICF session.
Columns 17-22 (Spacing and Skipping)

Column 17 (Space Before)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>Number of lines spaced before a line is displayed for a CRT file or printed for a PRINTER file.</td>
</tr>
</tbody>
</table>

Column 18 (Space After)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>Number of lines spaced after a line is displayed for a CRT file or printed for a PRINTER file.</td>
</tr>
</tbody>
</table>

Columns 19-20 (Skip Before)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>For a CRT file display screen is blanked immediately.</td>
</tr>
<tr>
<td>01-99</td>
<td>For PRINTER files skip to lines 01 to 99 before printing.</td>
</tr>
<tr>
<td>A0-A9</td>
<td>For PRINTER files skip to lines 100 to 109 before printing.</td>
</tr>
<tr>
<td>B0-B2</td>
<td>For PRINTER files skip to lines 110 to 112 before printing.</td>
</tr>
</tbody>
</table>

Columns 21-22 (Skip After)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-99</td>
<td>For PRINTER files skip to lines 01 to 99 after printing.</td>
</tr>
<tr>
<td>A0-A9</td>
<td>For PRINTER files skip to lines 100 to 109 after printing.</td>
</tr>
<tr>
<td>B0-B2</td>
<td>For PRINTER files skip to lines 110 to 112 after printing.</td>
</tr>
</tbody>
</table>

Use columns 17 through 22 to specify line spacing and skipping for PRINTER and CRT files. Spacing refers to advancing one line at a time, and skipping refers to jumping from one print line to another.

If you make an incorrect entry in these columns, the compiler drops the entry and assumes a blank specification. If columns 17 through 22 are blank, single spacing occurs after each line is printed. You can specify different spacing and skipping for OR lines. If you do not specify spacing or skipping entries for the OR line, spacing and skipping are done according to the specifications for the line that comes before the OR line. You cannot specify spacing or skipping on AND lines.
<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-99</td>
<td>Any resulting indicator, field indicator, or record-identifying indicator previously specified.</td>
</tr>
<tr>
<td>KA-KN, KP-KY</td>
<td>Any function-key indicator previously specified in a SET operation or used with a WORKSTN file.</td>
</tr>
<tr>
<td>L0-L9</td>
<td>Any control-level indicators previously specified.</td>
</tr>
<tr>
<td>H1-H9</td>
<td>Any halt indicators previously specified.</td>
</tr>
<tr>
<td>U1-U8</td>
<td>Any external indicator set prior to the program running.</td>
</tr>
<tr>
<td>OA-OG, OV</td>
<td>Any overflow indicator previously assigned to this file.</td>
</tr>
<tr>
<td>MR</td>
<td>Matching-record indicator.</td>
</tr>
<tr>
<td>LR</td>
<td>Last-record indicator.</td>
</tr>
<tr>
<td>1P</td>
<td>First-page indicator. The first-page indicator cannot be specified for a WORKSTN file.</td>
</tr>
</tbody>
</table>

You can specify one indicator in each of the three separate output indicator fields (columns 23 through 25, 26 through 28, and 29 through 31). If these indicators are on, the output operation is done. An N in the column that comes before each indicator (column 23, 26, or 29) means that the output operation is done only if the indicator is not on. This is a negative indicator. No output line should be conditioned solely by negative indicators. At least one of the indicators should be positive. You should not specify solely negative indicators to condition a heading or detail operation because the operation is processed at the beginning of the program cycle when the first page lines are written.

If no output indicators are specified, the line is produced at output every time that that record is checked for output. If no output indicators are specified on a heading or detail line, that record is also produced as output at the beginning of the program cycle.

If you need more than three indicators to condition an output operation, use an AND line or an OR line. For more information, see “Columns 14-16 (AND/OR)” on page 607.
Columns 32-37 (Field Name)

In columns 32 through 37, use one of the following types of names to specify each field that is written out:

- Any field name or data structure name that you used earlier in this program
- The special words PAGE, PAGE1 through PAGE7, *PLACE, UDATE, UDAY, UMONTH, or UYEAR
- A table name, array name, or array element
- An EXCPT name.

Field Names

The field names you use must be the same as the field names on the input specifications (columns 53 through 58) or the calculation specifications (columns 43 through 48). Do not enter a field name if a constant is used in columns 45 through 70. If a field name is entered in columns 32 through 37, columns 7 through 22 must be blank.

Fields can be listed on the specifications sheet in any order because the order in which they appear on the output record is determined by the entry in columns 40 through 43. However, the fields are usually listed in order. If fields overlap, the last field specified is the only field completely written.

The sign (+ or -) of a numeric field is in the units position (rightmost digit). The units position prints as a letter unless the field is edited. See “Column 38 (Edit Codes)” on page 617 or “Columns 45-70 (Constant or Edit Word)” on page 619.

Rules for Field Names

A field name can be from one to six characters long. The first character must be alphabetic. The remaining characters can be any combination of alphabetic characters.

Special Words

Page Numbering (PAGE, PAGE1-PAGE7)

PAGE is a special word that causes automatic numbering of the pages. Enter the word PAGE or PAGE1 through PAGE7 in these columns if the pages are numbered. When a PAGE field is named in these columns without being defined elsewhere, it is assumed to be a four-digit, numeric field with zero decimal positions. Leading zeros are replaced with blanks automatically. A PAGE field can also be defined on input or calculation specifications as a numeric field from 1 to 15 digits long, with zero decimal positions.

The page number starts with 0001 unless otherwise specified, and 1 is automatically added for each new page. See “Columns 53-58 (Field Name)” on page 576 in Chapter 23, “Input Specifications,” for information concerning page numbering that starts at a number other than 1.

Page numbering can be restarted at any point in a job. To do this, set the PAGE field to zero before it is printed by specifying either blank after in column 39 or an output indicator. If the status of the indicator is as specified, the PAGE field is
reset to zero, and 1 is added to the PAGE field before it is printed (see Figure 226 on page 613).

The eight possible PAGE entries (PAGE, PAGE1 through PAGE7) may be needed for numbering different types of output pages or for numbering pages for different PRINTER files.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaNO1NO2NO3Excptn..............................................*
OPRINT H 01 L1
0 15 PAGE 1 75
Figure 226. Resetting the PAGE Fields to Zero

When indicator 15 is on, the PAGE field is reset to zero and a 1 is added before the field is printed. When 15 is off, a 1 is added to the contents of the PAGE field before it is printed.

Repeating Output Fields (*PLACE)
*PLACE is a special RPG word that allows you to write the same fields in several locations on one record without naming the fields and giving their end position each time the fields are written. The fields repeated by means of *PLACE are written ending in the position specified in columns 40 through 43 of the same specifications line. For example, if Fields A, B, and C appear twice on one record, the fields can be specified in two ways:

- Define each field and its corresponding end position each time the field is to be written (see Figure 227 on page 614).
- Use the special word *PLACE (see Figure 227 on page 614 and Figure 228 on page 615).

Both coding methods shown in Figure 227 on page 614 produce a record that looks like this:

<table>
<thead>
<tr>
<th>Table 18. Example of Code for Replacing Output Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ending Record Positions</td>
</tr>
<tr>
<td>Fields</td>
</tr>
</tbody>
</table>
To repeat an output field, each field can be defined each time it is printed or written to disk.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn....................................*
0OUTPUT D
0.................N01N02N03Field+YBEnd+PConstant/editword++++++++++...*
 0             FIELDA  10
 0             FIELDB  20
 0             FIELDc  30
 0             FIELDA  40
 0             FIELDB  50
 0             FIELDc  60
 0             FIELDD  75
```

Or the special word `PLACE` can be used to repeat a group of fields.

*Figure 227. Writing Fields Twice on the Same Record*
*PLACE can also be used to print the same group of fields several times on the same line. Fields A, B, and C are to be printed four times on one line as shown above. They are printed once when they are named and once for every *PLACE entry.

*PLACE is specified after the fields that are printed several times on the same line (below). All fields to which *PLACE applies appear on the same record. FIELDD, which appears on the total record, is not affected by *PLACE.

Notice that an end position is given for every *PLACE. Fields A, B, and C have a total length of 15 characters. Therefore, the end positions given for the *PLACE entries allow room for the printing of 15 characters. This eliminates any overlapping.

Figure 228. Example of Using *PLACE
When you specify *PLACE, all fields named for each record type (H/D/T/E) are written as usual in the location specified. The entry *PLACE then writes all of these fields ending at the position specified in columns 40 through 43 of the *PLACE specification. When you specify *PLACE, consider the following:

- *PLACE must be specified after the field names that are written in different positions in one record (see Figure 228 on page 615).
- *PLACE writes all fields within a record type, not just the field name on the line immediately above the *PLACE entry.
- *PLACE must appear on a separate specification line each time a field or a group of fields is written.
- An end position no greater than 256 must be specified for every *PLACE line. Allow enough space to write all fields (see Figure 228 on page 615); otherwise, overlapping occurs.
- Multiple or successive *PLACE entries can be specified if the fields preceding the first *PLACE specification are repeated more than once.
- The leftmost position of the fields written by the *PLACE specification is always assumed to be position 1.
- Additional fields or constants can be specified after the *PLACE specification and are not affected by any preceding *PLACE specification.

**Note:** Attempts to use the *PLACE function for other than its defined purpose may produce unpredictable results.

**Date Fields (UDATE, UMONTH, UDAY, UYEAR)**

To have the date printed on a report or program listing, use special words UDATE, UMONTH, UDAY, or UYEAR. UDATE is a six-digit value derived from the current program date in the current date format. Column 19 is used to specify which two digits of the six-digit UDATE value are to be interpreted in an RPG program as the year (UYEAR), month (UMONTH) and day (UDAY). The date fields are established by the RPG programmer at job setup time. UDATE contains the program date which may not be the same as the date in the result field of the TIME operation. The result field of the TIME operation contains the system date. See the System Reference for the System/36 Environment for a complete discussion of the system date, program date, and the OCL DATE statement. The following rules apply to date fields:

- UDATE prints a six-character numeric date field in one of three formats:
  
  - Month/day/year
  
  - Year/month/day
  
  - Day/month/year

  Use columns 19 and 20 of the control specification to specify the date format and the editing done. If columns 19 and 20 are blank, the date format is determined by the contents of column 21 of the control specification.
- Use UDAY for the day only, UMONTH for the month only, and UYEAR for the year only.
- These fields cannot be changed by any operations specified in the program. Therefore, these fields are generally used only in compare and test operations.

**EXCPT Names**

When the record type is an exception record (indicated by an E in column 15), a name can be placed in columns 32 through 37 of the record line. The EXCPT operation can specify the name assigned to a group of records written. This name is called an EXCPT name. An EXCPT name must follow the rules for field names. Also, an EXCPT name cannot be the same as a filename, field name, data structure name, array name, table name, label, or subroutine name. A group of any number of output records can use the same EXCPT name, and the records do not have to be consecutive records.

When the EXCPT operation is specified without an EXCPT name, only those exception records without an EXCPT name are checked and written if the conditioning indicators are satisfied.

When the EXCPT operation specifies an EXCPT name, only those exception records with that EXCPT name are checked and written if the conditioning indicators are satisfied.

The EXCPT name is specified on the main record line and applies to all AND/OR lines.

---

**Column 38 (Edit Codes)**

Use column 38 to:
- Suppress leading zeros in a numeric field
- Omit a sign from the low-order position of a numeric field
- Punctuate a numeric field without establishing an edit word.

For more information on edit codes, see Chapter 16, “Editing Numeric Fields.”
Column 39 (Blank After)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Field is not reset.</td>
</tr>
<tr>
<td>B</td>
<td>Field specified in columns 32 through 37 is reset to blank or zero after the output operation is complete.</td>
</tr>
</tbody>
</table>

Use column 39 to reset a numeric field to zeros or an alphameric field to blanks. If the field is conditioned by indicators in columns 23 through 31, the blank after is also conditioned. This column must be blank for look-ahead and UDATE fields.

Resetting fields to zeros is useful when totals are accumulated and written for each control group in a program. After the total is accumulated and written for one control group, the total field can be reset to zeros before accumulation begins on the total for the next control group.

If blank after (column 39) is specified for a field to be written more than once, the B should be entered on the last line specifying output for that field. When blank after is specified with a table name, the field that is blanked contains the last element found by a successful LOKUP. If no LOKUP or no successful LOKUP occurred, the first element of the table is blanked.

If the file description specifications for the file to which the field on this output specification is to be written contains an external indicator in columns 70 and 71, you may want to use the same external indicator in columns 23 through 31 of this specification to prevent the field from being blanked when the file is not being used by the program.

Columns 40-43 (End Position in Output Record)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Place after prior field</td>
</tr>
<tr>
<td>1-9999</td>
<td>End position for DISK or SPECIAL file</td>
</tr>
<tr>
<td>1-4075</td>
<td>End position for BSCA file</td>
</tr>
<tr>
<td>1-1919</td>
<td>End position for WORKSTN file</td>
</tr>
<tr>
<td>1-225</td>
<td>End position for 225-position printer</td>
</tr>
<tr>
<td>1-79</td>
<td>End position for CRT file</td>
</tr>
<tr>
<td>K1-K8</td>
<td>Length of format name for a WORKSTN file</td>
</tr>
</tbody>
</table>

Use columns 40 through 43 to define the end position of a field or constant on the output record. All entries in these columns must end in column 43. Enter only the position of the rightmost character in the field or constant.

Note: If columns 40 through 43 are left blank, the field or constant is placed in the output record immediately following the field specified in the previous output specification for that record. If no previous field specification exists for the record, the high-order position of the field is placed in position 1. A blank end position with *PLACE means the *PLACE is ignored.
Column 44 (Packed-Decimal or Binary Field)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Field is zoned-decimal numeric data or alphanemic data. Leave this column blank for nondisk files.</td>
</tr>
<tr>
<td>P</td>
<td>Field is written on disk in packed-decimal format.</td>
</tr>
<tr>
<td>B</td>
<td>Field is written on disk in binary format.</td>
</tr>
</tbody>
</table>

Use column 44 to specify if a numeric field (decimal number) is written to disk or to ICF output in packed-decimal or binary format. Packed-decimal and binary fields cannot be displayed or printed; these fields can be written only to disk or to ICF output. Column 44 must be blank for *PLACE.

Decimal numbers can be left in the zoned-decimal format. However, for more efficient use of disk space, convert decimal numbers into packed-decimal or binary format. When binary output is specified, a numeric field one to four digits long (zoned-decimal in storage) is converted into a two-byte binary field when it is written on disk; a numeric field five to nine digits long is converted into a four-byte binary field. When packed-decimal output is specified, a byte of disk storage (except for the low-order byte) can contain two decimal numbers. See “Column 43 (Packed-Decimal or Binary Field)” on page 571, for a description of how data fields are represented in zoned-decimal, packed-decimal, and binary formats.

Columns 45-70 (Constant or Edit Word)

Use columns 45 through 70 to specify a constant, the format name for a WORKSTN file, or an edit word. If you are using edit codes, you can also use columns 45 through 47 to specify a floating currency symbol or asterisk fill.

Constants

A constant is any unchanging information that is to appear on a report. Constants are usually words used for report headings or column headings.

The following rules apply to constants (see Figure 229 on page 620 for examples):

- Field name (columns 32 through 37) must be blank.
- A constant must be enclosed in apostrophes. Enter the leading apostrophe in column 45.
- An apostrophe in a constant must be represented by two apostrophes. For example, if the word you’re appears in a constant it must be coded as ‘YOU’RE’.
- Numeric data can be used as a constant.
- Up to 24 characters of constant information can be placed in one line. Additional lines can be used, but each line must be treated as a separate line of constants. The end position is specified in columns 40 through 43. If no end position is specified, the constant is placed in the output record immediately following the field or constant specified in the previous output specification line for that record (see “Columns 40-43 (End Position in Output Record),” in this chapter).
Figure 229. Examples of Output Constants

Format Name

The name of the display format that is used by the WORKSTN file must be specified in columns 45 through 54. One format name is required for each output record for the WORKSTN file; the specification of more than one format name per record is not allowed. The format name must be enclosed in apostrophes. This is the same name that is specified in columns 7 through 14 of the S specification line on the display screen format specifications. You must also enter Kn in the rightmost of columns 40 through 43, where n is the length of the format name. For example, if the format name is FORM1, enter K5 in columns 42 and 43.

For more information on the display screen format, see Chapter 7, “Using a WORKSTN File.”

Note: The output specifications line containing the format name cannot be conditioned by any indicators.

Edit Words

See Chapter 16, “Editing Numeric Fields,” for a complete discussion on edit words.

Columns 71-74

Columns 71 through 74 are not used. Leave them blank.

Columns 75-80 (Program Identification)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Program identification defaults to the program name specified on the control specification.</td>
</tr>
<tr>
<td>Any valid program name</td>
<td>Program identification. The first character must be alphabetic but cannot be #, $, or @. The remaining characters must be alphanemic with no imbedded blanks. No special character can be used.</td>
</tr>
</tbody>
</table>

Columns 75 through 80 can contain any characters. These columns can contain the program name used in the control specification, or they can contain any other
characters to identify a certain portion of the program. These entries are ignored by the compiler but appear in the source program listing.

**Note:** To be compatible with other RPG systems, the specifications sheets show only 80 positions for each statement. However, each statement in an RPG source program can contain up to 91 characters. Columns 81 through 91 are available for comments.
Chapter 26. The Auto Report Feature

An auto report source program is composed of the following specifications:

- The auto report options specification
- Auto report /COPY statements coded as input specifications
- Auto report page heading specifications coded as both record-description and field-description specifications
- Auto report output specifications coded on both record-description specifications and field-description specifications
- Standard RPG specifications.

The *AUTO page-heading function and the *AUTO output function provide simplified methods of describing printed output. These functions are requested when the characters *AUTO are present in columns 32 through 36 of a record description specification on the standard RPG output specifications. *AUTO can be entered on a heading, detail, or total specification (H, D, or T in column 15), but not on an exception output specification (E in column 15). Use *AUTO with only one PRINTER file in the program.

Standard RPG output specifications are divided into two general types (see Figure 230 on page 624):

- **Record-description specifications** (columns 7 through 31) describe when and where the output line is to be printed. One record-description specification is required for each type of line to be printed. Only the first record-description for a file need contain a filename in columns 7 through 14.

- **Field-description specifications** (columns 23 through 74) following a record-description specification tell when, where, and how each item of data (field or literal) is printed on the output record. There can be several field-description specifications following a record-description specification.

Auto report page headings and auto report output specifications are also divided into the same two types: record-description specifications and field-description specifications. However, the entries on these specifications are used differently from the entries on the standard RPG specifications.

The following output specifications are not changed when they are used with *AUTO:

- Columns 1-2 (page)
- Columns 3-5 (line)
- Column 6 (form type)
- Columns 75-80 (program identification).

Columns 71 through 74 must always be blank on auto report output specifications.
Auto report is a feature of RPG that makes it easier to write RPG programs. An auto report program is basically a brief program that contains instructions that copy previously programmed files and members into itself. You can also enter RPG specifications as for a regular program. When an auto report program is processed, the result is a complete RPG program, which can be compiled.

Specific auto report statements control the three separate functions of auto report, which can be used in any combination:

- The /COPY statement allows you to copy a source member, containing a group of RPG source specifications, into an RPG source program. Use /COPY so that you do not have to repeatedly code identical or nearly identical specifications that are used in several programs. This is the most important benefit of using auto report.
- The *AUTO page heading specification provides a simplified method of coding page headings.
- The *AUTO output specification provides a simplified method of coding output specifications.

Figure 231 on page 625 is an overview of what the auto report feature does.
The Auto Report Source Program

To use the auto report feature, you must first create an auto report source program.

The source program would consist of:

- Auto report options specification
  
  This statement is not required, but if it is used, it must be the first statement in the program. If the options specification is omitted, the system assumes all options are blank. The options specification cannot be coded into a source program that is copied into the auto report program.

- *AUTO page headings and *AUTO output specifications

- Standard RPG specifications

- Auto report /COPY statements coded into the auto report source program, with or without modifier statements

- Standard RPG specifications, including arrays and tables, and *AUTO specifications that are copied from the library by the auto report copy function.
Figure 232 on page 626 shows an example of an auto report source program, and Figure 233 on page 627 shows the calculation and output specifications created by the *AUTO specifications.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn..............................................
OPRINTER H *AUTO
0..............N01N02N03Field+YBEnd+PConstant/editword+++++++++++.*
 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'

Figure 232. *AUTO Output Specifications That Create the Calculation and Output Specifications Shown in Figure 26-2
### Calculations to Roll Totals for SOLDVA and VALUE Fields

<table>
<thead>
<tr>
<th>Calculations</th>
<th>Roll Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLDV2 ADD SOLDV1 SOLDV2</td>
<td>92</td>
</tr>
<tr>
<td>VALUE2 ADD VALUE1 VALUE2</td>
<td>92</td>
</tr>
<tr>
<td>SOLDVR ADD SOLDV2 SOLDVR</td>
<td>92</td>
</tr>
<tr>
<td>VALUE2 ADD VALUE VALUE2</td>
<td>92</td>
</tr>
<tr>
<td>SOLDVR ADD SOLDV2 SOLDVR</td>
<td>92</td>
</tr>
</tbody>
</table>

### Calculations to Roll Total Output Specifications

<table>
<thead>
<tr>
<th>Calculations</th>
<th>Roll Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLDV1KB ADD SOLDVA SOLDV1</td>
<td>92</td>
</tr>
<tr>
<td>VALUE1KB ADD VALUE VALUE1</td>
<td>92</td>
</tr>
</tbody>
</table>

### Calculations to Roll Column Headings

<table>
<thead>
<tr>
<th>Column Headings Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON-HAND</td>
</tr>
<tr>
<td>NUMBER</td>
</tr>
</tbody>
</table>

### Calculations to Roll Detail Output Specifications

<table>
<thead>
<tr>
<th>Detail Output Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLDQYK</td>
</tr>
<tr>
<td>SOLDVAKB</td>
</tr>
<tr>
<td>ONHANDK</td>
</tr>
</tbody>
</table>

### Calculations to Roll Total Output Specifications

<table>
<thead>
<tr>
<th>Total Output Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLDV2KB</td>
</tr>
<tr>
<td>SOLDVRKB</td>
</tr>
</tbody>
</table>

---

Figure 233. Calculation and Output Specifications Created by the *AUTO Specifications in Figure 26-3*
Compiling the Auto Report Source Program

When you compile an auto report source program, it first creates the complete RPG source program and then compiles it. The complete RPG source program contains all the RPG specifications in a special format and in the order required by the RPG compiler.

Format of Created Specifications

The created specifications have the following format:

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>Sequence number of the specification. This number starts at 0010 for the control specification and increases by 0010 for each specification that follows. If the program has more than 999 specifications, the sequence starts again at 0000.</td>
</tr>
<tr>
<td>5</td>
<td>Code that identifies the specification as follows:</td>
</tr>
<tr>
<td></td>
<td>Blank Standard RPG specification in the auto report program</td>
</tr>
<tr>
<td></td>
<td>C Specification copied from the library member specified in the /COPY statement</td>
</tr>
<tr>
<td></td>
<td>M Specification copied from the library member specified in the /COPY statement and modified</td>
</tr>
<tr>
<td></td>
<td>E Specification created by auto report</td>
</tr>
<tr>
<td>6-80</td>
<td>Standard RPG specification</td>
</tr>
</tbody>
</table>

Compile-time arrays and tables are not changed by auto report; they remain in the standard format for an array or table record.

Order of Created Specifications

Auto report creates the specifications in the order required by the RPG compiler. When specifications are included by means of a /COPY statement, those specifications are placed immediately after the /COPY statement. Then, after all specifications are copied but before auto report creates RPG specifications from the H-*AUTO (heading records) and D/T-*AUTO (detail/total records) specifications, the entire auto report source program is sorted into the following order:

1. Control specification
2. File description specifications
3. Extension specifications
4. Line counter specifications
5. Telecommunications specifications
6. Input specifications; sorted input records first, then data structures
7. Calculation specifications (in the following order: detail, L0, L1 through L9, LR, and subroutines)
8. Output specifications
9. Arrays and tables loaded at compilation time, which must be placed last among the input statements to auto report.
Calculation Specifications
Auto report places the created calculation specifications in the following order:

1. Detail calculations that you code
2. EXSR statement for the created subroutine
3. Total calculations created by auto report, grouped in order by level (all L0 calculations, then all L1 calculations, and so on)
4. Total calculations that you code
5. Subroutines that you code
6. Created RPG subroutine that accumulates the lowest-level total.

Note: If the /COPY statement copies a subroutine that contains a BEGSR operation but no ENDSR operation, incorrect sorting of the created RPG source program can result. Incorrect sorting can also result if invalid characters are coded as control-level indicators in columns 7 and 8.

Output Specifications
Output heading specifications created for H-*AUTO specifications appear in the order in which they are coded on the output specifications in relation to other RPG and *AUTO output specifications for the file.

Normally, RPG output specifications created from a D/T-*AUTO specification are in the following order:

1. Heading specifications created for column headings
2. Detail specifications
3. Total specifications, with the lowest level first and LR last.

This group of specifications is placed in the same relative position in the program as the original D/T-*AUTO specification. All other RPG output specifications remain in their original order.

However, if you specify a normal RPG total output specification conditioned by a positive control-level indicator (no N in column 23) in columns 24 and 25 for the file that has a D/T-*AUTO specification, all output specifications in the program are sorted into the following format:

1. All heading, detail, and exception output specifications remain in the order in which they are coded in the created source program. Total specifications that are not conditioned by a positive control-level indicator in columns 24 and 25 remain as they were in the program.

2. Total specifications that are conditioned by a positive control-level indicator in columns 24 and 25 are sorted into ascending order according to the control-level indicator in columns 24 and 25, with LR last.

See “Examples of Using Auto Report” on page 686 for examples of created specifications.

Comment Statements
Comment statements (identified by an asterisk in column 7) are allowed among the statements read by auto report. However, the sorting of RPG specifications is based on the contents of column 6; therefore, comments may be sorted into an unexpected order. To ensure that comments remain with the correct specification, place them after that specification and put the same entry in column 6.
Restriction
The order of arrays and tables does not change when the source specifications are sorted. Therefore, when arrays and tables are included from a library member, they may occur in an incorrect order after the sort. For example, if the auto report source specifications contain a table for translating a file or for changing the collating sequence of characters, then any compile-time arrays or tables included from a library member are out of order. That is, the included arrays or tables are placed ahead of the table for translating files. Compile-time arrays and tables must be loaded in the following order:

1. Tables for translating files
2. Tables for changing the collating sequence of characters
3. Compile-time arrays and tables in the order described on the extension specifications.

A solution to this restriction is to place the tables for translating files and for changing the collating sequence of characters in a library member, and then to copy them from the library member before any other compile-time tables and arrays are copied. This procedure ensures that the tables for translating files and for changing the collating sequence of characters are the first compile-time tables in the created RPG source program.

Auto Report Option Specifications (U)
Specify options for the auto report program on the RPG auto report specifications as shown below:

The auto report option specifications are not required in the auto report program. If present, they must appear as the first specifications in the program. If they are not present, auto report assumes the options that correspond to blank entries (see individual entries for the meanings of the blank entries). Option specifications cannot be contained in a library member that is copied by a /COPY statement.

If a control specification (H in column 6) is not present either in the auto report source program or in a copied library member (see “Auto Report /COPY Statement Specifications” on page 633), auto report creates a control specification with blank entries.
The following columns on the auto report specifications are used in the same way as corresponding columns on other RPG specifications:

- Columns 1-2 (page)
- Columns 3-5 (line)
- Columns 75-80 (program identification).

**Note:** The U-specification options override those specified by the OUTFILE, OUTMBR, and some RPTOPT parameters when compiling a program with the CL command CRTS63RPT. The RPTOPT parameters *NOSRC, *SRC, *NOAST, *AST, *DATE, and *NODATE can be specified on a U-specification. If a U-specification exists, the parameter values in the columns will be used rather than the values otherwise specified for the command. A blank in column 27, for example, will override an RPTOPT(*NODATE) specification. To use the values specified in the CL command CRTS36RPT, remove the U-specification.

**Column 6 (Form Type)**
Enter a U in column 6 to identify this line as an auto report option specification.

**Column 7 (Source)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The created source program is not cataloged.</td>
</tr>
<tr>
<td>C</td>
<td>The created source program is cataloged in a library on disk.</td>
</tr>
</tbody>
</table>

Use column 7 to specify if the created source program is to be cataloged in a library. In either case, the created source program is written to a disk work file from which it is immediately compiled. Created source programs that are cataloged become library source members.

The created source program is not cataloged when terminal errors exist in the auto report specifications.

**Columns 8-24 (Source Member Reference)**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library, member</td>
<td>Identifies the library member cataloged. The library name can be up to eight characters long, beginning in column 8. Use a comma after the library name. The member name can also be up to eight characters long.</td>
</tr>
</tbody>
</table>

Make an entry in columns 8 through 24 to catalog the created source program in a library (C in column 7). The first character of the library name and of the member name must be alphabetic. The remaining characters can be alphabetic or numeric.

If you enter F1 or blanks for the library name, the library name defaults to #LIBRARY. If the member name is not specified or is specified incorrectly, an error results.

If the name used by auto report to catalog the created source program is the same as the name of an existing member in the library, the old member is replaced by the new member.
Columns 25-26
Columns 25 and 26 are not used. Leave them blank.

Column 27 (Date Suppress)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Date and page number are printed on the first *AUTO page heading line.</td>
</tr>
<tr>
<td>N</td>
<td>Date and page number are not printed on the first *AUTO page heading line.</td>
</tr>
</tbody>
</table>

To prevent the date and page number from printing on the first *AUTO heading line, enter N in column 27. When these fields are suppressed, the page title and any other fields specified can occupy the entire line. See “Auto Report Page-Heading Specifications” on page 640 for further information on the date and page numbers.

Column 28 (*Suppress)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Asterisks are printed for total output lines.</td>
</tr>
<tr>
<td>N</td>
<td>Asterisks are not printed for total output lines.</td>
</tr>
</tbody>
</table>

To prevent asterisks from printing beside created totals, enter N in column 28. See “Auto Report Output Specifications” on page 644 for rules used in printing asterisks.

Column 29
Column 29 is not used. Leave it blank.

Column 30 (List Options)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>The source program listing, headings, and diagnostics are printed.</td>
</tr>
<tr>
<td>B</td>
<td>The program listing is not printed; however, a source program is produced.</td>
</tr>
<tr>
<td>P</td>
<td>A partial program listing is printed that includes appropriate headings and diagnostics.</td>
</tr>
</tbody>
</table>

Column 30 provides options for printing a source listing when auto report creates RPG source specifications. If any terminal errors are found in auto report specifications, the listing is completed (provided a listing is printed).

The auto report source listing consists of the RPG specifications included in the input to auto report, RPG specifications created by auto report, and specifications copied from a library member.

Use the B entry to produce a source program for which you already have a listing.

Use the P entry to determine if minor changes to a previously tested program created any errors.
Auto Report /COPY Statement Specifications

The auto report copy function provides a way to include cataloged RPG source specifications in an RPG program. The source specifications that are included must reside as a library member on disk. 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 the library by the Source Entry Utility (SEU) and included in any program by the copy function.

Auto report specifications and any valid RPG specifications, including arrays and tables, can be copied in this manner. The auto report option specifications and other copy statements cannot be copied. See “Examples of Using Auto Report” on page 686 for an example of using the copy function.

The specifications included in an auto report program by the copy function are first placed in the program immediately following the /COPY statement. After all specifications are copied from the library members, the entire auto report program is sorted into the order required by the RPG compiler (see “Order of Created Specifications” on page 628). Note that the auto report compiler truncates any record that is longer than 96 characters.

To request the copy function, use the /COPY statement. This statement identifies the library and library member containing the RPG specifications included in the source program created by auto report. /COPY statements must follow the auto report option specifications, and they must precede source tables for translating files, tables for changing the collating sequence of characters, and compile-time array and table data.

The format of the /COPY statement is:

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Page and line number indicating the placement of the statement in the sequence of auto report source specifications.</td>
</tr>
<tr>
<td>6</td>
<td>This column can contain any entry except H or U, or can be blank.</td>
</tr>
<tr>
<td>7-11</td>
<td>Enter the characters /COPY.</td>
</tr>
<tr>
<td>12</td>
<td>Blank.</td>
</tr>
<tr>
<td>13-29</td>
<td>Identifies the library and member included. Specify the library name, which can be up to eight characters long, beginning in column 13. Use a comma to separate the library name from the member name, which can also be up to eight characters long. If you do not enter a library, or if you enter F1, the default is #LIBRARY.</td>
</tr>
<tr>
<td>30-49</td>
<td>Blank.</td>
</tr>
<tr>
<td>50-80</td>
<td>Enter any information or comments. The contents of these columns are not read by auto report.</td>
</tr>
</tbody>
</table>
Figure 234 shows an example of the /COPY statement.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC......................................*  
I*  
I* F1 is the library name.  
I* For F1 entry, the default library is #LIBRARY.  
I*  
I/COPY F1,SALETR  
I*  
I* SALETR is the name of member to be copied.  
I*  
I*  
I/COPY USERLIB,SALETR  
Note: It is convenient to code the /COPY statement on the input specifications if the input specifications are to be changed as they are copied.  

Figure 234. Example of the /COPY Auto Report Statement

Changing Copied Specifications

Statements can be included in the auto report specifications to change file description and input field specifications as they are copied from a library member. No other types of specifications can be changed. /COPY modifier statements from the source program that add, change, or delete entries on cataloged input field specifications are identified by an X in print position 6 of the auto report listing.

Changing File Description Specifications

To change a file description specification that is copied from a library member, enter the filename in columns 7 through 14 of a file description specification. Then make only those entries on the line that are to replace existing entries in the copied specification or that are 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 are to be copied from #LIBRARY. The original specification contains I in file type (column 15), defining SALES as an input file (see Figure 235 on page 635). To update the SALES file, change column 15 to a U by including in the auto report source program a modifier file description specification that contains the filename, SALES, and the new file type entry, U. As a result of the modifier file description specification, the file type on the copied file description specification is changed from I to U.
I/COPY Statement to Copy Specifications for SALES File from the System Library Member Named SALETR.

I/COPY F1,SALETR

Figure 235. Changing a Copied File Description Specification
To set an entry to blanks, enter an ampersand (&) in the first position of that entry on the modifier statement, and leave the remaining positions blank. For example, to remove the block length entry (columns 20 through 23) from the cataloged specification shown in Figure 235 on page 635, add an ampersand to the modifier statement in column 20, as shown in Figure 236, and leave columns 21 through 23 blank.

Modifier statements for file description specifications do not have to be in any particular order in the auto report source program, except that they cannot immediately follow the /COPY statement if input field specifications are also being changed.

Only one file description specification with a particular filename is allowed to come from the library entries, and a particular filename can be used only once on a modifier statement.

No changes 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 (specifications describing individual fields on the input record) can be changed. To change an input field specification copied from a library member, enter the field name in columns 53 through 58 of an input field modifier statement (I in column 6). Modifier statements for input field specifications must immediately follow the /COPY statement in the auto 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 I in position 6 is not considered the end of the input field modifier statements.)

The fields that can be modified are:

- Column 43 (packed/binary)
- Columns 44-51 (field location)
- Column 52 (decimal positions)
- Columns 59-60 (control levels)
- Columns 61-62 (matching or chaining fields)
- Columns 63-64 (field record relation)
- Columns 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 237 on page 638 shows examples of changing input specifications.
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, those statements that change existing input field specifications should come first; then those that are added as new input field specifications. This order 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 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.
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC........................................*  
ISALES AA 01
I..........................................................PFromTo++DField+L1M1FrPoNeEq...*  
I  1  7 ITEMNO
I  8  9 BRANCH
I 10 10 REGION
I 11 25 DESC
I 26 27 SOLDQY
I 28 34 SOLDVA 13
I 35 36 ONHAND
I 37 43 VALUE

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC........................................*  
I/COPY F1,SALETR
I..........................................................PFromTo++DField+L1M1FrPoNeEq...*  
I*
I* Add an entry to BRANCH field description.
I*
I
BRANCH 1
I*
I* Blank out minus field indicator
I* on SOLDVA description.
I*
I
SOLDVA & 2
I*
I* Add a new file description.
I 1 43 RECORD 3

1 Add an entry to BRANCH field description
2 Blank out minus field indicator on SOLDVA description
3 Add a new file description

Figure 237 (Part 1 of 2). Changing Copied Input Field Specifications
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.................................................

**SALES AA 01**
I.............................................................PFromTo++DField+L1MIrPoNeEq...* 
I 1 7 ITEMNO
I*
I* Added L1 indicator
I*
I 8 9 BRANCL1 1
I 10 10 REGION
I*
I* Blanks in place of minus field indicator
I* indicator
I*
I 11 25 DESC 2
I 26 27SOLDQY
I 28 342SOLDVA
I 35 360ONHAND
I 37 432VALUE
I*
I* Added field description
I*
I 1 43 RECORD 3

Resulting input specifications for SALES file showing:

1  Added L1 indicator
2  Blanks in place of minus field indicator
3  Added field description

*Figure 237 (Part 2 of 2). Changing Copied Input Field Specifications*
Auto Report Page-Heading Specifications

The *AUTO page-heading specifications provide an easy way to produce a page heading at the top of every page of a printed report (see Figure 238). Up to five *AUTO page-heading specifications can be used for a page heading. If both standard RPG heading lines and *AUTO page headings are specified in combination for a file, they are printed in the order specified by the output specifications. The *AUTO page headings can be specified for only one PRINTER file per program.

The heading line created by the first *AUTO page heading (H-*AUTO) specification contains a date and page number. The first heading line can also contain a title. (See “Field-Description Specifications” on page 642 for information on entering a title.)

The created date is printed in the leftmost columns in the format mm/dd/yy unless you change the format by using the date or inverted-print option (columns 19 through 21 of the control specification).

The created page number is printed in the rightmost columns and is preceded by the word PAGE. The page number field is four digits long and is zero suppressed. Auto report uses one of the unused PAGE fields (PAGE, PAGE1 through PAGE7) for page numbering. If all PAGE fields are used in the program, auto report does not number pages.

To suppress the date and page number on the first heading line, enter N in column 27 of the auto report option specifications.

```
*+1+2+3+4+5+6+7*
OName++++DFBASbSaN01N02N03Excptn.................................................................*
OSAMPLE H *AUTO
0................N01N02N03Field+YBEnd+PConstant/editword++++++++++++++++*
0  'SAMPLE REPORT'
```

Figure 238. *AUTO Specification and the Heading Line That Is Printed
Record-Description Specifications

Each H-*AUTO record description defines a separate heading line. The record-description entries allow you to specify spacing, skipping, and the conditions under which the line is printed.

Columns 7-14 (Filename)

Enter the name of the PRINTER file on which the heading is to be printed.

Column 15 (Type)

Enter H in column 15 on each record-description specification line that defines a page-heading line. The H and the entry *AUTO in columns 32 through 36 define this as an H-*AUTO heading specification (see Figure 238 on page 640). Up to five H-*AUTO specifications are allowed.

Column 16

Column 16 is not used. Leave it blank.

Columns 17-22 (Spacing and Skipping)

Enter spacing and skipping values in these columns according to the rules given under “Columns 17-22 (Spacing and Skipping)” on page 610 in Chapter 25, “Output Specifications.” If these columns do not contain spacing and skipping values, auto report skips to line 06 before the first line is printed and spaces 2 after the last H-*AUTO line is printed. If multiple H-*AUTO lines are used, auto report spaces 1 after each line except the last. For additional information on created spacing and skipping values, see “Report Format” on page 660.

Columns 23-31 (Output Indicators)

On the first H-*AUTO specification, either leave columns 23 through 31 blank or enter output indicators according to the rules given under “Columns 23-31 (Output Indicators)” on page 611 in Chapter 25, “Output Specifications.”

If these columns are blank, auto report prints the corresponding output line first-page (1P) time in the program cycle and when overflow occurs. Therefore, the heading is printed at the top of each page of the printed report. Indicators can be assigned to subsequent H-*AUTO specifications. If columns 23 through 31 are blank on any H-*AUTO specification after the first, that specification is assigned the same indicators as the first.

If an overflow indicator is specified on the file description specifications for the PRINTER file, that indicator conditions the created heading specifications. Otherwise, auto report defines an unused overflow indicator for the PRINTER file and conditions the line with that indicator.

AND and OR lines can be used with H-*AUTO output indicators if an output indicator is used with the first specification. Standard RPG rules for AND and OR lines apply.
Columns 32-37 (*AUTO)
Enter *AUTO in columns 32 through 36. This entry and an H in column 15 of the output specifications (see Figure 238 on page 640) indicate that this is an auto report heading line.

Columns 38-70
Columns 38 through 70 are not used on the record-description line. Leave them blank.

Field-Description Specifications
Each H-*AUTO record-description specification can be followed by one or more field-description specifications. The field-description specifications specify the title printed on the heading line and describe any other fields and literals printed on the line.

Columns 7-31
Columns 7 through 31 are not used on field-description specifications. Leave them blank. Output indicators in columns 23 through 31 cannot be used to condition a field on an H-*AUTO specification.

Columns 32-37 (Field Name)
Entry Explanation
Blank A constant (enclosed in apostrophes) must be entered in columns 45 through 70. The constant is printed on the heading line.
Field name Field defined in the program is printed on the heading line.
Table name A table item is printed on the heading line/
Indexed array name An array item is printed on the heading line.

Use columns 32 through 37 to enter a field name, a table name, or an indexed array name (defined elsewhere in the program) that is to print on the heading line. If a name is entered, an edit word, not a constant, can be entered in columns 45 through 70. A constant must be entered in columns 45 through 70 if columns 32 through 37 are blank.

If output indicators (columns 23 through 31) are left blank on the record-description specification, auto report conditions all fields and all array or table items included on the heading line with N1P in columns 23 through 25. Therefore, the field or the array or table item does not print on the first page. (If printed on the first page, the field might contain meaningless data because the first record is not read.) N1P is not created for the following RPG reserved words: PAGE, PAGE1 through PAGE7, UDATE, UDAY, UMONTH, UYEAR.

For information on formatting and centering *AUTO heading lines, see “Report Format” on page 660.
Column 38 (Edit Codes)
An edit code can be entered in column 38 if a numeric field, numeric array item, or numeric table element is named in columns 32 through 37. If an edit code is used, columns 45 through 70 must be blank unless asterisk fill or a floating currency symbol is specified. If column 38 is blank, no editing is done by auto report unless an edit word is used.

Column 39 (Blank After)
Enter B in column 39 to reset a numeric field to zeros after it is printed or to reset an alphameric field to blanks after it is printed on the heading line.

Columns 40-44
Columns 40 through 44 are not used with *AUTO heading specifications. Leave them blank.

For information on the positioning of fields and constants in the title line and on the centering of heading lines in relation to the body of the report, see “Report Format” on page 660.

Columns 45-70 (Constant or Edit Word)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Columns 32 through 37 contain the name of a field that either is not edited or is edited by an edit code.</td>
</tr>
<tr>
<td>Constant</td>
<td>Title or other constant (enclosed in apostrophes) that is to appear on the printed line.</td>
</tr>
<tr>
<td>Edit word</td>
<td>The edit pattern used to edit the numeric field named in columns 32 through 37 of the same field-description line.</td>
</tr>
</tbody>
</table>

Use columns 45 through 70 to specify the title and other information that is to appear on the output line and to edit numeric fields that are to appear on the line. Rules for specifying constants and edit words are identical to those given under “Columns 45-70 (Constant or Edit Word)” on page 619 in Chapter 25, “Output Specifications,” except that no end positions can be specified.

For information on the positioning of fields and constants in the title line and on the centering of heading lines in relation to the body of the report, see “Report Format” on page 660.
Auto Report Output Specifications

Detail reports (in which a line is printed for each record that is read) and group printed reports (in which only totals are printed) can be specified by the *AUTO output function alone or in combination with standard RPG specifications. The *AUTO output function creates totals and formats columns and column headings.

A single detail or total *AUTO record-description (D/T-*AUTO) specification and its associated field-description specifications can specify:

- Up to three lines of column headings to appear above a field
- Accumulation of several levels of totals, including a final total (known as total rolling)
- Creation by auto report of end positions for column headings and fields
- Creation by auto report of the K edit code for numeric fields
- Fields or constants printed next to created totals.

Four types of description specifications can be associated with the *AUTO record-description specification. The four types are distinguished by entries in column 39. The remaining entries on a field-description specification have different meanings, depending on the entry in column 39.

The valid entries in column 39 of the field-description specifications and their meanings are:

- Blank or B: Indicates that the associated field or constant appears on the detail line.
- A: Indicates that the associated numeric field is printed on the detail line and is accumulated. A total is printed for each control level defined in columns 59 and 60 of the input specifications for the program. A final total is also printed (when the LR indicator is on).
- C: Indicates that the associated constant is printed on the second or third line of column headings.
- 1, 2, 3, 4, 5, 6, 7, 8, 9, R: Indicates that the associated field or constant appears on the total line created for the respective control-level indicator (L1 through L9, LR).

See “Group Printing” on page 655 for the effect of these entries in a group-printed report.

See “Examples of Using Auto Report” on page 686 for examples of the four types of field descriptions.
Record-Description Specifications

An auto report record-description specification must contain the entry *AUTO in columns 32 through 36. *AUTO can appear only on a record-description specification. This entry indicates that the record description and the following field descriptions are redefined according to their use by auto report.

Columns 7-14 (Filename)

Enter the name of the PRINTER file on which the report is printed. This must be the same file named on H-*AUTO specifications, if any.

Column 15 (Type)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>The auto report specifications describe a report containing detail lines.</td>
</tr>
<tr>
<td>T</td>
<td>The auto report specifications describe a report containing total lines, but no detail lines (group-printed report).</td>
</tr>
</tbody>
</table>

Enter D in column 15 and *AUTO in columns 32 through 36 for auto report to create a report that contains detail lines. The field-description specifications associated with the D-*AUTO record description specify:

- Fields to appear on the detail line
- Column headings
- Total rolling
- Constants to appear on total lines.

See “Examples of Using Auto Report” on page 686 for examples of D-*AUTO specifications.

Enter T in column 15 and *AUTO in columns 32 through 36 for auto report to create a group-printed report (see “Group Printing” on page 655).

Only one detail or one total *AUTO (D/T-*AUTO) record description specification can be used in a program.

Column 16 (Fetch Overflow)

Enter F in column 16 to specify fetch overflow. See “Column 16 (Fetch Overflow or Release)” on page 609 in Chapter 25, “Output Specifications” for the rules on using fetch overflow.

When used with the *AUTO output function, fetch overflow applies only to the detail line. If group printing is specified (T in column 15), fetch overflow applies to the lowest-level total line printed.

Columns 17-22 (Spacing and Skipping)

Enter spacing and skipping values in columns 17 through 22 according to the standard RPG rules. Entries specified apply only to the detail line created by a D-*AUTO specification or to the first total line created by a T-*AUTO specification.

Leave columns 17 through 22 blank to single space after each detail line printed or, if group printing is specified, after the first total line printed. For information on spacing and skipping for created column heading and total lines, see “Report Format” on page 660.
Columns 23-31 (Output Indicators)
Enter any valid output indicators in columns 23 through 31 to condition the detail or group-print line created by this *AUTO specification. If these columns are left blank on a D-*AUTO specification, the created detail line is conditioned by N1P. Therefore, it is not printed at first-page (1P) time in the RPG program cycle. If these columns are left blank for a T-*AUTO specification, the first created total line is conditioned by the lowest control-level indicator defined in the program. (See “Group Printing” on page 655 for additional information about the use of this entry with a T-*AUTO specification.)

AND and OR can be used with *AUTO output indicators if an output indicator is specified on the first record-description specification. Standard RPG rules for AND and OR lines apply.

Indicators specified in columns 23 through 31 of the record-description specification (and its associated AND/OR lines) apply only to the detail line created by a D-*AUTO specification or to the group-print line (lowest-level total specification) created by a T-*AUTO specification.

If column headings are specified in the field-description specifications that follow this *AUTO record description, they are conditioned by one of the following:

- The same indicators that are specified for the first H-*AUTO specification.
- The first-page (1P) indicator in an OR relationship with the overflow indicator specified for the file on the file-description specifications. If no overflow indicator is specified, auto report defines an unused overflow indicator and uses it to condition the lines.

Restriction: If N1P is specified on a D-*AUTO record-description specification that is followed by field-description specifications for totaling fields (A in column 39), the calculations created for the totaling fields are also conditioned by N1P. This causes a terminal diagnostic in the RPG compiler.

Columns 32-37 (*AUTO)
To indicate that this is an auto report specification, enter *AUTO in columns 32 through 36 on the record-description line. Column 15 must contain D or T to indicate a detail or total *AUTO specification. Only one D/T-*AUTO specification can be used in a program.

Columns 38-70
Columns 38 through 70 are not used on a D/T-*AUTO record-description specification. Leave them blank.

Field Description (Blank or B in Column 39)
D-*AUTO and T-*AUTO field-description specifications containing a blank or B in column 39 describe:

- An alphameric field such as an item description
- A numeric field that is not totaled
- A constant
- A field with a literal used as a column heading (see Figure 239 on page 647).

A field named on the line (or a constant when no field is named) following a D-*AUTO record-description specification is printed only on the detail report line. If
the field (or constant when no field is named) on the line follows a T-*AUTO record
description, it appears only on the first total line created.

```
.. 1   .. 2   .. 3   .. 4   .. 5   .. 6   .. 7   *
OName+++DFBASbSaNO1NO2NO3Excpn.................................*
OSAMPLE  D       *AUTO
0.............NO1NO2NO3Field+YBEnd+PConstant/editword+++++++++
0
0
0
0
0

FIELD1   'COLUMN HEADING 1'
FIELD2
'LITERAL 3'
```

As a result of these specifications, FIELD1 prints on each detail line under the heading COLUMN
HEADING 1. FIELD2 and LITERAL 3 print on each detail line without a column heading.

*Figure 239. Auto Report Field-Description*

**Columns 7-22**
Columns 7 through 22 are not used on the field-description lines. Leave them blank.

**Columns 23-31 (Output Indicators)**
Enter any valid output indicators in columns 23 through 31, or leave them blank. If
these columns are left blank, the field (or constant when no field is named on the
line) is printed on each detail line conditioned by the indicators for that record.
When group printing is specified (T-*AUTO specification), the field (or constant
when no field is named on the line) is printed each time the lowest-level total line is
printed. If a column heading is specified in columns 45 through 70 to appear over
a field named in columns 32 through 37, the column heading is not affected by
output indicators entered in columns 23 through 31.

**Columns 32-37 (Field Name)**
Enter a field name, a data structure name, an indexed array name, a table name,
or blanks in columns 32 through 37. If columns 32 through 37 are blank, a con-
stant must be entered in columns 45 through 70 of the same field-description spec-
ification. If a field name, data structure name, indexed array name, or table name
is entered, the value of the field or item is printed on the detail line (or on the first
total line if group printing is specified).

**Column 38 (Edit Codes)**
Enter a valid edit code in column 38 if columns 32 through 37 contain the name of
a numeric field, a numeric array item, or a numeric table. This column must be
blank for alphameric fields, data structures, array items, table items, and literals. If
column 38 is left blank on a field-description line for a numeric field, an array item,
or a table item, the auto report program provides a K edit code. The K edit code
prints a numeric field or item with commas and a decimal point, such as 3,489.13.
It also causes zero suppression, does not print zero balances, and prints a minus
sign on the right of negative balances.
Column 39 (Blank After)

Entry  Explanation
Blank  Field is not reset to zeros or blanks after printing.
B    Numeric field is reset to zeros after it is printed. Alphameric field is reset to blanks after it is printed.

Enter B in column 39 to reset alphameric fields or data structures to blanks or to reset numeric fields to zeros after they are printed. Blank after cannot be used for constants. This entry applies only to the detail line (or to the first total line if group printing is specified).

Columns 40-43 (End Position in Output Record)

Either leave columns 40 through 43 blank, or enter the print position of the rightmost character of the field (or constant if no field is named in columns 32 through 37) printed. If this column is blank, auto report creates end positions for fields, constants, and column headings. See “Report Format” on page 660 for additional information and considerations.

Column 44

Column 44 is not used, because packed-decimal and binary data cannot be specified. Leave this column blank.

Columns 45-70 (Constant)

Enter a constant or blanks in columns 45 through 70 when column 39 contains a blank. Constants are enclosed in apostrophes according to the standard RPG rules for coding constants. If these columns are left blank, a field name, data structure name, indexed array name, or table name must be entered in columns 32 through 37. Column-heading continuation lines can follow this field-description line, but the first line of the printed column heading will be blank. See “Field Description (C in Column 39)” on page 652.

If a constant is entered in these columns along with a field name in columns 32 through 37, the constant is printed on the first column-heading line over the field value. When a column heading is used, the length used to space the column on the report is the greater of the longest column-heading length or the field length, adjusted for editing. See “Report Format” on page 660 for additional information on how columns and fields are centered and spaced by auto report.

If a constant is entered in columns 45 through 70 and field name (columns 32 through 37) is blank, the constant is printed each time the detail report line is printed. In group printing, the constant is printed each time the first created total line is printed.

Field Description (A in Column 39)

Enter A in column 39 of a field-description specification following a D/T-*AUTO specification to accumulate and print totals for the field named in columns 32 through 37 (see Figure 240 on page 649). The number of levels of totals that are printed is determined by the control-level entry (columns 59 and 60) on input specifications. A final total is also printed when the LR indicator is on. (This process is called total rolling.)
If group printing is specified and a control-level indicator higher than the lowest-defined control level is specified in columns 23 through 31 on the record-description specification, totals are created for the indicator entered, all higher defined indicators, and LR.

The total output record created by auto report if you entered A in column 39 of a field description specification is conditioned by the associated control-level indicator defined in the input specifications. One total output record is created for each control-level indicator defined in the program.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn........................................ *
OSAMPLE D    *AUTO
0................N01N02N03Field+YBEnd+PConstant/editword++++++++++...*
0*
0* The A in column 39 causes the AMOUNT field to be accumulated.
0* Totals are printed for each control level and a final total is
0* printed. A column heading is specified in columns 45 through 70.
0*
0          AMOUNT A     'COLUMN HEADING'
```

**Created Total Fields**

When A is specified in column 39 of a detail or total *AUTO field-description specification, auto report creates and names total fields used in accumulating the required levels of totals. Auto report creates the field names for the total fields based on the name in columns 32 through 37 of the A-type field description. Names are created in the following way:

- If the specified field name has fewer than six characters, one character is added to the name to create a name for the total field. The added character is 1 through 9 or R, corresponding to the total indicators L1 through L9 and LR, respectively. For example, if ITEM is the specified field name and all nine control levels are defined, the created field names are ITEM1, ITEM2, ... ITEM9, and ITEMR.

- If the specified field name has six characters, the last character is replaced by one of the characters, 1 through 9, or R. For example, if AMOUNT is the specified field name and all nine control levels are defined, the created field names are AMOUN1, AMOUN2, ... AMOUN9, and AMOUNR.

Total fields are created and named for all control-level indicators defined in the program and for LR. (For an exception to this rule, see Figure 244 on page 658.) For example, if L1 and L3 are assigned to control fields on the input specifications and the field QTY is specified, three total fields, QTY1, QTY3, and QTYR, are created and named by auto report. All total fields created for the same level, such as QTY1 and AMOUN1, are printed on the same total line, and that line is conditioned by the corresponding control-level indicator.

Created total fields are two digits longer than the original field. For example, if the field QTY is defined with a length of 3, QTY1, QTY3, and QTYR all have lengths of 5. The number of decimal positions remains the same in the created fields. If a
field previously defined in a program has the same name as a created field name, you can redefine the previous field, giving it whatever length and number of decimal positions you want. If you do this, the created field is assigned the previously defined length and number of decimal positions (if the previous field is numeric).

Considerations

You can specify created field names in RPG specifications that are included in the program. You must be aware, however, that the use of created fields in this way can interfere with auto report’s automatic accumulation of totals.

Field names ending in 1 through 9 or R should not be used in an auto report program that accumulates totals, because auto report creates total fields ending in those characters. This is especially important for six-character field names, because auto report forms total field names by replacing the last character with 1 through 9 or R. No field name can be used more than once with A in column 39. Also, if a five- or six-character field name is specified with A in column 39, a second five- or six-character field name in which the first five characters are identical cannot be specified with A in column 39. For example, if the following four field names are specified with A in column 39 in an auto report specification, all but the first are invalid:

FIELD
FIELDX Invalid because the first five characters duplicate the first five characters of the first field.
FIELDY Invalid for the same reason as for FIELDX.
FIELD Invalid because it duplicates the first field.

Columns 7-22

Columns 7 through 22 must remain blank on the field-description lines.

Columns 23-31 (Output Indicators)

Enter any valid output indicators in columns 23 through 31, or leave them blank. If these columns are blank, the field described is printed on each detail line. If indicators are entered in columns 23 through 31, the field is printed only when the conditions represented by those indicators are met. Leave these columns blank for group printing.

If a column heading is specified in columns 45 through 70 to appear over a field named in columns 32 through 37, the column heading is not affected by output indicators entered in these columns. Also, output indicators specified when column 39 contains A do not affect the creation of calculations for the field.

Output indicators specified on an A-type field-description specification following a D-*AUTO specification condition the calculations created for the field. If the A-type field description follows a T-*AUTO specification, however, a specified indicator does not condition calculations created for the field.

Columns 32-37 (Field Name)

When column 39 contains A, the name of a numeric field that is to be accumulated must be entered in columns 32 through 37. These columns cannot identify an array, array item, or table. The field named is printed on each detail line of the report. If group printing is specified, the total field for the lowest control-level indicator defined (L1, L2, ... L9, LR, in that order) is printed on the created total line. (For an exception to this rule, see Figure 240 on page 649.) Totaling for any par-
ticular field by means of an A entry in column 39 can be specified only once in each program.

To create calculation and output specifications that accumulate and print the various levels of totals required, auto report creates and names additional totaling fields. Names created for the fields are based on the field name specified in these positions according to a set of rules (see “Created Total Fields” on page 649).

**Column 38 (Edit Codes)**
Enter an edit code in column 38, or leave it blank. If this column is blank, auto report creates a K edit code for the field named in columns 32 through 37. The K edit code causes the field to be edited with commas and a decimal point, such as 1,234,567.89. The field is also zero suppressed. Zero balances are not printed; negative balances are printed with a minus sign on the right. The edit code specified, or the created K edit code, applies to all created total fields as well as to the field named in columns 32 through 37.

**Column 39**
Enter A in column 39 to indicate that totals are accumulated for the field named in columns 32 through 37 of this field description. A total is printed for every control-level indicator defined in the input specifications and for the LR indicator. When column 39 contains A, columns 32 through 37 must contain the name of a numeric field. Columns 45 through 70 can contain a constant used as the first line of a column heading. (See “Created Specifications” on page 665 for additional information.)

When the lowest control-level indicator used for a T-*AUTO specification is higher than the lowest control-level indicator defined in the input specifications, auto report creates only the total lines corresponding to the lowest control-level indicator used for the T-*AUTO specification, the higher defined control levels, and LR (see “Group Printing” on page 655).

**Resetting Total Fields to Zero:** When column 39 contains A, the auto report program creates a B (blank after) in column 39 of all the detail and total field-description specifications created from the field name specified. Therefore, the value in the specified field and in any created fields is reset to zero after the field value is printed. If group printing is specified, auto report creates a calculation to reset the specified field to zero on each cycle. This prevents the same value from being accumulated more than once. An unconditioned total-calculation operation (Z-ADD) sets the field value to zero. This calculation is the first total calculation in the created RPG source program.

**Asterisk Indication:** To indicate that a printed line is a created total line, asterisks are printed on the line to the right of the highest end position created from the D/T-*AUTO specification. One asterisk is printed to the right on the lowest-level total line created. One additional asterisk is printed on each higher level line, including the final total. For example, if L1 and L3 are defined control-level indicators in a program, one asterisk is printed to the right of the L1 line, two asterisks are printed on the L3 line, and three are printed on the LR line. As many as 10 asterisks are printed on the LR line if all nine control-level indicators are defined in the program.

To suppress the creation of asterisks on total lines, enter N in column 28 of the auto report option specifications.
Columns 40-43 (End Position in Output Record)
Enter the print position of the rightmost character of the field to be printed, or leave these positions blank. If this entry is blank, auto report creates end positions for fields and column headings. See “Report Format” on page 660 for additional information and considerations.

Column 44
Column 44 is not used with auto report, because packed-decimal and binary data cannot be used. Leave this column blank.

Columns 45-70 (Constant)
Either leave columns 45 through 70 blank, or enter a literal. Do not enter an edit word; editing is done by an edit code. If a literal is entered when column 39 contains A, the literal becomes the first line of the column heading over the accumulated field.

If these columns are left blank, the first line of the column heading is blank, but column-heading continuation lines can specify the second and third lines of the column heading. See “Field Description (C in Column 39).” Also see “Report Format” on page 660 for information on how column heading and fields are centered and spaced by auto report.

Field Description (C in Column 39)
Enter C in column 39 of the *AUTO field descriptions to specify a second and third column-heading line. At times you may want more information in a column heading than can be contained on one line. Auto report enables you to specify the second and third lines of column headings by simply specifying the literals to appear on those lines. No additional heading output lines need be coded; no end position need be calculated. The special field-description specification that allows you to do this is identified by C in column 39 (see Figure 241).

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaNO1N2N3Excptn............................................
OSAMPLE D  *AUTO
0..................N01N2N3Field+YBEnd+PConstant/editword+++++++++++++++*
0*
0* C in column 39 is used to specify second and third column-heading
0* lines. A maximum of three column-heading lines (two C-type field
0* descriptions) can be used.
0*
0           FIELD     'FIRST HEADING LINE'
0            C     'SECOND HEADING LINE'
0            C     'THIRD HEADING LINE'
```

Figure 241. Specifying Second and Third Column-Heading Lines
Columns 7-38
Columns 7 through 38 must be blank on a field description that has C in column 39.

Column 39
Enter C in column 39. One or two C-type specifications can follow a field-description specification that has A, B, or blank in column 39 and an entry in columns 32 through 37. The first C-type specification causes a second column-heading line to be created. The second C-type specification creates a third column-heading line (see Figure 241 on page 652).

Columns 40-44
Columns 40 through 44 must be blank on a C-type field-description specification.

Columns 45-70 (Constant)
Enter a constant, up to 24 positions long including blanks, enclosed in apostrophes. The constant becomes the second or third line of column headings, depending on if it is on the first or second C-type specification. If two or three column-heading lines are specified, the shorter literals are centered on the longest.

Field Description (1-9 or R in Column 39)
Enter a digit (1 through 9) or R in column 39 of a field description to specify a field or constant printed on a specific total line.

Auto report allows you to print other information on created total lines in addition to the created totals resulting from A-type field descriptions. The value entered in column 39 corresponds to the level of the total line on which the information is printed (the corresponding control level must be defined in columns 59 and 60 in the input specifications). For example, 3 in column 39 indicates that the information is printed on the L3 total line; R indicates that the information appears on the final total, or LR, line (see Figure 242 on page 654). Fields and constants specified in this way are printed to the left of the leftmost created total on the line. See "Report Format" on page 660 for exact placement.

This type of field description can print information such as DISTRICT TOTAL, GRAND TOTAL, or other literal information. It can also print a field and specify an edit word, floating currency symbol, or asterisk fill for the field.

If none of the *AUTO output fields is defined with A in column 39, then 1 through 9 or R cannot be used in column 39. In group printing, only specify numbers that are higher than the lowest control-level indicator used to condition the T-*AUTO specification. If the T-*AUTO specification is not conditioned by a control-level indicator, use only numbers that are higher than the lowest control level defined in columns 59 and 60 on the input specifications.
In this example, the literal “GRAND TOTAL AS OF” followed by the current data prints on the left of the created final total line, as shown below.

![Figure 242. Specifying a Literal](image.png)

**Columns 7-31**
Columns 7 through 31 must be blank on a field-description line with 1 through 9 or R in column 39.

**Columns 32-37 (Field Name)**
Enter the name of a field, an indexed array name, or a table name. The corresponding field or item value prints on the total line indicated by the entry in column 39. If columns 32 through 37 are blank, a constant must be entered in columns 45 through 70.

**Column 38 (Edit Code)**
Enter an edit code in column 38 to edit a numeric field named in columns 32 through 37, or leave column 38 blank. If column 38 is left blank, an edit word can be entered in columns 45 through 70. If column 38 is blank, no edit code is assumed by auto report.

**Column 39**
Enter a digit (1 through 9) or R. These entries correspond to the indicators L1, L2, . . . L9, and LR. The entry identifies a specific total line on which the field or literal described is to be printed. The entry in column 39 must correspond to a control level that is defined by the input specifications. In group printing, the entry in this column must be higher than the control level of the first total line created.
Columns 40-43 (End Position in Output Record)
Do not make an entry in columns 40 through 43 on field-description specifications with 1 through 9 or R in column 39. See “Report Format” on page 660 for additional information and considerations.

Column 44
Leave column 44 blank.

Columns 45-70 (Constant or Edit Word)
Leave columns 45 through 70 blank, or enter a constant or edit word. If field name (columns 32 through 37) on this specification line contains an entry, then columns 45 through 70 can contain any of the following:

- Blanks, if no editing is needed for the field or if the field is already edited by an edit code in column 38
- Edit word, if special editing is desired
- Floating currency symbol or asterisk-fill entry used with an edit code.

Columns 45 through 70 cannot contain a constant when field name contains an entry. However, when field name is blank, columns 45 through 70 must contain a constant.

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 either subtotals and a final total or only a final total.

Specifications
To specify group printing using auto report, enter T in column 15 and *AUTO in columns 32 through 36. A control-level indicator can be specified in columns 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 literals defined by field-description specifications that have a blank or B in column 39 and follow a T-*AUTO record description are printed on the lowest-level total line. Fields defined with A in column 39 are not printed on the total lines, but the total fields created by auto report are. Continued column headings (C in column 39) and total-indicated fields (1 through 9 or R in column 39) can also be specified by field descriptions following a T-*AUTO record description.

Output indicators can be entered in columns 23 through 31 of a field-description specification following a T-*AUTO record description if column 39 of the field-description specifications contains a blank or B. If output indicators are used in a field description that has A in column 39 following a T-*AUTO specification, those indicators are ignored by auto report. Output indicators cannot be used in a field description that contains C, 1 through 9, or R in column 39.
Examples
Figure 243 on page 657 shows the file description and input specifications for the group-printed reports shown in Figure 244 on page 658 and Figure 245 on page 659. BRANCH and REGION are defined as control fields.

Figure 244 on page 658 shows the calculation specifications, 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.

A DISK 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 output specifications in the same program with *AUTO specifications. The output record described is written on the DISK file, DISKSUM, when there is an L1 control break (BRANCH field changes). Because the T-*AUTO specification is conditioned by L2, auto report does not create fields for the L1 control level. Therefore, standard RPG 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 245 on page 659 shows a group-printed report similar to the one shown in Figure 244 on page 658. 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.
Figure 243. File Description and Input Specifications For Group-Printed Reports
AUTO REPORT OUTPUT SPECIFICATIONS

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C  01  SOLDQ1  ADD  SOLDQY  SOLDQ1  40
C  01  SOLDV1  ADD  SOLDVA  SOLDV1  92
C  01  VALUE1  ADD  VALUE  VALUE1  92

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn.............................................*
OPRINTER H  *AUTO
0..................N01N02N03Field+YBEnd+PConstant/editword+++++++++++*
0*
0* T in column 15 with *AUTO in columns 32 through 37 specifies a
0* group-printed report.
0*
0* Because L2 is entered under output indicators, total lines are
0* printed only for L2 and LR, although L1 is also a defined
0* control-level indicator.
0*
0 'SALES FOR ANY COMPANY'
0 ' BY REGION'
0 T   L2  *AUTO
0 REGION  'REGION'
0 SOLDQY A  'NUMBER OF SALES'
0 SOLDVA A  'VALUE'
0 VALUE A  'VALUE OF STOCK'
0 C  ' ON HAND'
0 R  'COMPANY TOTAL'
ODISKSUM T   L1
0 REGION  1
0 BRANCH  3
0 SOLDQ1 B  7
0 SOLDV1 B  16
0 VALUE1 B  25

Figure 244 (Part 1 of 2). Using *AUTO to Produce a Group-Printed Report Showing Region and Final Totals
In group printing, the lowest-level total lines printed (L2, in this case) are single-spaced, like detail lines.

<table>
<thead>
<tr>
<th>REGION</th>
<th>NUMBER OF SALES</th>
<th>VALUE</th>
<th>VALUE OF STOCK ON HAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23</td>
<td>71,000.00</td>
<td>19,000.00</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>70,000.00</td>
<td>29,000.00</td>
</tr>
<tr>
<td>COMPANY TOTAL</td>
<td>53</td>
<td>141,000.00</td>
<td>48,000.00</td>
</tr>
</tbody>
</table>

Figure 244 (Part 2 of 2). Using *AUTO to Produce a Group-Printed Report Showing Region and Final Totals

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn.........................................
OPRINTER H *AUTO
0.................N01N02N03Field+yBEnd+pConstant/editword+++++++++++...
0          'SALES FOR ANY COMPANY'
0          'BY BRANCH AND REGION'
0          *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          2  'REGION'
0          REGION 2
0          2  'TOTALS'
0          R  'COMPANY TOTALS'
```

Figure 245 (Part 1 of 2). Using *AUTO to Produce a Group-Printed Report Showing Branch, Region and Final Totals
When no control-level indicators are entered under output indicators, a total line is created for each defined control-level indicator (L1 and L2, in this case) and for LR.

Figure 245 (Part 2 of 2). Using *AUTO to Produce a Group-Printed Report Showing Branch, Region and Final Totals

Report Format

One of the advantages of auto report is that it frees you from the task of specifying the format of your report on the output specifications sheet. Auto report can completely format the report by spacing, skipping, centering lines, and calculating end positions for fields and constants.

Spacing and Skipping

You can specify spacing and skipping in an RPG source program, or you can leave it to auto report. Figure 246 on page 661 shows spacing and skipping created by auto report. For the specifications used to produce the report, see “Compiling the Auto Report Source Program” on page 628. If columns 17 through 22 are left blank on an H-*AUTO specification, auto report skips to line 06 before printing the first heading line, and it spaces two lines after the last heading line. If more than one heading line is specified, auto report spaces one line after the first heading line and after all succeeding heading lines except the last. To specify spacing and skipping, follow the standard RPG rules for spacing and skipping.

Column-heading lines are spaced like page headings. Auto report spaces one line after all column headings except the last. It spaces two lines 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 created. The entries do not apply to column headings or to total lines created by auto report from the D-*AUTO specification. Standard RPG rules for spacing and skipping must be followed. If spacing and skipping entries are not made, auto report spaces one line after printing the created detail line.
Two lines are spaced after all total lines produced by auto report from a D-AUTO specification. In addition, the lowest-level total line and the final total line are also created with one space before.

If spacing and skipping entries are made on a T-AUTO specification, the entries apply to the lowest-level total line created, but not to column headings or to higher level total lines. If spacing and skipping entries are not made, one line is spaced after the lowest-level total lines; two lines are spaced after all higher levels. One line is always spaced before the next-to-the-lowest-level total and before the final total (see Figure 245 on page 659 for an example).

<table>
<thead>
<tr>
<th>REGION</th>
<th>BRANCH</th>
<th>ITEM NUMBER</th>
<th>DESCRIPTION</th>
<th>SALES</th>
<th>AMOUNT</th>
<th>ON-HAND VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>5</td>
<td>25,000.00</td>
<td>2,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AG77055</td>
<td>PICK-UP</td>
<td>10</td>
<td>20,000.00</td>
<td>1,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP6545B</td>
<td>CAMPER</td>
<td>2</td>
<td>8,000.00</td>
<td>2,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>53,000.00</td>
<td>12,000.00</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>2</td>
<td>10,000.00</td>
<td>5,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AG77055</td>
<td>PICK-UP</td>
<td>4</td>
<td>8,000.00</td>
<td>2,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18,000.00</td>
<td>7,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71,000.00</td>
<td>19,000.00</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>AG6545B</td>
<td>CAMPER</td>
<td>10</td>
<td>40,000.00</td>
<td>20,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP6549P</td>
<td>1/4 TON TRUCK</td>
<td>20</td>
<td>30,000.00</td>
<td>9,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70,000.00</td>
<td>29,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70,000.00</td>
<td>29,000.00</td>
</tr>
</tbody>
</table>

**Figure 246. Report Showing the Format Created by Auto Report**
Placement of Headings and Fields

Auto report creates end positions for fields and constants and centers column headings, columns, and report lines (see Figure 246 on page 661 for an example). However, 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 created from the field description. (The specified end position may be changed slightly by auto report when the line is centered or when the column heading and field are positioned in relation to each other.) If the specified end position causes an overlay with a previous field or constant, auto report creates 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 column 27 of the auto report option specifications), 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 created 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 in relation 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 created 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. Auto 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 created by auto report. If you want to find the end positions that are created for page, date, and title information, see the listing of the created source program that is produced by the RPG compiler (see “Created Specifications” on page 665).

Catalog the created RPG source program in a library by specifying the C option in column 7 of the auto report option specifications, and change the end positions on the created source statements by using the Source Entry Utility (SEU).
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 constant in a column heading. However, if the field is longer than the longest constant in a column heading, the column heading is printed in the leftmost positions over an alphameric field and in the rightmost characters over a numeric field. When more than one column-heading line is specified, shorter column headings are always centered on the longest column heading (see Figure 246 on page 661).

Fields and constants appear from left to right on a line in the order in which they are specified by the output specifications. At least two blank spaces appear before each field on the line. However, no spaces are provided before a constant; you must incorporate blanks within constants if you want to provide additional spacing.

Total indication (fields and constants specified with 1 through 9 or R in column 39 of the output specifications) is placed to the left of the first total field (A in column 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 created from a D/T-*AUTO specification are longer than the record length specified for the PRINTER file, a second print line (overflow line) is created for each column-heading line, detail (or group-print) line, and total line. (Remember, a second print line is not created for *AUTO page-heading lines.) The excess information is placed in the rightmost positions on the overflow line in the order specified.

Figure 247 on page 664 shows the result of an overflow condition.

In the output specifications for the report shown in Figure 247 on page 664, no spacing or skipping is specified. If spacing and skipping are specified, however, auto report spaces the report as follows:

- Column heading lines and total lines are spaced as shown in Figure 247 on page 664.
- The space-before and skip-before entries specified are for the original detail (or group-print) line. Auto report creates one space after this line.
- The space-after and skip-after entries specified are for the overflow line. Auto report creates blanks for space-before and skip-before for the overflow line.
### Figure 247. Report Showing Overflow of D-*AUTO Print Lines

<table>
<thead>
<tr>
<th>REGION</th>
<th>ACCOUNT NUMBER</th>
<th>ACCOUNT NAME</th>
<th>INVOICE NUMBER</th>
<th>INVOICE DATE</th>
<th>DATE PAID</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>DISCOUNT</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11243</td>
<td>JONES HARDWARE</td>
<td>27541</td>
<td>7/11/80</td>
<td>7/21/80</td>
<td>23.75</td>
<td></td>
<td>.47</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11352</td>
<td>NU-STYLE CLOTHIERS</td>
<td>27987</td>
<td>7/14/80</td>
<td>7/26/80</td>
<td>23.28</td>
<td></td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11886</td>
<td>MIDI FASHIONS INC</td>
<td>15771</td>
<td>7/04/80</td>
<td>7/14/80</td>
<td>87.07</td>
<td></td>
<td>1.47</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12874</td>
<td>ULOOK INTERIORS</td>
<td>25622</td>
<td>7/09/80</td>
<td>7/23/80</td>
<td>67.95</td>
<td></td>
<td>1.47</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>18274</td>
<td>STREAMLINE PAPER INC</td>
<td>29703</td>
<td>7/21/80</td>
<td>7/30/80</td>
<td>274.03</td>
<td></td>
<td>2.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>170.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>560.02</td>
<td></td>
<td>4.99</td>
<td></td>
</tr>
<tr>
<td>REGION TOTALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>406.86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>23347</td>
<td>RITE-BEST PENS CO</td>
<td>20842</td>
<td>7/18/80</td>
<td>7/20/80</td>
<td>15.80</td>
<td></td>
<td>5.80</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25521</td>
<td>IMPORTS OF NM</td>
<td>29273</td>
<td>7/20/80</td>
<td>7/27/80</td>
<td>797.40</td>
<td></td>
<td>11.93</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>26723</td>
<td>ALRIGHT CLEANERS</td>
<td>19473</td>
<td>7/07/80</td>
<td>7/23/80</td>
<td>462.00</td>
<td></td>
<td>11.93</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>28622</td>
<td>NORTH CENTRAL SUPPLY</td>
<td>17816</td>
<td>7/05/80</td>
<td>7/22/80</td>
<td>75.97</td>
<td></td>
<td>11.93</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>29871</td>
<td>FERGUSON DEALERS</td>
<td>27229</td>
<td>7/10/80</td>
<td>7/22/80</td>
<td>61.91</td>
<td></td>
<td>11.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,413.08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGION TOTALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,195.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30755</td>
<td>FASTWAY AIRLINES</td>
<td>26158</td>
<td>7/06/80</td>
<td>7/19/80</td>
<td>742.72</td>
<td></td>
<td>16.85</td>
<td>1.90</td>
</tr>
<tr>
<td>3</td>
<td>31275</td>
<td>ENVIRONMENT CONCERNS</td>
<td>20451</td>
<td>7/06/80</td>
<td>7/30/80</td>
<td>29.43</td>
<td></td>
<td>14.43</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32457</td>
<td>B SOLE SILOS</td>
<td>27425</td>
<td>7/10/80</td>
<td>7/20/80</td>
<td>110.05</td>
<td></td>
<td>14.43</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>37945</td>
<td>HOFFTA BREAKS INC</td>
<td>18276</td>
<td>7/06/80</td>
<td>7/23/80</td>
<td>47.23</td>
<td></td>
<td>14.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>929.43</td>
<td></td>
<td>16.85</td>
<td>1.90</td>
</tr>
<tr>
<td>REGION TOTALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>898.15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>42622</td>
<td>EASTLAKE GRAVEL CO</td>
<td>16429</td>
<td>7/05/80</td>
<td>7/23/80</td>
<td>29.37</td>
<td></td>
<td>29.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,931.90</td>
<td></td>
<td>33.77</td>
<td></td>
</tr>
<tr>
<td>COMPANY TOTALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,529.73</td>
<td></td>
<td>368.40</td>
<td>1.90 **</td>
</tr>
</tbody>
</table>

* Auto report prints those columns that cannot be completely contained on the original line on overflow lines.

** Auto report prints those columns that cannot be completely contained on the original line on overflow lines.
Created Specifications

Auto report creates standard RPG specifications and combines them with RPG specifications included in the input to auto report and with specifications copied from library members. From them, it produces the final RPG source program. This section describes the created RPG specifications and the order of those specifications in the RPG source program.

Figure 248 shows auto report specifications for a sales report and Figure 249 on page 667 shows the resulting RPG source specifications that are created for the report. Numbers are inserted in the figures to identify the auto report functions and to show the specifications that are created by each function.

*.. 1 2 3 4 5 6 7 ..* FFilenoNameIPEAFBlenRlenLK1AI0vKlocEDevice+....Exit++....A....U.* F* F* PRINTER File Description F* FPRINTER 0 F 120 120 OA PRINTER *.. 1 2 3 4 5 6 7 ..* IFilenoNameSqNORiPos1NCCPos2NCCPos3NCC.................................................* I* I* Copy Function and Modifier Statements I* I* File description and input specifications for the SALES file are I* contained in the source member named SALETR in the #LIBRARY (F1 is I* specified as the library). I* I/COPY F1,SALETR I...........................................PFromTo+DField+L1M1FrPoNeEq...* I* I* Modifier statements follow the /COPY statement to add control-level I* indicators. I* I BRANCHL1 I REGIONL2

Figure 248 (Part 1 of 2). Auto Report Specifications for a Sales Transaction Report
CREATED SPECIFICATIONS

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn....................................................*

OPRINTER H   *AUTO
0...............N01N02N03Field+YBEnd+PConstant/editword+++++++++++...*
0*
0*                           AUTO Page-Headings Function
0*  
O                      'SALES REPORT '
0                      'FOR ANY CO.'
O                               AUTO Output Function
0*  
0*                      AUTO Output Function
0*  
0   L2      REGION  'REGION'
0   L1      BRANCH  'BRANCH'
0   ITEMNO  'ITEM'
0             C  'NUMBER'
0   DESC    'DESCRIPTION'
0   SOLDQV  'SALES'
0*  
0*                          Accumulated Fields
0*  
0   SOLDVA A  'AMOUNT'
0   ONHAND   'ON-HAND'
0   VALUE A  'VALUE'
0             R  'FINAL TOTALS'

Figure 248 (Part 2 of 2). Auto Report Specifications for a Sales Transaction Report
If you do not specify a control specification, auto report creates an all-blank control specification for you.

Figure 249. RPG Source Program Created from Auto Report Specifications
**CREATED SPECIFICATIONS**

**Created Calculation Specifications**

Calculation specifications are created to accumulate totals for fields named on *
"AUTO field description specifications that have an A in column 39 (see
Figure 250).

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...*
OName++++DFBASbSaN01N02N03Excptn..........................................................*
OPRINTER H  */AUTO
0...................N01N02N03Field+YBEnd+PConstant/editword++++++++++++++.*
0*
0* Calculations are created for fields with an A in column 39.
0*
0  'SALES REPORT '              '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'
```

Total calculations roll higher level totals.

Subroutine accumulates the lowest level totals
(L1, in this example).

**Note:** Placement of the created calculation specifications in the RPG source program is shown in
Figure 249 on page 667.

*Figure 250. Calculation Specifications Created from Auto Report Coding for Sales Transaction Report*
An RPG subroutine is created to accumulate the values from these fields into the lowest-level created total fields. The name of the subroutine is always A$$SUM. The subroutine specifications are conditioned differently, depending on if detail or group printing is specified:

- If detail printing is specified, as in Figure 250 on page 668, 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-calculation specifications are created 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 250 on page 668, total-calculation specifications to accumulate L2 and LR totals are followed by the subroutine to accumulate the lowest-level total, L1.

Created total fields are defined (given length and number of decimal positions) when the total field is the result field in a created calculation specification. In the input specifications, SOLDVA and VALUE are numeric fields defined with a length of seven positions with two decimal positions. Figure 250 on page 668 shows that the total fields created 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 creates total-calculation specifications to reset each of the accumulated fields (A in column 39) on the lowest-level total line to zero on each cycle. A Z-ADD calculation, conditioned by L0, is created for each accumulated field. These calculations are the first total-calculation specifications in the created RPG source program.

Created Output Specifications

Figure 251 on page 670 shows the output specifications created by auto report. To identify specifications supplied by auto report (column-heading specifications, total specifications, conditioning indicators, spacing and skipping values, end-position values, blank after) compare the listing with the auto report specifications.

Auto report creates specifications to reset accumulated fields to zero after they are printed. See “Field Description (A in Column 39)” on page 648 for a discussion of resetting fields to zero. In this example, blank-after is created for accumulated fields.
Figure 251. Output Specifications Created from Auto Report Coding for Sales Transaction Report

Programming Aids

The chart shown in Figure 252 on page 671 should be helpful in determining valid *AUTO output entries depending on the contents of column 39.

The following programming suggestions may be helpful in specific programming situations.
A one-column heading can be printed over two or more fields if automatic column spacing is taken into consideration. For example, suppose the heading DATE is to print over a month field and a day field as follows:

```
DATE
MON DAY
XX XX
XX XX
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>40-43</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
<tr>
<td>44</td>
<td>Blank</td>
<td>Blank or column heading</td>
</tr>
<tr>
<td>45-70</td>
<td>Blank</td>
<td>Literal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-22</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>32-37</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22-31</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>38</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>38</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>38</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
<tr>
<td>38</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9, R</td>
<td>Blank</td>
<td>Blank or indicators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Blank</td>
<td>Blank or end position</td>
</tr>
</tbody>
</table>

Figure 252. Valid *AUTO Entries on the Output Specifications, Depending on the Contents of Column 39
Code the output specifications as shown in Figure 253.

To print a constant on only the first detail line under a column heading, move the constant to a field in the calculation specifications and print that field as shown in Figure 253.

```
*.. 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'

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments+++++++*
C
   L1  MOVE 'CONSTANT'FLDA 8

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
0Name++++DFBASbSaN01N02N03Excptn...........................................
0
   D                     *AUTO
0..........................N01N02N03Field+YBEnd+PConstant/editword++++++++++.*
0
   FLDA B                 'COLUMN HEADING'
```

Figure 253. Printing a Constant on Only the First Line

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 not present in all record types, the correct total is not created unless additional coding is used. The specifications shown in Figure 254 on page 673 give incorrect results because the T-*AUTO specification creates an unconditioned ADD subroutine if a field is added. Therefore, QTY is added when indicator 10 is on and when indicator 11 is on. Figure 255 on page 673 shows a method of obtaining the correct results.
Figure 254. Incorrect *AUTO Specifications for More than One Record Type

Figure 255. Correct *AUTO Specifications for More than One Record Type

Figure 256 on page 674 shows the specifications for counting records. This method is especially useful when you want to print a detail list, to take totals by control level, or to prevent 1s from being listed down the page.
USING CRTS36RPT TO COMPILE AUTO REPORT

Using CRTS36RPT to Compile an Auto Report Source Program on an AS/400 System

Use the CL command CRTS36RPT (Create Auto Report Program) to call the auto report preprocessor for an auto report source program. The CRTS36RPT command creates the program that can be run with a public authority of *CHANGE; you may want to change this on your system to maintain greater security.

The auto report preprocessor does not diagnose all errors in the source program. It leaves those that the RPG source program compiler will pick up. If a program object cannot be successfully generated, however, because of errors in the auto report specifications, the auto report compilation stops. If auto report ends, it issues the escape message RPT9001. A CL program can anticipate this exception by using the CL command MONMSG to monitor for the message.
When an RPG source program is successfully generated, and provided that the *NOCOMPIL\* option was not specified in the CRTS36RPT command, auto report automatically calls the RPG compiler.

The parameter and options for both the auto report preprocessor and RPG II source program compiler are entered initially with the CRTS36RPT compiler command.

**Completing the First CRTS36RPT Display**

Enter CRTS36RPT from the OS/400 operating system to bring up the first display on the display. Press F10 to display additional parameters as shown in the following display:

![CRTS36RPT Display](Figure 257. First CRTS36RPT Display, Showing Valid Options)

The first display shows the first set of parameters for the CRTS36RPT command and the valid options for each parameter.

Press F11 to display the parameters and their keywords for the CRTS36RPT command as shown in Figure 258 on page 676.
Using CRTS36RPT to Compile Auto Report

Create S/36 RPG II Auto Report (CRTS36RPT)

Type choices, press Enter.

- Program . . . . . . . . . . . . PGM *CTLSPEC
- Library . . . . . . . . . . . . *CURLIB
- Source file . . . . . . . . . . SRCFILE *S36SRC
- Library . . . . . . . . . . . . *LIBL
- Source member . . . . . . . . SRCMBR +*LIBL
- Generation severity level . . . GENLVL 21
- Never ending program . . . . NEP +*NO
- Maximum MRT devices . . . . MRTMAX 0
- Text 'description'. . . . . . TEXT +SRCMBRTXT

Additional Parameters

- Source listing options . . . . OPTION + for more values
- Generation options . . . . . . GENOPT

More...

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

Figure 258. First CRTS36RPT Display, Showing Keywords

Press F11 again to return to the CRTS36RPT display showing valid options for each parameter.

Completing the Second CRTS36RPT Display

Press F10 from the first CRTS36RPT display showing valid options to display additional parameters. Then, press the Roll Up key (page down) to see the remaining additional parameters as shown in the following display:

Create S/36 RPG II Auto Report (CRTS36RPT)

Type choices, press Enter.

- Print file . . . . . . . . . . . . QSYSRPT *LIST, *NOLIST, *XREF...
- Library . . . . . . . . . . . . *LIBL Name, *LIBL, *CURLIB
- Library for ICF files . . . . *LIBL Name, *LIBL, *CURLIB
- Auto report options . . . . . + for more values
- Save file for auto report . . . *NONE Name, *NONE
- Auto report save file member . . *NONE Name, *NONE
- Replace program . . . . . . . +YES *YES, *NO
- Target release . . . . . . . . . +CURRENT +CURRENT, *PRV
- User profile . . . . . . . . . . +USER +USER, +OWNER
- Authority . . . . . . . . . . . +CHANGE +CHANGE, +ALL, +USE...
- Phase trace . . . . . . . . . . . +NO +NO, +YES
- Intermediate text dump . . . . . . +NONE

More...

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

Figure 259. Second CRTS36RPT Display, Showing Valid Options

Press F11 to display the parameters and their keywords for the CRTS36RPT command as shown in the following display:
Create S/36 RPG II Auto Report (CRTS36RPT)

Type choices, press Enter.

- Print file . . . . . . . . . . . PRTFILE
  + for more values ___________ .

- Library . . . . . . . . . . . +LIBL
  + for more values +CURLIB

- Library for ICF files . . . . . ICFLIB

- Auto report options . . . . . RPTOPT
  + for more values __________

- Save file for auto report . . . OUTFILE
  + for more values __________

- Library . . . . . . . . . . . +LIBL

- Auto report save file member . OUTMBR
  + for more values __________

- Replace program . . . . . . REPLACE
  + for more values __________

- Target release . . . . . . TGTRLS
  + for more values __________

- User profile . . . . . . USRPRF
  + for more values __________

- Authority . . . . . . . . . . . AUT
  + for more values __________

- Phase trace . . . . . . . . . . . PHSTRC
  + for more values __________

- Intermediate text dump . . . . . . ITDUMP
  + for more values __________

More...

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

Figure 260. Second CRTS36RPT Display, Showing Keywords

Press F11 again to return to the CRTS36RPT display showing valid options for each parameter.
Completing the Third CRTS36RPT Display

Press the Roll Up key (page down) from the first CRTS36RPT display to see the following display:

```
Create S/36 RPG II Auto Report (CRTS36RPT)

Type choices, press Enter.

Snap dump ............... *NONE

Codelist ............... *NONE

Fix decimal data ........ *YES +YES, *NO
```

This display shows the third set of parameters and the valid options for each parameter.

Figure 261. Third CRTS36RPT Display, Showing Valid Options
Press F11 to display the parameters and their keywords for the CRTS36RPT command as shown in the following display:

![CRTS36RPT Display](image)

Figure 262. Third CRTS36RPT Display, Showing Keywords

Press F11 again to return to the CRTS36RPT display showing valid options for each parameter.

The CRTS36RPT Command

Entering the CRTS36RPT command is similar to entering the CRTS36RPG command described in Chapter 3, “Compiling an RPG II Program.” All object names must consist of alphameric characters. The first character must always be alphabetic, and the length of the name cannot exceed eight characters.

The CRTS36RPT command recognizes all the parameters that the CRTS36RPG command does. Some of these parameters, however, are not used by auto report itself, but are passed through to the RPG compiler. These are the PGM, GENLVL, NEP, MRTMAX, TEXT, OPTION, GENOPT, ICFLIB, REPLACE, TGTRLS, USRPRF, AUT, PHSTRC, ITDUMP, SNPDUMP, CODELIST, and FIXDECDTA parameters. The PRTFILE parameter specifies a file that auto report itself uses, and then passes to the RPG compiler.

The CRTS36RPT command can be submitted in a batch stream, entered interactively at a work station, or in a CL program.

CRTS36RPT has three additional parameters of its own: RPTOPT, OUTFILE, and OUTMBR. A description of these parameters follows. Defaults are shown first and underlined.

**RPTOPT**

Specifies the options to use when the auto report program is compiled. Any or all of the following keyword options can be specified in any order. Separate the keywords with a blank space.

* **NOSOURCE** or **NOSRC**: Do not produce a listing of the source program.
*SOURCE or *SRC: Produce a listing of the source program consisting of the auto report source program and all compile-time errors.

Note: You will override this option if you create a U-specification.

*NOFLOW: Do not write a flow of the major routines that are run while the auto report source program is being compiled.

*FLOW: Write a flow of the major routines that are run while the auto report source program is being compiled.

*NOAST: Do not generate asterisks for total output lines.

*AST: Generate asterisks for total output lines.

Note: You will override this option if you create a U-specification.

*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.

Note: You will override this option if you create a U-specification.

*COMPILE: Call the RPG compiler after the auto report source program statements are processed and the complete RPG source program generated.

*NOCOMPILE: Do not call the RPG compiler after the auto report source program statements are processed and the complete RPG source program generated.

*NOSECLVL: Do not print second-level message text.

*SECLVL: Print second-level message text the on line following the first-level message text.

OUTFILE
Specifies the name of the file where the complete RPG source program is placed and the library in which the file is located. The file is also used as the source input file to the RPG compiler unless the RPTOPT parameter value *NOCOMPILE is specified.

File
*NONE: Create a file in QTEMP to pass the generated RPG source program to the RPG compiler.

file-name: Enter the name of the file used to contain the complete RPG source program.

Library
*LILB: The system searches the library list to find the library in which the source output file is located.
**CURLIB**: The current library will be used. 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.

**Note**: The OPTIONS specification (U specification) can override the values specified for this parameter, as described in “Auto Report Option Specifications (U)” on page 630.

**OUTMBR**

Specifies the name of the member of the file that will contain the output from auto 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 auto report.

**Note**: The OPTIONS specification (U specification) can override the values specified for this parameter, as described in “Auto Report Option Specifications (U)” on page 630.
Using AUTOC to Compile an Auto Report Source Program in the System/36 Environment

The AUTOC procedure compiles an RPG program that contains auto report specifications. The AUTOC procedure has two displays.

Calling AUTOC

To call AUTOC, type AUTOC and then press Enter or F4.

Using the First AUTOC Display

The first AUTOC display looks like this:

```
AUTOC PROCEDURE Optional-+
   Compiles an RPG II program that contains auto report specifications

Name of source program to be compiled . . . . . . . . . . . . . ________
Name of library containing source program . . . . . . . . . . . #LIBRARY
Call RPG II compiler . . . . . . . . . . . . . . . COMP,NOCOMP COMP__
Output option for compiler listings . . . . PRINT,NOPRINT,CRT PRINT__
Create cross-reference listing . . . . . . . . . . NOXREF,XREF NOXREF
Maximum number of requesting display stations . . . . . 0-99 0__
Never-ending program . . . . . . . . . . . . . . . . NONEP,NEP NONEP
Name of library to contain compiled program . . . . . . . . . ________

Press Enter for more options
F3=Exit     F6=Put on job queue     F17=Procedure complete
(C) COPYRIGHT IBM CORP. 1994
```

Figure 263. The First AUTOC Display

Respond to each prompt by entering the appropriate information.

*Name of source program to be compiled:* Enter the name of your source program.

*Name of library containing source program:* Enter the name of the library that contains the source member to be compiled. If no library name is specified, the name of the current library is assumed.

*Call RPG II compiler:* Enter COMP or NOCOMP.

- **COMP** The RPG compiler runs as part of the auto report function.
- **NOCOMP** The RPG compiler does not run as part of the auto report function.

If no option is specified, COMP is assumed.
Output option for compiler listings: Enter PRINT, NOPRINT, or CRT.

PRINT Print the listings created by the AUTOC procedure.

NOPRINT Do not print or display the listings.

CRT Displayed at the display station that requested the AUTOC procedure.

If no option is specified, PRINT is assumed.

Create cross-reference listing: Enter NOXREF or XREF.

NOXREF The AUTOC procedure does not create a cross-reference listing for the RPG program.

XREF Create a cross-reference listing. The cross-reference listing is created only if the program contains no terminal errors. The cross-reference listing is part of the compiler listing, so the printing or displaying of cross-reference listing depends on your response to the previous prompt.

If no option is entered, NOXREF is assumed.

Maximum number of requesting display stations: Enter the number (0 through 99) of display stations that can use a single copy of the program at the same time. If no number is specified, a value of 0 is assumed (the program is not a MRT program).

Never-ending program: Enter NONEP or NEP.

NONEP The program is not a never-ending program.

NEP The program is a never-ending program. A never-ending program is one that uses system resources (such as disk storage, display stations, or printers) that are not shared with other programs. Use this option if your program will be requested frequently.

If no option is entered, NONEP is assumed.

Name of library to contain compiled program: Enter the name of the library that is to contain the compiled program. If no library name is specified, the name of the source input library is assumed.

You can use the Help key from the first AUTOC Procedure display to see which function keys to use and for additional information on parameters.
# Using the Second AUTOC Display

The second AUTOC display looks like this:

<table>
<thead>
<tr>
<th>AUTOC PROCEDURE</th>
<th>Optional-+</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOC JHY,EXAMPLES,COMP,PRINT,NOXREF,0,NONEP,,</td>
<td>Ignored-+</td>
</tr>
</tbody>
</table>

- **Override print option in source**: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.
- **Override debug option in source**: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.
- **Override size-to-execute option in source**: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.
- **Halt on serious program error**: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.
- **Replace duplicate members**: Enter REPLACE or NOREPLAC.
  - **REPLACE**: If a program object is being created, and if a program object with the same name already exists in the output library, you want the newly compiled program to replace the existing program object.
  - **NOREPLAC**: If a program object is being created, and if a program object with the same name already exists in the output library, you want an error message to be displayed.

If no option is specified, REPLACE is assumed.

---

**Figure 264. The Second AUTOC Display**

Respond to each prompt by entering the appropriate information.
Create program that can be run: Enter LINK or NOLINK.

- **LINK** Creates a program that can be run.
- **NOLINK** Does not create a program that can be run.

Create program that must be link-edited: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

Name of subroutine input library: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

Generate CONSOLE file display formats: Enter GEN or NOGEN.

- **GEN** The procedure creates and compiles source specifications for 24-line, 1920-character display formats for a CONSOLE file. The specifications are created and compiled only if the program contains no terminal errors.
- **NOGEN** The procedure does not create or compile the source specifications for the display formats for a CONSOLE file.

If no option is specified, GEN is assumed. However, the procedure ignores this option if your program does not use a CONSOLE file.

Size of work files in blocks: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

Name of the data dictionary to be used: Enter the name of the library that contains the ICF file to be used during the compilation, if you are using user-defined communication formats. On a System/36, these formats were located in a data dictionary.

Create a program with memory-resident overlays: This prompt applies only to the System/36 processor. Any value that you enter will be syntax-checked but not processed.

You can use the Help key from the first AUTOC Procedure display to see which function keys to use and for additional information on parameters.
Examples of Using Auto Report

Examples 1 through 4 explain how auto report is used to create page headings and such output specifications as column headings, detail lines, and total lines.

Examples 5 and 6 illustrate the use of the auto report copy function to copy specifications from a library member and to change copied specifications for a particular job.

Example 7 prepares a cash receipts register.

Example 1–Sales Report using *AUTO

Problem

Produce the sales report shown below using the *AUTO page headings and *AUTO output functions of auto report.

Procedure


2. Code *AUTO page headings to produce a one-line page heading that includes date and page number.

3. Code *AUTO output to produce one-line column headings, detail report lines, and final totals.

Letters refer to fields on the following page.

<table>
<thead>
<tr>
<th>REGION</th>
<th>BRANCH</th>
<th>ITEM</th>
<th>DESCRIPTION</th>
<th>SALES</th>
<th>AMOUNT</th>
<th>ON-HAND</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>5</td>
<td>25,000.00</td>
<td>2</td>
<td>10,000.00</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>AG7705S</td>
<td>PICK-UP</td>
<td>10</td>
<td>20,000.00</td>
<td>1</td>
<td>2,000.00</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>AP8545B</td>
<td>CAMPER</td>
<td>2</td>
<td>8,000.00</td>
<td>2</td>
<td>5,000.00</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>2</td>
<td>10,000.00</td>
<td>1</td>
<td>5,000.00</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>AG7705S</td>
<td>PICK-UP</td>
<td>4</td>
<td>8,000.00</td>
<td>1</td>
<td>2,000.00</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>AG8545B</td>
<td>CAMPER</td>
<td>10</td>
<td>40,000.00</td>
<td>5</td>
<td>20,000.00</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>AP6549P</td>
<td>1/4 TON TRUCK</td>
<td>20</td>
<td>30,000.00</td>
<td>6</td>
<td>9,000.00</td>
</tr>
</tbody>
</table>

Figure 265. Sales Report Using *AUTO
EXAMPLES OF USING AUTO REPORT

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEIPEAFBlenRlenLK1AI0vKlocEDevice+....Exit++.......A....U+.*
FSALES   IP   F   473   43    DISK
FPRINTER  0   F   120   120   PRINTER
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMESqNORiPos1NCCPos2NCCPos3NCC...........................................
SALES  AA  01
I.....................................PFromTo++DField+L1M1FrPoNeEq...*
I                   1    7 ITEMNO
I                   8    9 BRANCH
I                  10    10 REGION
I                  11    25 DESC
I                 26 27 SOLDQY
I                 28 34 SOLDVA
I                35 36 ONHAND
I                37 43 VALUE

Field Name    Contents
A ITEMNO       Item number
B BRANCH       Number of the branch office where the item
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.

Figure 266. RPG File Description and Input Specifications for Example 1
Code *AUTO page-heading specifications.

Enter an H in column 15 and *AUTO in columns 32 to 36 to request an auto 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 number, enter an N in column 27 of the auto report option specifications.

The title information is centered by auto report; do not enter end positions in columns 40 through 43. Fields and table/array elements can also be used.

When space and skip entries (columns 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.)

When output indicators (columns 23 through 31) are left blank, auto report page headings are printed on each page (conditioned by 1P or overflow). If no overflow indicator is defined for the PRINTER file, auto report assigns an unused overflow indicator to the printer line.

Figure 267. Coding of *AUTO Page Headings
### Code AUTO output specifications to produce:

**A** Detail report line

**B** Column headings

**C** Final totals.

#### AUTO Report Format

**Figure 268. Coding of *AUTO Output Specifications**

Auto 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 column 39 are edited by the K edit code unless a different edit code is specified.
Example 2—Sales Report with Three Levels of Totals

**Problem**

Expand sales report from Example 1 to include three levels of totals:

1. Total for each branch
2. Total for each region
3. Final total.

**Procedure**

2. Add control-level indicators to the input fields BRANCH and REGION.

Note: The `AUTO` output function can also be used to produce a group-printed report. See Group Printing in this chapter for a discussion and examples of group printing.

![Figure 269. Input Specifications For a Sales Report With Three Levels of Totals](image-url)

Because two control-level indicators 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).

*Figure 269. Input Specifications For a Sales Report With Three Levels of Totals*
As in Example 1, an A in column 39 of the output specification causes SOLDVA and VALUE to be accumulated.

Auto 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.

Auto report prints asterisks (*) to the right of created total lines to aid in identifying them. If you want to suppress the asterisks, enter N in column 28 of the auto report option specifications.

Total fields are always two positions longer than the original fields and have the same number of decimal positions as the original fields.

**Figure 270. Output Specifications For Three Levels of Totals**

---

### Example of Using Auto Report

**Before**

<table>
<thead>
<tr>
<th>Type (H/D/T/E)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
<td>Not</td>
<td>Not</td>
</tr>
</tbody>
</table>

**Edit Codes**

<table>
<thead>
<tr>
<th>P/B/L/R</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

**Form Type**

<table>
<thead>
<tr>
<th>Line</th>
<th>Filename or Record Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2-TON TRUCK</td>
</tr>
<tr>
<td>1</td>
<td>PICK-UP</td>
</tr>
<tr>
<td>1</td>
<td>CAMPER</td>
</tr>
</tbody>
</table>

**Field Name or EXCPT Name**

<table>
<thead>
<tr>
<th>Space Skip</th>
<th>And</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Constant or Edit Word**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Commas Zero Balances to Print**

<table>
<thead>
<tr>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
</table>

**Position in Output Record**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>
Example 3—Sales Report with Group Heading

Expand the sales report from Examples 1 and 2 to contain:

A. Group indication for REGION and BRANCH fields
B. Second column-heading line
C. Literal (constant) on the final total line.

2. Code AUTO output with:
   A. Output indicator on field description specifications
   B. C in column 39 and a literal in columns 45 through 70
   C. R in column 39 and a literal in columns 45 through 70.
EXAMPLES OF USING AUTO REPORT

AG7701T
AG77055
AP6545B
2-TON TRUCK
PICK-UP
CAMPER

2 10 2 25,000.00
2 10 8,000.00
5 1 10,000.00
5 2 2,000.00

53,000.00
10,000.00
8,000.00
18,000.00
71,000.00
40,000.00
30,000.00
70,000.00
70,000.00
141,000.00

Figure 271 (Part 2 of 2). Sales Report with Group Heading

Chapter 26. The Auto Report Feature 693
Example 4—Sales Report with Cross-Column Totals

**Problem**

Expand the sales report from Examples 1, 2, and 3 to include a cross-totals column and:

- A new report page for each region
- Two heading lines on each page
- A field in a page-heading line
- Identification of branch and region totals.

**Procedure**

1. Code file description and input specifications as in Example 3; add an overflow indicator to the PRINTER file.
2. Code RPG calculation specifications for cross-total.
3. Code *AUTO specifications:
   - Output indicators on page-heading specifications
   - Two heading lines per page
   - Use of a field in an *AUTO page-heading specification
   - Fields and literals on L1 through L9 total lines (through 9 in column 39).

<table>
<thead>
<tr>
<th>11/11/80</th>
<th>SALES REPORT FOR ANY CO.</th>
<th>PAGE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BRANCH</strong></td>
<td><strong>ITEM NUMBER</strong></td>
<td><strong>DESCRIPTION</strong></td>
</tr>
<tr>
<td>17</td>
<td>AQ7701T</td>
<td>2-TON TRUCK</td>
</tr>
<tr>
<td></td>
<td>AQ7705S</td>
<td>PICK-UP</td>
</tr>
<tr>
<td></td>
<td>AP8545B</td>
<td>CAMPER</td>
</tr>
</tbody>
</table>
| **BRANCH 17 TOTALS** | | | | 63,000.00 | 12,000.00 | 75,000.00 | **
| 22 | AQ7701T | 2-TON TRUCK | 2 | 10,000.00 | 1 | 5,000.00 | 15,000.00 |
| | AQ7705S | 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 | **
| **REGION 1 TOTALS** | | | | 71,000.00 | 19,000.00 | 90,000.00 | **

<table>
<thead>
<tr>
<th>11/11/80</th>
<th>SALES REPORT FOR ANY CO.</th>
<th>PAGE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BRANCH</strong></td>
<td><strong>ITEM NUMBER</strong></td>
<td><strong>DESCRIPTION</strong></td>
</tr>
<tr>
<td>25</td>
<td>AG8545B</td>
<td>CAMPER</td>
</tr>
<tr>
<td></td>
<td>AG8549P</td>
<td>1/4 TON TRUCK</td>
</tr>
</tbody>
</table>
| **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 ( ) on this and the following pages to see the auto report coding to obtain this report.

*Figure 272 (Part 1 of 2). Sales Report With Cross-Column Totals*
RPG calculations can be among the input statements for auto report. This specification calculates a cross-total of the sales and on-hand values. (The placement of the calculation in relation to calculations created by auto report is described under Created Specifications.)

**Figure 272 (Part 2 of 2). Sales Report With Cross-Column Totals**
Example 5—Copy Function for Specifications of Sales Report

<table>
<thead>
<tr>
<th>Region</th>
<th>Branch</th>
<th>Item</th>
<th>Description</th>
<th>Sales</th>
<th>Amount On-Hand</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>5</td>
<td>25,000.00</td>
<td>2 10,000.00</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>AG7705S</td>
<td>PICK-UP</td>
<td>10</td>
<td>20,000.00</td>
<td>1 2,000.00</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>AP6545B</td>
<td>CAMPER</td>
<td>2</td>
<td>8,000.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>AG7701T</td>
<td>2-TON TRUCK</td>
<td>2</td>
<td>10,000.00</td>
<td>1 5,000.00</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>AG77055</td>
<td>PICK-UP</td>
<td>4</td>
<td>8,000.00</td>
<td>1 2,000.00</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>AG6545B</td>
<td>CAMPER</td>
<td>10</td>
<td>40,000.00</td>
<td>5 20,000.00</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>AP6540P</td>
<td>1/4 TON TRUCK</td>
<td>20</td>
<td>30,000.00</td>
<td>8 9,000.00</td>
</tr>
</tbody>
</table>

141,000.00  48,000.00

*Figure 273 (Part 1 of 3). Copy Function for Specifications of Sales Report*
EXAMPLES OF USING AUTO REPORT

F* The specifications for the SALES File could be replaced by
F* a single statement if they were put in a library member.
F* See Figure 26-44 (Part 3 of 3).
F*

FSALES IP F 473 43 DISK
FPRINTER O F 120 120 PRINTER

ISALES AA 01
I..............................PFromTo++DField+L1M1FrPoNeEq....
I 1 7 ITEMNO
I 8 9 BRANCH
I 10 10 REGION
I 11 25 DESC
I 26 27 SOLDQY
I 28 34 SOLDVA
I 35 36 ONHAND
I 37 43 VALUE

OName++++DFBASbSaN01N02N03Excptn....................................
OPRINTER H *AUTO

'SALES REPORT '
FOR ANY CO.'

REGION 'REGION'
BRANCH 'BRANCH'
ITEMNO 'ITEM'
DESC 'DESCRIPTION'
SOLDQY 'SALES'
SOLDVA A 'AMOUNT'
ONHAND 'ON-HAND'
VALUE A 'VALUE'

Figure 273 (Part 2 of 3). Copy Function for Specifications of Sales Report
EXAMPLES OF USING AUTO REPORT

*I* Column 6 of a /COPY statement must not contain a U or an H.
*I*
The source member is in #LIBRARY.
*I*
The /COPY statement copies file description and input specifications
*I* for the SALES file from the library member named SALETR.
*I*
I/COPY F1,SALETR
*I*

OPrinter H *AUTO

'SALES REPORT'

FOR ANY CO.'

'REGION'

'BRANCH'

'ITEM'

'DESRIPTION'

'SALES'

'AMOUNT'

'ON-HAND'

'VALUE'

Figure 273 (Part 3 of 3). Copy Function for Specifications of Sales Report
Example 6—Override Copied Input Specifications

Problem
Override copied input specifications to produce a report (below) that includes subtotals for branch and region.

Procedure

1. Put the specifications for the SALES file in a library member, 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.

Put input specifications for the SALES file in a source member.

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 library member.
EXAMPLES OF USING AUTO REPORT

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEIPEAFBlenRlenLK1AI0vKlocEDevice+......Exit++......A....U+.*
F*
F* 2 and 3 Code/COPY and modifier statements. As
F* a result of the modifier statements, three levels of totals are
F* accumulated for the SOLDVA and VALUE fields (L1, L2, and LR).
F*

FPRINTER 0  F 120 120  PRINTING

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAMEISqNORiPos1NCCPos2NCCPos3NCC.................................* 
I/COPY F1,SALETR
I............................................................PFromTo++DField+L1M1FrPoNeEq...*
I*
I* Entries on the modifier statements override the corresponding entries
I* in the copied specifications. The field names, BRANCH and REGION,
I* identify the input field specifications that are to be changed.
I*
I
I

BRANCHL1
I
BRANCHL2

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName++++DFBASbSaN01N02N03Excptn.................................* 
OPRINTER H *AUTO
0.....................N01N02N03Field+YBEnd+PConstant/editword+++++++++++.*
0
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'

Figure 274 (Part 2 of 2). Override Copied Input Specifications
File description or input specifications in the copied member are overridden as follows (see “Auto Report /COPY Statement Specifications” on page 633 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.
- New fields can be added to input specifications by new input field specifications added as modifier statements.
- Modifier statements do not change the specifications in the copied source member. The modification is only for the program into which the specifications are copied.

**Example 7—Cash Receipts Register**

Example 7 prepares a cash receipts register. The *AUTO page heading function and the *AUTO output function create the RPG output specifications for the report and the calculation specifications to accumulate final totals for several fields on the report. RPG calculation specifications that cannot be created by auto report are included in the auto report program to verify the discount taken by each customer and to calculate the balance due.

The file description specifications for the cash receipts register PRINTER file, CSHRECRG, and the file description and input specifications for the input file, CASHRC, are located in separate members in the library (see Figure 275 on page 702). These specifications are included in the program by the auto report copy function.

The input data for the file CASHRC in EXAUT2 is created by the program EXAUT1 (see Figure 276 on page 703). Figure 277 on page 704 shows the input data.
**EXAMPLES OF USING AUTO REPORT**

<table>
<thead>
<tr>
<th>File Type</th>
<th>Mode of Processing</th>
<th>Device</th>
<th>Symbolic Device</th>
<th>Name of Label Exit</th>
<th>Extent Exit for DAM</th>
<th>Number of Extents</th>
<th>Storage Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**continuation lines**

<table>
<thead>
<tr>
<th>K</th>
<th>Option</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Example 1:**

For the PRINTER file, CSHRECRG, the file description is in the library member EXAUT3.

**Example 2:**

For the DISK file, CASHRC, the file description and input specifications are in the library member EXAUT4.

---

**Figure 275. File Description and Input Specifications That are in the Library Members EXAUT3 and EXAUT4**

1. The file description for the PRINTER file, CSHRECRG, is in the library member named EXAUT3.
2. The file description and input specifications for the DISK file, CASHRC, are in the library member named EXAUT4.
### EXAMPLES OF USING AUTO REPORT

<table>
<thead>
<tr>
<th>RG 004</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CODE</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCASHRC</td>
<td>O 1020 68 SETON LR</td>
</tr>
<tr>
<td>OCASHRC</td>
<td>T LR</td>
</tr>
</tbody>
</table>

| 24 | '11243JONES HARDWARE ' |
| 48 | ' 27541123199 2375CASH ' |
| 68 | ' 47 47 2328123199' |
| 24 | '11352NU-STYLE CLOTHIERS ' |
| 48 | ' 27987123199 976CASH ' |
| 68 | ' 174 4000123199' |
| 24 | '11886MIDI FASHIONS INC ' |
| 48 | ' 15771123199 1072CASH ' |
| 68 | ' 214 214 10508123199' |
| 24 | '12874ALRIGHT CLEANERS ' |
| 48 | ' 25622123199 6795CASH ' |
| 68 | ' 136 6795123199' |
| 24 | '18274STREAMLINE PAPER IN' |
| 48 | ' C2970312399 27403CASH ' |
| 68 | ' 548 238 17055123199' |
| 24 | '23347RITY-BEST PENS CO ' |
| 48 | ' 20842123199 1580' |
| 68 | ' 31 1000123199' |
| 24 | '25521IMPORTS OF NM ' |
| 48 | ' 29273123199 19140 1' |
| 68 | ' 593 1193 58547123199' |
| 24 | '26723ALRIGHT CLEANERS ' |
| 48 | ' 19473123199 4600CASH ' |
| 68 | ' 924 46200123199' |
| 24 | '28622NORTH CENTRAL SUPPL' |
| 48 | 'Y17816123199 7597CASH ' |
| 68 | ' 152 7597123199' |
| 24 | '29871FERGUSON DEALERS ' |
| 48 | ' 27229123199 6191CASH ' |
| 68 | ' 124 6191123199' |
| 24 | '30755FASTWAY AIRLINES ' |
| 48 | ' 26158123199 7427CASH 1' |
| 68 | ' 495 1685 72587123199' |
| 24 | '31275ENVIRONMENT CONCERN' |
| 48 | 'S20451123199 2943' |
| 68 | ' 59 1500123199' |
| 24 | '32457SOLE SILOS ' |
| 48 | ' 27425123199 11005CASH ' |
| 68 | ' 220 11005123199' |
| 24 | '3794SHOFTTA BREAKS INC ' |
| 48 | ' 18276123199 4723CASH ' |
| 68 | ' 94 4723123199' |
| 24 | '42622EASTLAKE GRAVEL CO ' |
| 48 | ' 16429123199 2937CASH ' |
| 68 | ' 58 2937123199' |

**Figure 276. EXAUT1 Program**
EXAMPLES OF USING AUTO REPORT

DATA FOR SAMPLE PROGRAM

11243 JONES HARDWARE  27541123199  2375 CASH  47  47  2328123199
11352 NU-STYLE CLOTHIERS  27987123199  9707 CASH  174  4000123199
11886 MIDI FASHIONS INC  15771123199  10722 CASH  214  214  10508123199
12874 ULLOOK INTERIORS  25622123199  6795 CASH  136  6795123199
18274 STREAMLINE PAPER INC  29703123199  27403 CASH  548  238  17055123199
23347 RITE-BEST PENS CO  20842123199  1580  31  1000123199
25521 IMPORTS OF NM  29273123199  19140  1593  1193  58547123199
26723 ALRIGHT CLEANERS  19473123199  46200 CASH  924  46200123199
28622 NORTH CENTRAL SUPPLY  17816123199  7597 CASH  152  7597123199
29871 FERGUSON DEALERS  27229123199  6191 CASH  124  6191123199
30755 FASTWAY AIRLINES  26158123199  74272 CASH  1495  1685  72587123199
31275 ENVIRONMENT CONCERNS  20451123199  2943  59  1500123199
32457 B SOLE SILOS  27425123199  11005 CASH  220  11005123199
37945 HOFFTA BREAKS INC  18276123199  4723 CASH  94  4723123199
42622 EASTLAKE GRAVEL CO  16429123199  2937 CASH  58  2937123199

Figure 277. Input Data Created by EXAUT1 for Auto Report Sample Program EXAUT2
Control Specification
The RPG control specification shown in Figure 278 should be included in the auto report program because it is not present in the copied specifications (see Figure 275 on page 702). None of the control specification options are required in this program, so the specification need contain only an H in column 6 and the program identification, EXAUT2, in columns 75 through 80.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
H...OLExeD...CDYI....S................I...1.F.H...........T................*  
H      012  
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
IFilenameSqNORiPos1NCCPos2NCCPos3NCC..........................*  
I/COPY F1,EXAUT3  
I/COPY F1,EXAUT4  
I..........................................................PFromTo++DField+L1M1FrPoNeEq...*  
I 1 1 REGIONL1  
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++  
C DISTAK SUB DISCAL DIFF 42  
C DIFF COMP 1.00  10 10  
C AMTOWD SUB DISTAK NETOWD 62  
C NETOWD SUB AMTPD BAL 62

Figure 278 (Part 1 of 2). RPG and Auto Report Specifications to Produce the Cash Receipts Register
EXAMPLES OF USING AUTO REPORT

```
*.. 1 ...+.. 2 ...+.. 3 ...+.. 4 ...+.. 5 ...+.. 6 ...+.. 7 ..*
OName+++DFBASbSaN01N02N03Excptr.................................*
OCSHRECRGH *AUTO
0................N01N02N03Field+YBEnd+PConstant/editword+++++++++++*
0 Your Auto Report Specifications to Produce the Cash Receipts Register
0
0       D  01  *AUTO
0       REGION  'REGION'
0       ACCTNO  'ACCOUNT'
0       C  'NUMBER'
0       ACCTNM  'ACCOUNT NAME'
0       INVNO 3  'INVOICE'
0       C  'NUMBER'
0       INVDATY  'INVOICE'
0       C  'DATE'
0       DATPD Y  'DATE PAID'
0       AMTOWDJA  'AMOUNT'
0       C  'OWED'
0       DISTAK A  'DISCOUNT'
0       C  'TAKEN'
0       AMTPD A  'AMOUNT'
0       C  'PAID'
0       BAL A  'BALANCE'
0       C  'DUE'
0       10  DIFF A  'EXCESS'
0       C  'DISCOUNT'
0       1  'REGION TOTALS'
0       R  'COMPANY TOTALS'
Figure 278 (Part 2 of 2). RPG and Auto Report Specifications to Produce the Cash Receipts Register
```
/COPY Statements
The /COPY statements shown in Figure 278 on page 705 copy the file description and input specifications for the job from the source members in #LIBRARY. The first statement copies the file description specifications for the PRINTER file from the library member named EXAUT3. The second statement copies the file description and input specifications for the DISK file named CASHRC from the library member named EXAUT4. A modifier statement adds an input field definition for the REGION field. As a result of these /COPY statements, the file description and input specifications shown in Figure 275 on page 702 are included in the RPG source program created by auto report.

Calculation Specifications
The calculation specifications shown in Figure 278 on page 705 are included in the auto report program to do special operations that cannot be created by auto report. First, the discount allowed for each customer is subtracted from the discount taken by each customer. Indicator 10 turns on if the difference is $1.00 or more. The remaining calculations subtract the discount taken and the amount paid from the amount owed.

The order in which these calculations are placed in relation to the calculations created by auto report is shown in the auto report listing of the created RPG source program (see Figure 279 on page 708).
EXAMPLES OF USING AUTO REPORT

Figure 279 (Part 1 of 3). Auto Report Sample Program EXAUT2
EXAMPLES OF USING AUTO REPORT

Figure 279 (Part 2 of 3). Auto Report Sample Program EXAUT2
## Examples of Using Auto Report

The following code snippet demonstrates how to use an Auto Report in a System/36-Compatible RPG II environment.

```rpg
0034 0380EO  PAGE 2  131
0036 0390EO  127 'PAGE'
0037 0400EO0C SHREC RH 1  1P
0038 0410EO  OR OA
0039 0420EO  6 'REGION'
0040 0430EO  15 'ACCOUNT'
0041 0440EO  29 'ACCOUNT NAME'
0042 0450EO  46 'INVOICE'
0043 0460EO  56 'INVOICE'
0044 0470EO  67 'DATE PAID'
0045 0480EO  80 'AMOUNT'
0046 0490EO  92 'DISCOUNT'
0047 0500EO  105 'AMOUNT'
0048 0510EO  118 'BALANCE'
0049 0520EO  128 'EXCESS'
0050 0530EO0C SHREC RH 2  1P
0051 0540EO  OR OA
0052 0550EO  14 'NUMBER'
0053 0560EO  45 'NUMBER'
0054 0570EO  54 'DATE'
0055 0580EO  79 'OWED'
0056 0590EO  90 'TAKEN'
0057 0600EO  104 'PAID'
0058 0610EO  116 'DUE'
0059 0620EO  129 'DISCOUNT'
0060 0630EO0C SHREC GT 1  01
0061 0640EO  REGION  3
0062 0650EO  ACCTNO  14
0063 0660EO  ACCTNM  37
0064 0670EO  INVNO  45
0065 0680EO  INVDATE  56
0066 0690EO  DATPD Y  66
0067 0700EO  AMTOW0JB  80
0068 0710EO  DISTAKKB  92
0069 0720EO  AMTD PK KB  105
0070 0730EO  BAL KB  118
0071 0740EO  10 DIFF KB  129
0072 0750EO0C SHREC GT 12 L1
0073 0760EO  AMTOWLJB  80
0074 0770EO  DISTAKKB  76
0075 0780EO  AMTPD1KB  105
0076 0790EO  BAL1 KB  118
0077 0800EO  DIFF1 KB  129
0078 0810EO  67 'REGION TOTALS'
0079 0820EO  130 '*'
0080 0830EO0C SHREC GT 12 LR
0081 0840EO  AMTOWRJB  80
0082 0850EO  DISTARKB  92
0083 0860EO  AMTPORKB  105
0084 0870EO  BALR KB  118
0085 0880EO  DIFFR KB  129
0086 0890EO  67 'COMPANY TOTALS'
0087 0900EO  131 '**'
```

Figure 279 (Part 3 of 3). Auto Report Sample Program EXAUT2

---

710 System/36-Compatible RPG II User’s Guide and Reference
**Examples of Using Auto Report**

*Auto Specifications*

The coding for the *Auto page heading and the *Auto output functions is shown in Figure 278 on page 705. Notice that the Y edit code is used for the date fields (lines 10 and 12). Auto report creates a K edit code for numeric fields when an edit code is not specified. No edit code is created for numeric fields when they are described with a digit (1 through 9) or R in column 39. The edit code 3 is specified for the INVNO field to suppress the printing of the comma edit character.

DIFF is printed on the detail line only if it is $1.00 or more. Remember, output indicator 10 conditions only the printing of the field on the detail line; it does not affect the printing of the created field on the total line.

The J edit code allows zero balances to print for the AMTOWD field.

Totals are accumulated and printed by auto report for five fields as indicated by A entries in column 39. Because an L1 control level is defined in the input field specifications for REGION, which is added to the input specifications for CASHRC (see Figure 278 on page 705), regional and final totals are accumulated for each field that has A in column 39. The total lines are identified by the literals shown in lines 23 and 24 of the *Auto specifications (see Figure 278 on page 705).

Figure 280 shows the output data produced by EXAUT2.

---

**Figure 280. Output from Auto Report Sample Program EXAUT2**

<table>
<thead>
<tr>
<th>REGION</th>
<th>ACCOUNT NUMBER</th>
<th>ACCOUNT NAME</th>
<th>INVOICE NUMBER</th>
<th>INVOICE DATE</th>
<th>PAID DATE</th>
<th>AMOUNT OWED</th>
<th>DISCOUNT</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11243</td>
<td>JONES HARDWARE</td>
<td>27541</td>
<td>12/31/99</td>
<td>12/31/99</td>
<td>23.75</td>
<td>.47</td>
<td>23.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11352</td>
<td>NU-STYLE CLOTHIERS</td>
<td>27867</td>
<td>12/31/99</td>
<td>12/31/99</td>
<td>97.07</td>
<td>40.00</td>
<td>57.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11886</td>
<td>MIDI FASHIONS INC</td>
<td>15771</td>
<td>12/31/99</td>
<td>12/31/99</td>
<td>107.22</td>
<td>2.14</td>
<td>105.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12874</td>
<td>ULOOK INTERIORS</td>
<td>25622</td>
<td>12/31/99</td>
<td>12/31/99</td>
<td>67.95</td>
<td>67.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>18274</td>
<td>STREAMLINE PAPER INC</td>
<td>29703</td>
<td>12/31/99</td>
<td>12/31/99</td>
<td>274.03</td>
<td>2.38</td>
<td>170.55</td>
<td>101.10</td>
<td></td>
</tr>
</tbody>
</table>

**REGION TOTALS**

<table>
<thead>
<tr>
<th>REGION</th>
<th>AMOUNT OWED</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15.80</td>
<td>10.00</td>
<td>5.80</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>191.40</td>
<td>585.47</td>
<td>406.00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>462.00</td>
<td>462.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>61.91</td>
<td>61.91</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**REGION TOTALS**

<table>
<thead>
<tr>
<th>REGION</th>
<th>AMOUNT OWED</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>742.72</td>
<td>725.87</td>
<td>1.90</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>29.43</td>
<td>30.00</td>
<td>1.43</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>110.05</td>
<td>110.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>47.23</td>
<td>47.23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**REGION TOTALS**

<table>
<thead>
<tr>
<th>REGION</th>
<th>AMOUNT OWED</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>29.37</td>
<td>29.37</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**REGION TOTALS**

<table>
<thead>
<tr>
<th>REGION</th>
<th>AMOUNT OWED</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COMPANY TOTALS**

<table>
<thead>
<tr>
<th>AMOUNT OWED</th>
<th>AMOUNT PAID</th>
<th>BALANCE DUE</th>
<th>EXCESS DISCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,335.90</td>
<td>2,529.73</td>
<td>227.60</td>
<td>1.90 **</td>
</tr>
</tbody>
</table>
Chapter 27. Operation Codes

The RPG language allows you to do many different types of operations on your data. Special codes, which are entered on the calculation specifications, indicate the operations processed. Usually these codes are just abbreviations of the names of the operations.

Many operation codes can be placed into categories. The first part of this chapter includes general information about these categories. The rest of the chapter describes each operation code in alphabetical order and shows one or more examples for most of the operations. Table 20 is a summary of the specifications for each operation code.

Table 20 (Page 1 of 3). Summary of Operation Code Specifications

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Conditioning Indicators(^2) Positions 9-17</th>
<th>Factor 1</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators Positions 54-55</th>
<th>Resulting Indicators Positions 56-57</th>
<th>Resulting Indicators Positions 58-59</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACQ</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td>O: ER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD(^3)</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
<tr>
<td>BEGSR</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BITOF</td>
<td>O</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>O</td>
<td></td>
<td>R</td>
<td>O</td>
<td></td>
<td>O: ER</td>
<td>O</td>
</tr>
<tr>
<td>CASxx</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O: HI</td>
<td>O: LO</td>
<td>O: EQ</td>
</tr>
<tr>
<td>CHAIN</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O</td>
<td></td>
<td>O: NR</td>
<td></td>
</tr>
<tr>
<td>COMP</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td>O: HI(^4)</td>
<td>O: LO(^4)</td>
</tr>
<tr>
<td>DEBUG</td>
<td>O</td>
<td></td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFN</td>
<td></td>
<td>R</td>
<td>O</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV(^3)</td>
<td>O</td>
<td></td>
<td>O</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>O</td>
<td></td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUxx(^3)</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOWxx(^3)</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>O(^5)</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDSR</td>
<td></td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXCPT</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, but not in subroutines. On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.
2 Conditioning indicators are valid only for executable operation codes.
3 Half adjust (position 53) can be specified for this operation.
4 At least one resulting indicator must be specified in positions 54 through 59.
5 The END operation code can have a conditioning indicator if it is part of a DO, DOUxx, or DOWxx group, but not if it is part of a CASxx or IFxx group.

Fields without entries must be blank.

+ = Plus          \(=\) = Equal  \(=\) = Required
- = Minus         \(=\) = Error     \(=\) = Zero
BOF = Beginning of file  NR = No record found  ZB = Zero or blank
EOF = End of file   O = Optional
<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Conditioning Indicators&lt;sup&gt;2&lt;/sup&gt; Positions 9-17</th>
<th>Factor 1</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators Positions 54-55</th>
<th>Resulting Indicators Positions 56-57</th>
<th>Resulting Indicators Positions 58-59</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXSR</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORCE</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREE</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td>O: ER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF&lt;sup&gt;xx&lt;/sup&gt;</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>O</td>
<td>O</td>
<td></td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: ZB</td>
</tr>
<tr>
<td>LOKUP (Array)</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O</td>
<td>O: HI&lt;sup&gt;4&lt;/sup&gt;</td>
<td>O: LO&lt;sup&gt;4&lt;/sup&gt;</td>
<td>O: EQ&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td>LOKUP (Table)</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O</td>
<td>O: HI&lt;sup&gt;4&lt;/sup&gt;</td>
<td>O: LO&lt;sup&gt;4&lt;/sup&gt;</td>
<td>O: EQ&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td>MHHZO</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MHLZO</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLHZO</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLLZO</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVEA</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVEL</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT&lt;sup&gt;3&lt;/sup&gt;</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
<tr>
<td>MVR</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
<tr>
<td>NEXT</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARM</td>
<td>O</td>
<td>O</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>O</td>
<td>O</td>
<td>O&lt;sup&gt;5&lt;/sup&gt;</td>
<td>O&lt;sup&gt;5&lt;/sup&gt;</td>
<td>O: ER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>O</td>
<td></td>
<td></td>
<td>O</td>
<td>O: ER</td>
<td>R: EOF</td>
<td></td>
</tr>
<tr>
<td>READE</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td>O</td>
<td>R: EOF</td>
<td></td>
</tr>
<tr>
<td>READP</td>
<td>O</td>
<td></td>
<td></td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R: BOF</td>
</tr>
<tr>
<td>REL</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETRN</td>
<td>O</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLALBL</td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

<sup>1</sup> Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, but not in subroutines. On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.

<sup>2</sup> Conditioning indicators are valid only for executable operation codes.

<sup>3</sup> Half adjust (position 53) can be specified for this operation.

<sup>4</sup> At least one resulting indicator must be specified in positions 54 through 59.

<sup>5</sup> The END operation code can have a conditioning indicator if it is part of a DO, DOU<sup>xx</sup>, or DOW<sup>xx</sup> group, but not if it is part of a CAS<sup>xx</sup> or IF<sup>xx</sup> group.

Fields without entries must be blank.

+ = Plus
- = Minus
BOF = Beginning of file
EOF = End of file
EQ = Equal
ER = Error
NR = No record found
Z = Zero
ZB = Zero or blank
R = Required
O = Optional
<table>
<thead>
<tr>
<th>Operation Code&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Conditioning Indicators&lt;sup&gt;2&lt;/sup&gt; Positions 9-17</th>
<th>Factor 1</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators Positions 54-55</th>
<th>Resulting Indicators Positions 56-57</th>
<th>Resulting Indicators Positions 58-59</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETLL</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETOF</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETON</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHTDN</td>
<td>O</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORTA</td>
<td>O</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB&lt;sup&gt;3&lt;/sup&gt;</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
<tr>
<td>TAG</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TESTB</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TESTZ</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XFOOT&lt;sup&gt;3&lt;/sup&gt;</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z-ADD&lt;sup&gt;3&lt;/sup&gt;</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
<tr>
<td>Z-SUB&lt;sup&gt;3&lt;/sup&gt;</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>O: +</td>
<td>O: -</td>
<td>O: Z</td>
</tr>
</tbody>
</table>

<sup>1</sup> Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, but not in subroutines. On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.<br>
<sup>2</sup> Conditioning indicators are valid only for executable operation codes.<br>
<sup>3</sup> Half adjust (position 53) can be specified for this operation.<br>
<sup>4</sup> At least one resulting indicator must be specified in positions 54 through 59.<br>
<sup>5</sup> The END operation code can have a conditioning indicator if it is part of a DO, DOUxx, or DOWxx group, but not if it is part of a CASxx or IFxx group.<br>

Fields without entries must be blank.

| +   = Plus   | EQ = Equal | R = Required |
| -   = Minus | ER = Error | Z = Zero    |
| BOF = Beginning of file | NR = No record found | ZB = Zero or blank |
| EOF = End of file   | O = Optional |            |
MOVE OPERATIONS

Arithmetic Operations

Arithmetic operations (ADD, SUB, MULT, DIV, Z-ADD, and Z-SUB) can be processed only on numeric fields or numeric literals. The result field must also be numeric. Decimal alignment is done for all arithmetic operations. Even though truncation can occur, the position of the decimal point in the result field is not affected. For arithmetic operations in which all three fields are used:

- Factor 1, factor 2, and the result field can be three different fields.
- Factor 1, factor 2, and the result field can all be the same field.
- Factor 1 and factor 2 can be the same field but different from the result field.
- Either factor 1 or factor 2 can be the same as the result field.

The length of any field specified in an arithmetic operation cannot exceed 15 characters. If the result exceeds 15 characters, characters are dropped from either or both ends depending on the location of the decimal point. The results of all operations are signed (+ or -). Any data placed in the result field replaces the data that was there before.

Move Operations

Move operations (MOVE, MOVEA, and MOVEL) move all or part of factor 2 to the result field. Factor 2 remains unchanged. Factor 1 must be blank, and no resulting indicators can be specified in columns 54 through 59.

The MOVE and MOVEL operations can be used to change numeric fields to alphameric fields and alphameric fields to numeric fields. To change a numeric field to an alphameric field, enter the name of the numeric field in factor 2 and specify an alphameric result field. To change an alphameric field to a numeric field, enter the name of the alphameric field in factor 2 and specify a numeric result field.

When an alphameric field is moved into a numeric result field, the digit portion of each character is converted to its corresponding numeric character and then moved to the result field. Blanks are transferred as zeros. For the MOVE operation, the zone portion of the rightmost alphameric character is converted to its corresponding sign and is moved to the rightmost position of the numeric field where it becomes the sign of the field. For the MOVEL operation, the zone portion of the rightmost character of factor 2 is converted and used as the sign of the result field whether or not the rightmost character is included in the move operation.

When move operations are specified to move data into numeric fields, the decimal positions specified for the factor 2 field are ignored. For example, if the data 1.00 is moved into a numeric field with one decimal position, the result is 10.0.

The MOVEA operation can be used to move several contiguous array elements to a single field, a single field to several contiguous array elements, or contiguous elements of one array to contiguous elements of another array. All arrays and fields used in a MOVEA operation can be alphameric or numeric.
Move Zone Operations

The move zone operations (MHHZO, MHLZO, MLHZO, and MLLZO) move only the zone portion of a character. A minus (-) sign in a move zone operation does not yield a negative character in the result field, because a minus sign is represented by a hexadecimal 60 internally and a D zone is required for a negative character. Characters J through R have D zones and can be used to obtain a negative value (J = hexadecimal D1, ..., R = hexadecimal D9).

Note: Whenever the word high is used in a move zone operation, the field involved must be alphameric; whenever low is used, the field involved can be either alphameric or numeric.

Compare and Testing Operations

The compare and testing operations test fields for certain conditions. These operations are COMP and TESTZ. Another group of compare and testing operations consists of IFxx, DO, DOUxx, DOWxx, and CASxx (structured programming operations). The following rules of comparing fields apply to all compare and testing operations:

- If numeric fields are compared, fields of unequal length are aligned at the implied decimal point. The shorter field is filled with zeros to the left or right of the decimal point to make the fields of equal length. The maximum field length for numeric fields compared is 15 digits.

- If alphameric character fields are compared, fields of unequal length are aligned at their leftmost character. The shorter field is filled with blanks to equal the length of the longer field for comparison. The maximum field length for alphameric fields compared is 256 characters.

- All numeric comparisons are algebraic. A positive value (+) is always greater than a negative (-) value.

- Blanks within numeric fields are assumed to be zeros.

- Numeric fields are converted to packed-decimal format, if necessary, before they are compared.

- If an alternate collating sequence (position 26 of the control specification) has been specified for the comparison of character fields, the fields are translated into the alternate sequence before comparison.

- An alphameric field cannot be compared to a numeric field.

- An array name and a data structure name cannot be specified in a compare operation, but an array element, a table element, and a data structure subfield can.

With the COMP, CASxx and TESTZ operations, the resulting indicators assigned in positions 54 through 59 are set according to the results of the operation. With the CASxx operation, the branch to a subroutine specified in the result field occurs if the condition specified in the xx portion of the operation is met.

No fields are changed by compare and testing operations.
Structured Programming Operations

The structured programming operations are:

- **DO (Do)**
- **DOWxx (Do While)**
- **DOUxx (Do Until)**
- **IFxx (If/Then Do)**
- **CASxx (Case)**
- **END (End)**
- **ELSE (Else Do)**

The **DO** operation allows an operation or a series of operations to be processed a fixed number of times. You indicate how many times the operation(s) will be processed by specifying the starting value in factor 1, incrementing each time by the value in factor 2 of the associated END operation or by 1 if factor 2 on the END operation is not specified, until the index value (result field) exceeds the limit value (factor 2).

The **DOWxx** and **DOUxx** operations allow the processing of an operation or a series of operations one or more times based on the results of comparing factor 1 with factor 2.

The **IFxx/ELSE** operations allow the processing of an operation or a series of operations based on the results of comparing factor 1 with factor 2.

The **CASxx** operation allows conditional branching to a subroutine based on the results of comparing factor 1 with factor 2.

The **xx** portion of the **IFxx**, **DOUxx**, **DOWxx**, and **CASxx** operations 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 the **CASxx** operation only, if the **xx** portion of it contains blanks.

Conditioning indicators can be specified.

Factor 1 and factor 2 can contain a character literal, a numeric literal, an array element, a table name, a data structure subfield, a field name, or blanks (blanks are valid only for **CASbb**). If factor 1 and factor 2 are not blanks, both must be character data or both must be numeric. Only numeric literals, field names, array elements, table names, or data structure subfields with zero decimal positions can be specified in factor 1 and factor 2 of the **DO** operation.

The rules for comparing factor 1 and factor 2 on the **IFxx**, **DOUxx**, **DOWxx**, and **CASxx** operation codes are the same as those given under “Compare and Testing Operations” on page 717 in this chapter. The same rules apply to comparing a result field (index value) and factor 2 (limit value) on the **DO** operation.
The group of operations that begins with a DO, DOUxx, DOWxx, or IFxx operation and ends with an END operation, is called a do group. Each do group must end with an END operation, which either ends the do group or allows the do group (except the IF group) to continue running.

If a do group contains another complete do group, together they form a nested do group. The following is an example of nested do groups, three levels deep:

```
 DOU/DOW
 │ ┌── DOU/DOW
 │ │ └── END
 │ ┌── IF
 │ │ ┌── DOU/DOW
 │ │ └── END
 │ │ ┌── IF
 │ │ └── END
 │ └── END
 └── END
```

Remember the following when specifying do groups:

- Do groups can be nested to a maximum depth of 100 levels.
- Each do group must contain both a DO, DOUxx, DOWxx, or IFxx operation and an associated END operation.
- A do group must be contained in either detail, total, LR, or subroutine calculations; it cannot be split between the different calculation times.
- If you branch into a do group from outside the do group, the loop controls will not have been initialized. This may yield undesirable results.

A CAS group can contain only CASxx operations. An END operation must follow the last CASxx operation to denote the end of the CAS group.

After the subroutine is processed, the program continues at the next operation following the END operation for the CAS group, unless the subroutine is the INFSR and factor 2 on the ENDSR operation is not blank, in which case the return point for the subroutine is specified in factor 2 of the ENDSR operation.

**Bit Operations**

The bit operations (BITON, BITOF, and TESTB) set and test individual bits. Use the individual bits as switches in a program in order to save storage for binary-type switches.

When you use the BITON, BITOF, and TESTB operations, any field named in factor 2 or result field must be a one-position alphameric field. A field is considered alphameric if there are no entries in the decimal positions column of the input or calculation specifications. The field specified as factor 2 or as the result field can be an array element if each element in the array is a one-position alphameric element.

The initial value of an alphameric field is blank (hexadecimal 40).
SETON and SETOF Operations

The operation codes SETON and SETOF turn indicators on or off. Any indicator turned on or off is specified in columns 54 through 59. The headings for these columns (plus or high, minus or low, zero or equal) have no meaning in these operations. When setting indicators, consider:

- The following indicators cannot be turned on by the SETON operation: first-page (1P), matching-record (MR), level-zero (L0), function-key (KA through KN, KP through KY).
- The following indicators cannot be turned off by the SETOF operation: first-page (1P), matching-record (MR), level-zero (L0), and last-record (LR).
- If the last-record (LR) indicator is turned on by a SETON operation that is conditioned by a control-level indicator (columns 7 and 8 of the calculation specifications), processing stops after all total output operations are finished. If it is turned on by a SETON operation at detail time (not conditioned by a control-level indicator in columns 7 and 8), processing stops after the next total output operation is completed.
- If the halt indicators (H1 through H9) are turned on and not turned off before the detail output operations are complete, the system stops. The operator can continue processing by responding to the halt for every halt indicator that is on.
- Turning control-level indicators (L1 through L9) on or off does not automatically turn any lower control-level indicators on or off.
- Control-level indicators (L1 through L9) and the record-identifying indicators always turn off after the next detail output operations are completed regardless of the previous SETON or SETOF operation.
- Whenever a new record is read, record-identifying indicators (01 through 99) and field indicators turn on or off to reflect conditions on the new record. The setting from any previous SETON or SETOF operation does not apply then.
- If an indicator with the two-character entries 01 through 99 is turned on and is not changed in other calculations, it remains on until it is turned off by another calculation specification.

Branching within RPG

Operations are normally processed in the order in which they appear on the calculation specifications. There may be times, however, when the operations should be processed in a different order, such as when:

- Several operations should be skipped when certain conditions occur.
- Certain operations should be done for several, but not all, record types.
- Several operations should be repeated.

See “Conditional Branching” on page 14 and “Repeating an Operation” on page 16 in “Structured Programming in RPG II” for the details.
Subroutine Operations

The operation codes BEGSR, ENDSR, EXSR, and CASxx are used only for subroutines. In an RPG program, a subroutine is a group of calculation specifications that can be processed several times in one program cycle. A subroutine must be coded after all other calculation operations for a program. Subroutine specifications must be identified by SR or blanks in columns 7 and 8 on the calculation specifications. Therefore, individual operations within a subroutine cannot be conditioned by control-level indicators in columns 7 and 8. Within a subroutine, SR or blanks in columns 7 and 8 can be intermixed.

Calling External Subroutines or Programs

The EXIT/RLABL operation allows an RPG program to call an external program, in particular the IBM-supplied subroutines SUBR20, SUBR21, SUBR23, and SUBR95. There is a special interface between the calling program and the called program using EXIT/RLABL. See “RLABL (RPG Label)” on page 813 for a description of this interface.

The CALL/PARM operation also allows an RPG program to call other programs.

The PLIST and PARM operations are used in a called RPG program to receive data from the calling program. The program can be called through either the EXIT/RLABL or CALL/PARM operations.

WORKSTN Operations

The operation codes ACQ and REL are used only with the WORKSTN file. For these operations, factor 1 specifies either the name of a two-character field that contains the device identification or a two-character alphameric literal that is the device identification. Factor 2 specifies the name of the WORKSTN file for which the operation is requested. Columns 56 and 57 on the calculation specifications can contain a resulting indicator that turns on if an exception or error occurs.

Note: For WORKSTN files, a device can be either a display station or an ICF session.

Programmed Control of Input and Output

The normal program cycle can be changed to allow input and output operations during calculations. (See “RPG Program Cycle” on page 6, for a description of the program cycle.) The following operations provide this capability:

- EXCPT (Exception Output)
- READ (Read)
- READE (Read Equal Key)
- READP (Read Previous Record)
- FORCE (Force)
- NEXT (Next)
- CHAIN (Chain)
- SETLL (Set Lower Limits).
ACQ (ACQUIRE)

OPERATION CODES

The rest of this chapter discusses individual operation codes in alphabetical order.

ACQ (Acquire)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-8</td>
<td>Optional</td>
<td>Required</td>
<td>ACQ</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>9-17</td>
<td>Optional</td>
<td>Required</td>
<td>Blank</td>
<td>Blank</td>
<td>Opt: ER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54-55</td>
<td>56-57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>58-59</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The ACQ operation acquires the device specified in factor 1 for the program. Factor 2 must contain the name of the WORKSTN file.

If the device is available, ACQ attaches it to the program. If it is not available or is already attached to the program, an error occurs. If an indicator is specified in columns 56 and 57, the indicator turns on.

If no indicator is specified but the program contains the INFSR (WORKSTN exception/error-processing) subroutine, the INFSR subroutine automatically receives control when an exception or error occurs.

If no indicator is specified and the program does not contain the INFSR subroutine, the program halts when an exception or error occurs. No input or output operation occurs when the ACQ operation is processed. For more information about the ACQ operation, see Chapter 7, “Using a WORKSTN File.”
Factor 2 is added to factor 1. The sum is placed in the result field. Factor 1 and factor 2 are not changed by the operation. If factor 1 is not present, factor 2 is added to the result field, and the sum is placed in the result field.
The BEGSR operation serves as the beginning point of a subroutine. Factor 1 must contain the name of the subroutine. The control-level entry (columns 7 and 8) can be SR or blank. Columns 9 through 17 must not contain any conditioning indicators.

The subroutine name can be from one to six characters long. It must begin with an alphabetic character in column 18. The remaining characters can be any combination of alphabetic or numeric characters. However, special characters are not allowed, and blanks cannot appear between characters in the name. Every subroutine must have a different name. This name cannot be used as the label of a TAG or ENDSR operation.
BITOF (Set Bit Off)

The BITOF operation sets off (sets to 0) bits identified in factor 2 in the field named as the result field. Factor 2 is always a source of bits for the result field. The result field is the field in which the bits are set off.

Factor 2 can contain:

- Bit numbers 0-7: From one to eight bits can be set off per operation. The bits set off are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in column 33. For example, to set off bits 0, 2, and 5, enter '025' in factor 2.

- Field name: The name of a one-position alphameric field, array element, or table element can be specified in factor 2. In this case, the bits that are on in the field, array element, or table element are set off in the result field; bits that are off are not affected.

See Figure 281 on page 726 for a summary of BITOF operations.

The operation code BITOF must appear in columns 28 through 32. Conditioning indicators can be used in columns 7 through 17. However, factor 1, decimal positions, half-adjust, and the resulting-indicator columns must be blank.
The following BITOF operation sets bit 5 off in the field named BITSW.
The field is defined in the same line with a field length of 1.

```
C
BITOF'5'       BITSW 1
```

The following operation sets bits 1, 2, 4, and 6 off in the field named BITSW. The one-position field has been previously defined.

```
C
BITOF'1246'    BITSW
```

The following operation uses a one-position alphameric field as a source of bits. Any bits that are on in the field named ALPHA cause corresponding bits to be set off in the field named BITSW. If bits 5 and 7 are on in the field named ALPHA, the BITOF operation sets bits 5 and 7 off in the field named BITSW.

```
C
BITOFALPHA     BITSW
```

The following operations use a one-position alphameric array element either as a source of bits or as a result field, or both. In the first operation, any bits that are on in the field named ALPHA cause corresponding bits to be set off in the array element ARR,NX.

```
C
BITOFALPHA     ARR,NX
```

```
C
BITOF'137'     ARR,NX
```

```
C
BITOFARR,NX    ARE,12
```

* Figure 281 (Part 1 of 2). Summary of BITOF Operations
BITOF (SET BIT OFF)

C*
C* BITS is a one-position field containing hex F0 (numeric zero).
C* To change hex F0 to hex 40 (blank), set bits 0, 2, and 3 off:
C*  
C  BITOF'023' BITS
C*
C* To create a hex 1C (dup character) in the one-position field
C* ASTRSK, set all bits off, then set on bits 3, 4, and 5.
C*
C  BITOF'01234567'ASTRSK
C  BITON'345' ASTRSK

Figure 281 (Part 2 of 2). Summary of BITOF Operations
The BITON operation sets on (sets to 1) the bits identified in factor 2 in the field named as the result field. Factor 2 is always a source of bits for the result field. The result field is the field in which the bits are set on.

Factor 2 can contain:

- Bit numbers 0-7: From one to eight bits can be set on per operation. The bits set on are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in column 33. For example, to set on bits 0, 2, and 5, enter '025' in factor 2.

- Field name: The name of a one-position alphabetic field, array element, or table element can be specified in factor 2. In this case, the bits that are on in the field, array element, or table element are set on in the result field; bits that are off are not affected.

See Figure 282 on page 729 for a summary of BITON operations.

The operation code BITON must appear in columns 28 through 32. Conditioning indicators can be used in columns 7 through 17. However, factor 1, decimal positions, half-adjust, and the resulting-indicator columns must be blank.
The following BITON operation sets bit 4 on in the field named BITS. The field is defined in the same line with a field length of 1.

```
C
  BITON'4' BITS 1
C*
```

The following operation sets bits 0, 3, 5, and 7 on in the field named BITS. This one-position field has been previously defined.

```
C
  BITON'0357' BITS
C*
```

The following operation uses a one-position alphabetic field as a source of bits. Any bits that are on in the field named ALPHA cause corresponding bits to be set on in the field named BITS. If bits 5 and 7 are on in the field named ALPHA, the BITON operation sets bits 5 and 7 on in the field named BITS.

```
C
  BITONALPHA BITS
C*
```

The following operations use a one-position alphabetic array element either as a source of bits or as a result field, or both. In the first operation, any bits that are on in the array element ARR,NX cause corresponding bits to be set on in the array element ARE,12.

```
C
  BITONARR,NX ARE,12
C*
```

```
C
  BITON'0246' ARR,NX
C*
```

```
C
  BITONALPHA ARR,NX
C*
```

Figure 282. Summary of BITON Operations
### CALL (Call a Program)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>7-8</th>
<th>9-17</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Required</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>CALL</td>
<td>Required</td>
<td>Optional</td>
<td>Blank</td>
<td>Opt: ER</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The CALL operation passes control to the program specified in factor 2.

Factor 2 must contain the name of a field, a literal, or an array element that contains the name of the program called. Optionally it can contain the name of the library where the program is located. Factor 2 must contain a character entry, and if you specify the library name, the library name must be followed immediately by a slash and then the program name (for example: LIB/PGM).

The total length of a literal, including the slash, cannot exceed eight characters. The total length of a field, including the slash, cannot exceed 21 characters.

In the result field, you can specify the name of PLIST in order to communicate values between the calling program and the called program. The result field can be blank if the called program does not access parameters. (For further information on the PLIST and PARM operations, see “PLIST (Identify a Parameter List)” on page 805 and “PARM (Identify Parameters)” on page 803.)

Positions 54 and 55 must be blank. Any valid resulting indicator can be specified in positions 56 and 57, and set on for an error return from the called program. Any valid resulting indicator can be specified in positions 58 and 59, and set on if the called program is an RPG program that returns with the LR indicator on.

Remember the following when specifying CALL:

- Avoid using the CALL function to call an MRT program because unpredictable results can occur.

- 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 called program ends in error or issues a return code greater than 1, the indicators in positions 56 and 57 are set on.

- An RPG program cannot call itself or a program higher in the program stack. For example, if program A calls program B, then program B cannot call program A or 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.

- If, in factor 2 of a CALL operation, you use a literal to specify a program name without a library name, RPG locates the program in your library list only on the
first time the CALL operation runs. When that CALL operation runs again, the same program runs and your library list is not searched. If you want to search your library list each time a CALL operation runs, specify the program name in a field rather than with a literal.

- When the CALL operation runs, the calling program passes control to the called program. After the called program has run, control returns to the first statement that can be processed after the CALL operation in the calling program. If an error occurs while the CALL operation runs (for example, the called program is not found), the RPG exception/error handling routine receives control.

- An RPG program can call a System/36 Environment program, a System/38 Environment program, or an AS/400 system program.
The CASxx operation conditionally selects a subroutine for processing. The selection is based on the relationship between factor 1 and factor 2, as specified by the xx portion of the CASxx operation. See “Structured Programming Operations” on page 718 for options available under the xx portion of the CASxx operation.

Conditioning indicators can be specified. Conditioning indicators on the CASxx operation control if this particular CASxx operation is processed.

Factor 1 and factor 2 can contain a character literal, a numeric literal, an array element, a table element, a data structure subfield, a field name, or blanks (blanks are valid only for CASb). If factor 1 and factor 2 are not blanks, both must be character data or both must be numeric. The rules for comparing factor 1 and factor 2 on the CASxx operation are the same as those given under “Compare and Testing Operations” on page 717.

The result field must contain the name of a valid RPG subroutine. If the relationship denoted by xx exists between factor 1 and factor 2, control passes to the subroutine specified in the result field. If the relationship denoted by xx does not exist, the program continues with the next CASxx operation in the CAS group.

A CAS group can contain only CASxx operations. An END operation must follow the last CASxx operation to denote the end of the CAS group. After the subroutine runs, the program continues at the next operation following the END operation for the CAS group.

**Note:** For the INFSR subroutine, an optional factor 2 entry on the ENDSR operation specifies the return point for the subroutine. See “Coding the INFSR Subroutine” on page 198 in Chapter 7, “Using a WORKSTN File,” for the description of entries allowed in factor 2 of the ENDSR operation used with the INFSR subroutine. For all other subroutines, factor 2 of the ENDSR operation must be blank.

You must not use conditioning indicators on the END operation for a CAS group.

In a CASb operation, factor 1 and factor 2 are required only if resulting indicators are specified in positions 54 through 59.

The CASb operation with no resulting indicators specified in position 54 through 59 is functionally identical to an EXSR operation because it causes the unconditional processing of the subroutine named in the result field of the CASb operation. Any CASxx operations that follow an unconditional CASb operation in the same CAS group are never tested. Therefore, the normal placement of an unconditional CASb operation is after all other CASxx operations in the CAS group.

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Optional</td>
<td>CASxx</td>
<td>Optional</td>
<td>Required</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>CASxx</td>
<td>Required</td>
<td>Opt: HI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Opt: LO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Opt: EQ</td>
</tr>
</tbody>
</table>
The CASGE operation (line 04) compares FIELDA with FIELDB. If FIELDA is greater than or equal to FIELDB, SUBR01 runs and the program continues with the operation specified on line 08. If FIELDA is not greater than or equal to FIELDB, the program next compares FIELDA with FIELDC (line 05). If FIELDA is equal to FIELDC, SUBR02 runs and the program continues with the operation specified on line 08. If FIELDA is not equal to FIELDC, the CASbb operation (line 06) causes SUBR03 to run before the program continues with the operation specified on line 08. The END operation on line 07 denotes the end of the CAS group.
CHAIN (Chain)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>CHAIN</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>Blank</td>
<td>Blank</td>
<td>Opt: NR Blank Blank</td>
</tr>
<tr>
<td>7-8</td>
<td>9-17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Field 54-55 56-57 58-59</td>
</tr>
</tbody>
</table>

The CHAIN operation reads one record from a DISK file during calculations. The CHAIN operation can be used either to read records randomly from a sequential, direct, or indexed file, or to load a direct file that does not allow deletions. For more information on loading a direct file that does not allow deletions, see “DIRECT FILES” on page 110 in Chapter 6, “Using a DISK File.”

Enter the operation code CHAIN in columns 28 through 32. Factor 1 defines the relative record number or the key field of the record selected for processing. If you want to use noncontiguous keys, the key field name may be the name of a data structure subfield. Alternative indexes and data structures may be used to create noncontiguous key fields. See “Creating an Alternative Index File for an Indexed File” on page 137 in Chapter 6, “Using a DISK File.” Factor 2 names the chained file or full-procedural file from which the record is read. This file must be defined with a C or F entry in column 16 of the file description specifications.

Indicators can be used in columns 7 through 17, but columns 43 through 53 and 56 through 59 must be blank. If the chained file is conditioned by an external indicator on the file description specifications, the CHAIN statement should be conditioned by the same external indicator. A maximum of 50 files of all file types are allowed per program.

Columns 54 and 55 should specify an indicator. This indicator must be specified for full-procedural files. If the record is not found (or, for a direct file load, if the record location does not exist in the file), the indicator turns on. No update is permitted to a chained update file when the specified record is not found; however, adding records to a file is allowed. Records with duplicate key fields are possible in the file after an unsuccessful chain to an update-add file if the key field is changed before an add to the file. If the original record is found, the indicator turns off.

If an indicator is not specified in columns 54 and 55 and the record is not found, the program halts, and the person using the display station must respond to the error message. When chaining to a file with key fields in packed-decimal format, the field specified in factor 1 of the CHAIN operation must have a packed-decimal length that is the same as the length of the key field in the chained file. Packed-decimal key fields can be up to 8 bytes long. The packed-decimal field equivalents for zoned-decimal fields up to 15 bytes long are shown in a chart under “Packed-Decimal Format (P)” on page 572.

**Note:** If you chain to one or more files during the same RPG cycle, record-identifying indicators assigned to the chained file or files remain on throughout the cycle if the previous chain operations were processed successfully. If you chain to the same file more than once during an RPG cycle, only the last record processed is updated during output time unless an exception output is associated with each CHAIN operation.
Random Processing

To read a record from a sequential or direct file with the CHAIN operation, the record must be identified by relative record number. To read a record from an indexed file with the CHAIN operation, the record must be identified by a key field. A field can be specified to contain the relative record number or key field.

If the record has been deleted from the file, the no-record-found indicator is turned on. If the no-record-found indicator is not specified, a message is displayed.

Factor 1 must contain a relative record number, a key field, or the name of a field that contains a relative record number or key field. Factor 2 must contain the name of the file from which the record is read.

Figure 284 shows an example of chaining to and updating an indexed file.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME=IPEAFB 96 96 DISK
FMASTER INV UC F 120 120R 9AI 1 DISK
FMESSAGE ID F 9 9 KEYBORD

RECVIN file consists of records sorted by item number, with each record containing a quantity ordered.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FILENAME=SQ NORi POS1 NCC POS2 NCC POS3 NCC

IRCVIN AA 01 96 CX
I.......................... P FROM TO++ D FIELD + L1 M1 FR PO NE EQ ...
I 21 290 ITEMNO L1
I 30 360 QTY

IMASTER INV BB 02 120 C1
I 1 90 ITEMNO
I 10 180 QOH

ITEMNO is used as a control field. When all the quantities for one item number are added, a control break occurs.

Figure 284 (Part 1 of 2). CHAIN Operation
The CHAIN operation then uses ITEMNO to find the master record and update it. If it is not found, indicator 20 turns on and a SET operation displays the item number on the screen. If the master record is found, the total quantity for the item number is subtracted from the quantity on hand.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C 01 QTY ADD TOTQTY TOTQTY 70 DETAIL CALCS
CL1 ITEMNO CHAINMASTINV 20 FIND MASTER REC
CL1 20 ITEMNO SET FOUND?NO-DISPLY
CL1N20 QOH SUB TOTQTY QOH YES-TOTAL CALCS
C* " 
C* " 
C* Other total calculations

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
OName+++DFBASbSaN01N02N03Excptn..............................................
OMASTINV T L1 02
0....................N01N02N03Field+YBEnd+PConstant/editword++++++++++...
0 QOH 18
```

After the total calculations, the QOH field in the master record is updated.

*Figure 284 (Part 2 of 2). CHAIN Operation*
The COMP operation compares factor 1 with factor 2. As a result of the compare, indicators turn on as follows:

- **High**: Factor 1 is greater than factor 2.
- **Low**: Factor 1 is less than factor 2.
- **Equal**: Factor 1 equals factor 2.

Indicators for conditions not met remain off, or turn off if they had been turned on previously.

Factor 1 and factor 2 must be both alphameric or both numeric.

At least one resulting indicator must be specified in positions 54 through 59.

The fields are automatically aligned before they are compared. If the fields are alphameric, they are aligned on their leftmost character. If one is shorter, the unused positions are filled with blanks (see Figure 285 on page 738). The maximum field length for alphameric fields compared is 256 characters.

If the fields are numeric, they are aligned on the decimal point. Any missing digits are filled with zeros (see Figure 286 on page 738). The maximum field length for numeric fields to be compared is 15 digits.

If an alternative collating sequence is specified, alphameric fields are compared according to the alternative sequence.

Figure 287 on page 739 shows some examples of specifications for compare operations.
Figure 285. Comparison of Alphameric Fields

Figure 286. Comparison of Numeric Fields
The contents of the field SLS81 (1981 sales) are compared with the contents of SLS82. If 1981 sales exceed 1982 sales, resulting indicator 21 turns on; if they are less, indicator 26 turns on; if the two years had equal sales, indicator 30 turns on.

The alphameric constant OCTOBER is compared with the contents of the field named MONTH, which must also be defined as alphameric. If the MONTH field does not contain the word OCTOBER, indicator 13 turns on; if it does, indicator 15 turns on.

The contents of the field named GRSPAY, which must be defined as numeric, are decimal-aligned with numeric constant 1250.00. If the value in field GRSPAY is greater than or equal to 1250.00, indicator 04 turns on; if its value is less than 1250.00, indicator 05 turns on.

The contents of the field NETPAY, which must be defined as numeric, are decimal-aligned with numeric constant 0 and then compared to it. If NETPAY is greater than zero, indicator H1 remains off; but if NETPAY is zero or negative, indicator H1 turns on.

Figure 287. Compare Operations
The DEBUG operation is an RPG function that helps you find errors in a program that is not working properly. Either one or two records containing information helpful for finding programming errors are written to an output file as a result of this operation. All DEBUG output in a program is written to the same file.

The DEBUG operation code can be specified at any point or at several points in the calculation specifications. Whenever the program finds the DEBUG operation, either one or two types of records are written, depending upon the specifications entered. The first record contains a list of all indicators that are on at the time of the DEBUG operation. The second and subsequent records, if the result field is specified, show the contents of the field specified in the result field.

Factor 1 can contain a literal or the name of a field to help identify the particular DEBUG operation. The length of the specified field can be from one to eight characters. The contents of the field or the literal are written in the first record. If factor 1 is not used, the RPG-created statement number of the DEBUG operation code is written in the first record. Factor 2 must contain the name of an output file on which the DEBUG lines are written and can be any valid output file, except a CRT file. This file must have a record length of at least 80 positions. The same output file name must appear in factor 2 for all DEBUG statements in a program. The result field can contain the name of a field or array whose contents are written in the second record. Any valid indicator can be used in columns 7 through 17. Columns 49 through 59 must be blank.

To use the DEBUG operation, you must enter 1 in column 15 of the control specification. If this entry is not made, the DEBUG operation code and its conditioning indicators are treated as a comment. See “Column 15 (Debug)” on page 474 in Chapter 18, “Control Specification,” for more information.

Externally described files are not allowed with the DEBUG operation.
Records Written for DEBUG

For a DEBUG operation, the first record is always written and appears in the following format:

<table>
<thead>
<tr>
<th>Column Positions in the Output Record</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DEBUG =</td>
</tr>
<tr>
<td>9-18</td>
<td>Statement number of the DEBUG operation code in the program.</td>
</tr>
<tr>
<td>19-26</td>
<td>Literal or contents of field coded in factor 1 (optional)</td>
</tr>
<tr>
<td>27-28</td>
<td>Blanks</td>
</tr>
<tr>
<td>29-44</td>
<td>INDICATORS ON =</td>
</tr>
<tr>
<td>45-any position (depending on length of output record)</td>
<td>The names of all indicators that are on, each separated by a blank. More than one record may be needed.</td>
</tr>
</tbody>
</table>

The second record is written only when an entry is made in the result field. The record has the following format:

<table>
<thead>
<tr>
<th>Output Positions</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-14</td>
<td>FIELD VALUE =</td>
</tr>
<tr>
<td>15-any position (depending on length of field)</td>
<td>The contents of the result field (up to 256 characters). If the result field is an array, more than one output record may be needed to contain the array.</td>
</tr>
</tbody>
</table>
The DEFN operation with *LIKE in factor 1 defines a field based on the attributes of another field. These attributes include length, decimal positions, and type (character or numeric).

Factor 1 must always contain the entry *LIKE.

Factor 2 must contain the name of the field that provides the attributes for the field being defined. Factor 2 cannot be a literal or a data structure name. If factor 2 is an array, an array element, or a table name, the attributes of an element of the array or table are used to define the field.

The result field must contain the name of the field being defined. The result field cannot be an array, an array element, a table name, or a data structure name.

The DEFN operation can be specified anywhere within calculations. The control-level entry can be blank or can contain an L1 through L9 indicator, the LR indicator, or an L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (columns 9 through 17) are not permitted.

Columns 49 through 51 (field length) can be used to make the result field entry longer or shorter than the factor 2 entry. A plus sign (+) in column 49 indicates a length increase; a minus sign (-) in column 49 indicates a length decrease. Columns 50 and 51 can contain the increase or decrease in length (right-adjusted) or can be blank. If columns 49 through 51 are blank, the result field entry is defined with the same length as the factor 2 entry.

The number of decimal positions in the new field cannot be determined by an entry in column 52 (decimal positions). The new field will have the same number of decimal positions as the factor 2 entry.

Resulting indicators are not permitted.

See Figure 288 on page 743 for examples of *LIKE DEFN.
FLDP is a seven-position character field.  

\* \*LIKE \ DEFN FLDA FLDP

FLDQ is a nine-position character field.  

\* \*LIKE \ DEFN FLDA FLDQ + 2

FLDR is a six-position character field.  

\* \*LIKE \ DEFN FLDA FLDR - 1

FLDS is a five-position numeric field with two decimal positions.  

\* \*LIKE \ DEFN FLDB FLDS

FLDT is a six-position numeric field with two decimal positions.  

\* \*LIKE \ DEFN FLDB FLDT + 1

FLDU is a three-position numeric field with two decimal positions.  

\* \*LIKE \ DEFN FLDB FLDU - 2

FLDX is a three-position numeric field with two decimal positions.  

\* \*LIKE \ DEFN FLDU FLDX

FLDA is a seven-position character field.  

\* \*LIKE \ DEFN FLDB FLDA

FLDB is a five-digit field with two decimal positions.  

\* \*LIKE \ DEFN FLDU FLDB

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
Factor 1 (dividend) is divided by factor 2 (divisor). The quotient (result) is placed in the result field. Factor 1 and factor 2 are not changed. If factor 1 is 0, the result of the divide operation is 0. Factor 2 cannot be 0. If it is, the job stops immediately. The person using the display station can continue processing, however, by responding to the error message. When processing continues, the result and remainder are set to 0. If factor 1 is not present, the result field is divided by factor 2, and the quotient is placed in the result field. Any remainder resulting from the divide operation is lost unless the move remainder (MVR) operation is specified as the next operation. When you use the move remainder operation to save the remainder, you cannot half-adjust the result of the divide operation.
The DO operation begins a group of operations that you do a fixed number of times. You indicate how many times to process this group of operations by specifying a starting value, a limit value, and an index value on the DO statement line.

An associated END operation marks the end of the do group.

In factor 1 (starting value) specify a numeric literal, a numeric field name, an array element, a table name, or a data structure subfield with zero decimal positions. If factor 1 is not specified, the starting value is assumed to be 1.

In factor 2 (limit value) specify a numeric literal, a numeric field name, an array element, a table name, or a data structure subfield with zero decimal positions. If factor 2 is not specified, the limit value is assumed to be 1.

In the result field, which is used to contain the current index value, specify a numeric field name, an array element, a table name, or a data structure subfield with zero decimal positions. If you do not specify an index field, the compiler will generate one for internal use.

Note that any value in the index field is replaced by factor 1 when the DO operation begins.

An increment value of the DO operation is specified in factor 2 on the associated END statement line. It can be a numeric positive literal, field, array element, table name, or data structure subfield with zero decimal positions. If factor 2 on the associated END operation is not specified, the increment value is 1.

Conditioning indicators can be specified. In addition to the DO operation itself, the conditioning indicators on the DO and END statements control the do group, as explained below:

1. If the conditioning indicators on the DO statement are satisfied, the DO operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation that can be processed following the associated END statement (step 7).

2. RPG begins the DO operation by moving the starting value (factor 1) to the index field (result field).

3. If the index value is greater than the limit value, control passes to the next operation that can be processed following the associated END statement. Otherwise, control passes to the first operation after the DO statement (step 4).

4. Each of the operations between the DO statement and the associated END statement is done.
5. If the conditioning indicators on the END statement are not satisfied, control passes to the next operation that can be processed following the associated END statement (step 7). Otherwise, the END operation is processed (step 6).

6. RPG processes the END operation by adding the increment to the index field. Control passes to step 3. Note that, unlike step 1, the conditioning indicators on the DO statement are not tested again when control passes to step 3.

7. The operation after the associated END statement is processed when the conditioning indicators on the DO or END statements are not satisfied (steps 1 or 5), or when the index value is greater than the limit value (step 3).

The rules applied when comparing the result field (index value) and factor 2 (limit value) are the same as those given for numeric comparison under “Compare and Testing Operations” on page 717 earlier.

Remember the following when specifying the DO operation and the associated END operation:

- The limit value you specify in factor 2 must be equal to or greater than the starting value specified in factor 1, otherwise the do group will never run.
- Specifying a negative or zero increment value in factor 2 of the associated END operation can cause the program to loop indefinitely.
- The index value, increment, limit value, and indicators can be modified within the do group to affect the ending of the loop.
- The length of the RPG-generated is always 15.

Figure 289 on page 747 illustrates how the DO operation works.
Indicator 17 is tested only once. If indicator 17 is off, processing will continue following the END statement (line 07). If indicator 17 is on, the do group (lines 2 through 6) is processed 10 times. Processing stops when the index value in field X (the result field) is greater than the limit value (10) in factor 2. Control then passes to the operation immediately following the END operation (line 7). Because factor 1 on the DO statement line is not specified, the starting value is 1. Because factor 2 on the END statement line is not specified, the increment value is 1.

The do group (lines 9 through 13) can be processed 10 times. Processing stops when the value in the index field (generated by the compiler because the result field is not specified) is greater than the limit value (20) in factor 2, or if indicator 50 is not on when the END operation is found. When indicator 50 is not on, the END operation is not processed; therefore, control passes to the operation following the END operation. The starting value of 2 is specified in factor 1 of the DO operation, and the increment value of 2 is specified in factor 2 of the END operation.
The DOUxx operation begins a group of operations that are processed until a condition specified in the xx portion of the DOUxx operation is met. See “Structured Programming Operations” on page 718 for options available under the xx portion of the DOUxx operation code. An associated END operation marks the end of the do group.

Factor 1 and factor 2 can contain a character literal, a numeric literal, an array element, a table element, a data structure subfield, or a field name. Both factor 1 and factor 2 must be character data or both must be numeric. The rules for comparing factor 1 and factor 2 on the DOUxx operation are the same as those given under “Compare and Testing Operations” on page 717.

The DOUEQ operation runs the operation within the do group (line 04) at least once. The END operation (line 05), causes the program to branch to line 03 where the test is processed again to determine if FLDA is equal to FLDB. If FLDA does not equal FLDB, line 04 is processed again. This loop continues processing until FLDA is equal to FLDB. When this occurs, the program branches to the operation immediately following the END operation (line 06).

Conditioning indicators can be specified. In addition to the DOUxx operation itself, the conditioning indicators on the DOUxx and END operations control the do group. The conditioning indicators on the DOUxx operation control if the DOUxx operation is begun, while the conditioning indicators on the associated END operation control if the do group is repeated.

Figure 291 on page 749 shows how the DOUxx operation with conditioning indicators works.
1. If the conditioning indicators on the DOUxx operation are satisfied, the DOUxx operation is processed (line 2). If the indicators are not satisfied, control passes to the operation following the associated END operation (line 6).

2. RPG processes the DOUxx operation by passing control to the next operation (line 3). The DOUxx operation does not compare factor 1 and factor 2 at this point.

3. Each operation of the do group is processed.

4. If the conditioning indicators on the END operation are not satisfied, control passes to the operation following the END operation (line 6). Otherwise, the END operation is processed (line 5).

5. RPG processes the END operation by comparing factor 1 and factor 2 of the DOUxx operation. If the relationship xx exists between factor 1 and factor 2, the do group is finished and control passes to the next operation after the END operation (line 6). If the relationship xx does not exist between factor 1 and factor 2, the operations in the group are repeated (line 3). Note that at this point the conditioning indicators on the DOUxx operation are not tested again.

6. Control passes to the operation following the END operation when the conditioning indicators on the DOUxx or END statements are not satisfied (lines 2 or 5), or when the relationship xx exists between factor 1 and factor 2 at line 5.

---

Figure 291. DOUxx Operation Using Conditioning Indicators
The DOWxx operation begins a group of operations that run while the relationship xx exists between factor 1 and factor 2. See “Structured Programming Operations” on page 718 for options available under the xx portion of the DOWxx operation code. An associated END operation marks the end of the do group.

Factor 1 and factor 2 can contain a character literal, a numeric literal, an array element, a table element, a data structure subfield, or a field name. Both factor 1 and factor 2 must be alphameric data, or both must be numeric. The rules for comparing factor 1 and factor 2 on the DOWxx operation are the same as those given under “Compare and Testing Operations” on page 717.
1. If the conditioning indicators on the DOWxx operation are satisfied, the DOWxx operation is processed (line 2). If the indicators are not satisfied, control passes to the operation following the associated END operation (line 6).

2. RPG processes the DOWxx operation by comparing factor 1 and factor 2. If the relationship xx does not exist between factor 1 and factor 2, the do group is finished and control passes to the operation following the END operation (line 6). If the relationship xx does exist between factor 1 and factor 2, the operations in the do group are done (line 3).

3. Each of the operations in the do group is done.

4. If the conditioning indicators on the END operation are not satisfied, control passes to the next operation processed following the END operation (line 6). Otherwise, the END operation is processed (line 5).

5. RPG processes the END operation by passing control to the DOWxx operation (line 2). Note that the conditioning indicators on the DOWxx operation at line 2 are not tested again.

6. Control passes to the operation following the END operation when the conditioning indicators on the DOWxx or END operation are not satisfied (lines 2 or 5), or when the relationship xx does not exist between factor 1 and factor 2 at line 2.

Figure 293. DOWxx Operation Using Conditioning Indicators
The ELSE operation is optional with the IFxx operation. ELSE is specified immediately following the calculations that are processed if the IFxx comparison is met, and is immediately followed by the calculations done if the IFxx comparison is not met.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statements within the appropriate section of the program. The control level entry is for documentation purposes only. Conditioning indicator entries (positions 9 through 17) are not permitted.

An END operation must be used to close the IFxx/ELSE group.
The END operation specifies the end of a CASxx, DO, DOUxx, DOWxx, or IFxx group.

This is how the END operation for the DO operation should be specified:

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td>54–55</td>
<td>56–57 58–59</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>END</td>
<td>Blank</td>
<td>Blank Blank Blank</td>
</tr>
</tbody>
</table>

The table below shows how the END operation for the DOUxx and DOWxx operations should be specified:

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td>54–55</td>
<td>56–57 58–59</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>END</td>
<td>Blank</td>
<td>Blank Blank Blank</td>
</tr>
</tbody>
</table>

And the following table shows how the END operation for the IFxx and CASxx operations should be specified:

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td>54–55</td>
<td>56–57 58–59</td>
</tr>
<tr>
<td>Optional</td>
<td>Blank</td>
<td>Blank</td>
<td>END</td>
<td>Blank</td>
<td>Blank Blank Blank</td>
</tr>
</tbody>
</table>

Note that a factor 2 entry is allowed only on an END operation associated with a DO operation. In this case, factor 2 of the END operation contains the increment value of the DO operation. If specified, factor 2 must contain a numeric positive value, which can be a literal, a field name, an array element, a table name, or a data structure subfield with zero decimal positions. If factor 2 is not specified on the associated END operation, the increment value of the DO operation is 1.

For an explanation of how conditioning indicators affect the END operation, see descriptions of the DO, DOUxx, and DOWxx operation codes. Do not use conditioning indicators on the END operation for the CASxx or IFxx operations.
ENDSR (End Subroutine)

The ENDSR operation defines the end of a subroutine; therefore, it must be the last statement in the subroutine. Factor 1 can contain a name that can be used as a point to which a GOTO operation within the subroutine can branch. The control-level entry (columns 7 and 8) can be SR or blank. Columns 9 through 17 must not contain any conditioning indicators.

The ENDSR operation ends the subroutine and automatically causes a branch back to the statement that follows the EXSR operation unless the subroutine is the INFSR (exception/error-processing) subroutine. For the INFSR subroutine, an optional factor 2 entry on the ENDSR operation specifies the return point for the subroutine. The valid entries for factor 2 for the INFSR subroutine are described in Chapter 7, “Using a WORKSTN File,” under “Coding the INFSR Subroutine” on page 198. For all other subroutines, factor 2 must not contain an entry.
EXCPT (Exception Output)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Blank</td>
<td>EXCPT</td>
<td>Blank</td>
<td>Blank 54–55 56–57 58–59</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank 54–55 56–57 58–59</td>
</tr>
</tbody>
</table>

The EXCPT operation allows your program to write records during detail or total calculation time instead of the normal time during the RPG program cycle. Consider the following when specifying the EXCPT operation:

- **On the calculation specifications:**
  - Columns 28 through 32 must contain EXCPT to indicate when records are written during calculation time.
  - Columns 7 through 17 can contain indicators.
  - Factor 2 can contain an EXCPT name. This EXCPT name can specify a group of exception lines to be written, reducing the need for indicators to condition the exception lines that are to be written. This name must follow the rules for field names. See “Rules for Field Names” on page 612 in Chapter 25, “Output Specifications.”
  - All other columns must be blank.

- **On the output specifications:**
  - Column 15 must contain an E to indicate the lines that are written during calculation time.
  - Columns 23 through 31 can contain indicators with or without group names specified.
  - Columns 32 through 37 can contain an EXCPT name for a group of records written during calculation time. This name must be the same name specified in factor 2 on a calculation specification containing the EXCPT operation code. The same name can be on multiple EXCPT output record lines.
  - Columns 38 through 74 must be blank.

- Only exception records, not heading, detail, or total records, can contain an EXCPT name.

- When the EXCPT operation with a name in factor 2 on the calculation specifications occurs, only those exception records (E in column 15) with the same name in columns 32 through 37 on the output specifications are written if the conditioning indicators are satisfied.

- When factor 2 on the calculation specifications with the EXCPT operation code is blank, only those exception records with no name in columns 32 through 37 on the output specifications are written if the conditioning indicators are satisfied.
Overflow indicators cannot be used in columns 23 through 31 of the output specifications when there is an E in column 15.

Only one EXCPT name can be blank. EXCPT names cannot be the same as a filename, field name, data structure name, array name, table name, label, or subroutine name used in your program.

See Figure 294 through Figure 296 for examples of the EXCPT operation.

![Figure 294. EXCPT Operation with/without Factor 2 Specified](image-url)
1. When the EXCPT operation with HDG specified in factor 2 is processed (line 01 in calculation specifications), all exception records with the EXCPT name HDG (lines 04 and 07 of the output specifications) are written.

2. When the EXCPT operation with DETAIL specified in factor 2 is processed (line 04 of the calculation specifications), all exception records with the EXCPT name DETAIL (line 10 of the output specifications) are written.

3. When the EXCPT operation with no entry in factor 2 is processed (line 07 of the calculation specifications), all exception records that do not have an EXCPT name specified in positions 32 through 37 (such as line 01 of the output specifications) are written if the conditioning indicators are satisfied. Any exception records without conditioning indicators and without an EXCPT name are always written by an EXCPT operation with no entry in factor 2.
EXCPT (EXCEPTION OUTPUT)

| Line | Form Type | Indicators | Control Level (L0-L9) | Factor 1 | Operation | Factor 2 | Name | Length | Result Field | Decimal Positions | Half Adjust (H) | Resulting Indicators | Arithmetic | Plus Minus Zero | Compare | Lookup(Factor 2) is | High Low Equal |
|------|-----------|------------|-----------------------|----------|-----------|----------|------|--------|-------------|----------------|----------------|------------------|------------|-----------------|---------|----------------|---------|-----------------|----------|
| 01   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 02   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 03   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 04   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 05   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 06   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 07   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |
| 08   | C         |            |                       |          |           |          |      |        |              |                 |                 |                  |            |                 |         |                 |         |

1. When the EXCPT operation is found in the calculation specifications (line 03), all exception records in the output specifications that do not have an EXCPT name are written if the conditioning indicators allow output. Line 01 of the output specifications is always output by the EXCPT operation because it is unconditioned.

2. When the SETON operation sets on indicator 15 in the calculation specifications (line 06) and the EXCPT operation on line 07 is processed, all lines in the output specifications that have no EXCPT name in positions 32 through 37 and that are unconditioned or conditioned by indicator 15 (lines 01, 04, and 06) are written.

3. The SETOF operation sets off indicator 15 (line 08).

Figure 295. EXCPT Operation without Factor 2 Specified
Figure 296 (Part 1 of 2). EXCPT Output with an Overflow Indicator
This example shows the coding for EXCPT output with an overflow indicator when you are printing the title and column headings on each page of a report.

1. The EXCPT operation with HDG in factor 2 (line 01 of the calculation specifications) prints all lines with the group EXCPT name HDG (lines 01 and 04 of the output specifications).

2. The EXCPT operation with DETAIL specified in factor 2 (line 07 of the calculation specifications) prints all lines with the group EXCPT name DETAIL (line 07 of the output specifications).

3. When the overflow indicator is set on, the EXCPT operation (line 08 of the calculation specifications) prints all HDG lines (lines 01 and 04 of the output specifications) on the overflow page.

4. The SETOF operation sets off the OF indicator (line 09 of the calculation specifications), and the program branches to the label specified in the GOTO operation.

*Figure 296 (Part 2 of 2). EXCPT Output with an Overflow Indicator*
EXIT (Exit to an External Subroutine)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Blank</td>
<td>EXIT</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>EXIT</td>
<td>Blank</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The EXIT operation designates the point in the calculation specifications at which control is transferred from an RPG program to an external subroutine or program. To an AS/400 system, programs and subroutines are identical, even in the System/36 Environment. Because they all have the same object type (*PGM), you can use the terms interchangeably.

RLABL must be specified immediately after EXIT.

The rules for use of the EXIT operation on the calculation specifications are as follows:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation (28-32)</td>
<td>EXIT</td>
</tr>
<tr>
<td>Factor 1 (18-27)</td>
<td>Blank</td>
</tr>
<tr>
<td>Factor 2 (33-42)</td>
<td>The name of the subroutine to which control is passed. The name must consist of five or six characters, the first four of which are SUBR. The remaining characters must be alphabetic for user-written subroutines. (Numeric characters are reserved for IBM-supplied subroutines.) The module name and entry point name must be the same.</td>
</tr>
<tr>
<td>Result field (43-48)</td>
<td>Blank</td>
</tr>
<tr>
<td>Resulting indicators (54-59)</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The EXIT operation can be controlled by a control-level indicator (columns 7 and 8) and conditioning indicators (columns 9 through 17). If no control-level indicator is used, the EXIT operation occurs at detail calculation time.

The position of the EXIT operation in the calculation specifications of the RPG program determines when the actual subroutine processing occurs (see the table below).
The table below shows the relationship between the position of the EXIT operation and processing of the subroutine.

<table>
<thead>
<tr>
<th>Position</th>
<th>Processing of the Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>First detail line in calculation specifications</td>
<td>Immediately following data routine file, that is, after data is extracted from input record</td>
</tr>
<tr>
<td>Last detail line in calculation specifications</td>
<td>Immediately before heading records output time</td>
</tr>
<tr>
<td>First total line in calculation specifications</td>
<td>Immediately following input routine (after determination of record type and testing for control-level break)</td>
</tr>
<tr>
<td>Last total line in calculation specifications</td>
<td>Immediately before total records output time</td>
</tr>
<tr>
<td>Any other detail or total line in calculation specifications</td>
<td>Immediately following the previous calculation operation</td>
</tr>
</tbody>
</table>

Remember the following when specifying EXIT:

- An RPG program cannot call itself or a program higher in the program stack. For example, if program A calls program B, then program B cannot call program A or program 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.

- In factor 2 of an EXIT operation, you use a literal to specify a subroutine name without a library name. RPG locates the subroutine in your library list when the EXIT operation runs.

- There is an extra parameter passed as a RLABL result field. It is an array, containing the attributes of the other passed parameters.
EXSR (Execute Subroutine)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Blank</td>
<td>EXSR</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The EXSR operation passes control to the subroutine named in factor 2. The EXSR operation can appear anywhere in the program. Whenever it appears, the subroutine is given control. After operations in the subroutine are processed, the operation in the line following the EXSR operation is processed.

The EXSR operation can be conditioned by any indicators; therefore, the subroutine is given control only when all conditions are satisfied. Any valid indicator can be used in columns 7 through 17. If no indicators are used, the subroutine is always given control.

Factor 2 must contain the name of the subroutine that is to be processed. This name must appear on a BEGSR operation.

Coding Subroutines

All RPG operations can be processed within a subroutine, and these operations can be conditioned by any valid indicators in columns 9 through 17. Because SR or blanks must appear in columns 7 and 8, control-level indicators cannot be used in these columns. However, AND/OR lines within the subroutine can be indicated in columns 7 and 8.

Fields used in a subroutine can be defined either in the subroutine or in the main program. In either instance, the fields can be used by both the main program and the subroutine.

Any number of subroutines can be included in a program; however, a subroutine cannot contain another subroutine. One subroutine can call another subroutine; that is, a subroutine can contain an EXSR or CASxx operation code. However, a subroutine cannot call itself directly or with another subroutine.

You do not have to specify subroutines in the order they are used. Each subroutine must have a unique name and must contain a BEGSR and ENDSR operation.

See Figure 297 on page 764 for an example of coding a subroutine.
EXSR (EXECUTE SUBROUTINE)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpctFactor2+++ResultLenDHHiLoEqComments+++++++
C*   "
C*   "
C   EXSR SRTNB
C*   "
C*   "
CL2  EXSR SRTNA
C*   "
C*   "
CSR  SRTNA BEGSR
CSR  1   "
CSR 20NH1N01   "
C*  
C* One subroutine can call another subroutine.
C*
CSR 50   EXSR SRTNC
CSR   "
CSR   "
CSR ENDSR
CSR  SRTNB BEGSR
CSR   "
CSR   "
C*  
C* GOTO and TAG operations can be used within a subroutine.
C*
CSR  START TAG
CSR   "
CSR GOTO END
CSR   "
CSR   "
CSR GOTO START
CSR END ENDSR
C  SRTNC BEGSR
C*   "
C* Calculation Operations
C*   "
C ENDSR

Figure 297. Example of Coding Subroutines
The FORCE operation allows selection of the file from which the next record is read. The FORCE operation can be used for primary or secondary input and update files; however, it cannot be used to read from files assigned to a KEYBORD or WORKSTN device.

Factor 2 in a FORCE operation identifies the file from which the next record is selected. If the operation is processed, the record is read at the start of the next program cycle. If more than one FORCE operation is processed during the same program cycle, all but the last are ignored. FORCE should not be specified at total time.

FORCE operations override the multifile processing method by which the program normally selects records. However, the first record processed is always selected by the normal method. The remaining records can be selected by FORCE operations.

Figure 298 shows how the FORCE operation can be used to control input from primary and secondary files.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
FFilenameIPEAFBlenRlenLK1AI0vKlocEDevice+.......Exit++......A....U+.*
FINPUT1 IPE F 64 64 DISK
FINPUT2 IS F 64 64 DISK
FOUTPUT O F 64 64 DISK
```

*Figure 298 (Part 1 of 2). Example of FORCE Operation Controlling Input*
FORCE (FORCE)

1. 2 3 4 5 6 7

IFilenameSqNORiPos1NCCPos2NCCPos3NCC

INPUT1 AA 01 1 C1

PFromTo++DField++L1M1FrPoNeEq...
I 2 20NBR
I 3 8 FIELDA
I 9 24 FIELDB
I 25 30 FIELDC

INPUT2 BB 02 1 C1

I 13 18 FIELDA
I 20 35 FIELDB
I 38 43 FIELDC

I*

CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments+++++++C*

The NBR field of each primary record (INPUT1) contains the number C* of secondary records (INPUT2) read and written after each primary C* record is read. If NBR is less than or equal to zero, a halt C* occurs. No primary or secondary records are read. Processing C* begins with the next primary record according to normal selection.

C 01 NBR COMP 0 H1H1
C* If NBR is greater than zero, the NBR field is reduced by one and C* tested. If the result is not negative, the FORCE operation calls C* for input from the secondary file (INPUT2) on the next program C* cycle. The primary record is written, and secondary records are C* read and written until the NBR field tests negative (indicator 03 C* is on). When indicator 03 turns on, the FORCE operation calls for C* input from the primary file on the next program cycle.

C NH1 NBR SUB 1 NBR 03
C N03NH1 FORCEINPUT2 NEXT CYCLE SEC
C 03NH1 FORCEINPUT1 NEXT CYCLE PRI
C*

OUTPUT D 01
0 OR 02
0.............N01N02N03Field+YBEnd+PConstant/editword+++++++++++*
0 FIELD 10
0 FIELDB 27
0 FIELDC 33
0*

Figure 298 (Part 2 of 2). Example of FORCE Operation Controlling Input
FREE (Deactivate a Program)

The FREE operation removes a program from the list of activated programs and ensures program initialization (first-cycle processing) the next time the program is called. It does not close files.

To use the FREE operation you must specify a program name (for example PROG) or a program name and library name (for example LIB/PROG).

Factor 2 must contain the name of a field, a literal, or an array element that contains the name of the program deactivated. The entry in factor 2 must be character data. Positions 56 and 57 can specify to set on any valid resulting indicator if FREE is not successful.

Remember the following when specifying FREE:

If the FREE operation is not successful, the RPG exception/error handling routine receives control.
The GOTO operation allows skipping of operations by instructing the program to go to (or branch to) another operation. A GOTO operation can be used to specify a branch:

- To a previous or a succeeding specification line
- From a detail calculation line to another detail calculation line
- From a total calculation line to another total calculation line.

However, a branch cannot be made:

- From a detail calculation line to a total calculation line or vice versa.
- From calculations conditioned by L0 through L9 to calculations conditioned by LR or vice versa. (A total calculation line is defined as one that is conditioned by a control-level indicator in columns 7 and 8 of the calculation specifications.)
- From a subroutine to other calculations or vice versa.

Factor 2 must contain the name of the label to which the program is to branch. This label is entered in factor 1 of a TAG operation. If the GOTO is within the subroutine, the label can be specified in factor 1 of the ENDSR operation. The label can be from one to six characters long and must begin with an alphabetic character in column 33. The remaining characters can be any combination of alphabetic or numeric characters. Blanks must not appear between characters in the label.

Factor 1 and the result field are not used in this operation. The GOTO operation can be conditioned by any indicators. If no indicators are specified, the operation is always done.

See Figure 299 on page 769 and Figure 300 on page 770 for examples of the GOTO operation.
<table>
<thead>
<tr>
<th>Line</th>
<th>Factor 1</th>
<th>Factor 2</th>
<th>Operation</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>FIELD A</td>
<td>FIELD B</td>
<td>SUB</td>
<td>FIELD B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>FIELD A</td>
<td>10</td>
<td>GOTO</td>
<td>RTN1</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>FIELD A</td>
<td>FIELD B</td>
<td>MUL T 4</td>
<td>SAVE 82</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>FIELD A</td>
<td>GOTO</td>
<td>RTN1</td>
<td>TAG</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>RTN2</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>RTN2</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>RTN1</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>RTN1</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>RTN1</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>RTN2</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>RTN2</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>N15</td>
<td>GOTO</td>
<td>RTN2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>TESTZ</td>
<td>FIELD C</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>GOTO</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>MHLZO</td>
<td>FIELD C</td>
<td>FIELD D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>END</td>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. If the result of the subtraction in line 01 is minus (indicator 10 is on), a branch is taken to RTN1 (routine 1) named by the TAG operation code in line 09. Notice that neither the GOTO (line 02) nor the TAG (line 09) is conditioned by control-level indicators.

2. If the branch is not taken in line 02, the multiplication in line 03 is processed. Then the branch to RTN1 (line 09) must be taken because this branch is not conditioned by indicators.

3. Operations in lines 10 through 12 are then done. If the operation in line 12 does not turn indicator 15 on, a branch is taken backwards to RTN2 (line 05).

4. Operations are then processed again in the order specified from lines 06 through 12. Nothing is done in line 09 because TAG gives only a name. These same operations are processed again and again until indicator 15 does turn on.

5. When indicator 15 is on, the branch to RTN2 is not taken. The TESTZ operation is then processed. If this operation causes indicator 20 to turn on, a branch is taken to line 17 (GOTO END). If indicator 20 is not on, the operation in line 16 is done.

Figure 299. Using GOTO and TAG (Skipping Operations)
Assume you want to make eight mailing labels for every customer you have. The customer’s name and address are found on an input record. Because you want to write eight labels for each record, you can use exception lines and the EXCPT operation instead of coding eight identical output line specifications. (See “EXCPT (Exception Output)” on page 755 for further information.)

However, by using branching, you can code it all in six lines as shown below. An EXCPT line is printed out. One is added to COUNT to keep track of how many times the line is printed. Then COUNT is compared to 8. If COUNT does not equal 8, a branch is taken back to the beginning (GOTO DOAGIN). If COUNT equals 8, the branch is not taken. Instead, the COUNT field is set to zero for the next cycle.

**Figure 300. Using GOTO and TAG to Eliminate Duplicate Coding**
IFxx (IF/THEN)

The IFxx operation allows a group of calculations to be processed if a relationship specified in the xx portion of the IFxx operation exists between factor 1 and factor 2. See “Structured Programming Operations” on page 718 for options available under the xx portion of the IFxx operation code.

Conditioning indicators can be specified. Conditioning indicators on the IFxx operation control if the IFxx operation (and the entire IFxx/ELSE group) will be processed, or if control will be passed to the operation immediately following the associated END operation.

Factor 1 and factor 2 can contain a character literal, a numeric literal, an array element, a table element, a data structure subfield, or a field name. Both factor 1 and factor 2 must be character data, or both must be numeric. The rules for comparing factor 1 and factor 2 on the IFxx operation are the same as those given under “Compare and Testing Operations” on page 717.

If the relationship between factor 1 and factor 2 does not exist and an ELSE operation is not specified, control passes to the first operation that will run following the associated END operation. If the relationship between factor 1 and factor 2 does not exist and an ELSE operation is specified, control passes to the first operation following the ELSE operation.

Conditioning indicator entries on the END operation associated with IFxx must be blank.

An END operation must be used to close an IFxx group. If an IFxx operation is followed by an ELSE operation, an END operation is needed after the ELSE operation but not after the IFxx operation.

Figure 301 on page 772 is an example of code using IFxx/END and IFxx/ELSE/END structures.
### IFxx (IF/THEN)

<table>
<thead>
<tr>
<th>Line</th>
<th>Operation</th>
<th>Result Field</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>C</td>
<td>FLDA IF EQ FLDB</td>
<td>IF EQUAL</td>
</tr>
<tr>
<td>04</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>C</td>
<td>END</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>C</td>
<td>FLDA IF EQ FLDB</td>
<td>IF EQUAL</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td>ELSE</td>
<td>IF NOT EQUAL</td>
</tr>
<tr>
<td>15</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>C</td>
<td>END</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Number of sheets per pad may vary slightly.*

**Figure 301. IFxx/END and IFxx/ELSE/END Designs**

1. If FLDA equals FLDB (line 03), the do group (lines 04 through 06) is processed. If FLDA does not equal FLDB, the program branches to the operation immediately following the END statement (line 07).

2. If FLDA equals FLDB (line 11), the calculations in lines 12 and 13 are processed and control passes to the operation immediately following the END statement (line 18). If FLDA does not equal FLDB, control passes to the ELSE statement (line 14) and the calculations in lines 15 and 16 are processed.
KEY (Key)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Optional</td>
<td>KEY</td>
<td>Blank</td>
<td>Required</td>
</tr>
</tbody>
</table>

The KEY operation causes a pause in calculation during which the person using the display station can enter data from the keyboard. All KEY operations are directed to the display station that loaded the program.

Factor 1 can contain the constant, literal, field name, or table or array element displayed. Factor 2 must be blank. The result field can contain the name of the field entered. See "Calculation Specifications for a KEY Operation" on page 299 in Chapter 10, "Using a CONSOLE, KEYBORD, or CRT File," for more information on coding the KEY operation.

To use the KEY operation code, you must specify the device name KEYBORD in columns 40 through 46 of the file descriptions specifications. KEY can be used only with a KEYBORD input file. As the person enters data, it is displayed on the screen in one of two forms:

- If the record length is 40 or less, the display consists of six lines, with 40 characters per line, centered both vertically and horizontally on the screen.
- If the record length is greater than 40, the display consists of 24 lines with 79 characters per line (one character is reserved for field attributes).

When the KEY operation is used, the contents of the result field are determined by the person's response. The possible responses are:

- The person types in the data and presses an entry function key. If not all positions of a field are entered, numeric fields are right-justified and padded to the left with zeros; alphameric fields are padded to the right with blanks.
- The person presses only an entry function key, which changes any data in the result field to zero or blank.
- The person presses the Dup key and then an entry function key, which does not modify the data in the result field.

**Note:** The person can use any one of the following four keys as an entry function key: Field Exit, Field-, Field+, or Enter. If data has been entered into a numeric field, the person must press one of the Field keys, then press the Enter key.
Array LOKUP

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>LOKUP</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>One required</td>
</tr>
</tbody>
</table>

Table LOKUP

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>LOKUP</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>One required</td>
</tr>
</tbody>
</table>

The LOKUP operation searches for a particular element in an array or table. The array or table is named in factor 2. Factor 1 is the search word (data for which you want to find a match in the array or table named).

Factor 1, the search word, can be:

- An alphabetic or numeric constant
- A field name
- An array element
- A table name.

When a table is named in factor 1, it refers to the element of the table last selected in a LOKUP operation, not to the whole table.

Resulting indicators are always used with a LOKUP operation. The indicators first specify the type of search and then reflect the result of the search. The specified indicator turns on only if the search is successful.
Resulting indicators specify the type of search and reflect the result of the search in the following manner:

- A resulting indicator assigned to equal (columns 58 and 59) instructs the program to search for an entry in the array or table equal to the search word. The first equal entry found turns on the indicator assigned to equal.
- An indicator assigned to low (columns 56 and 57) instructs the program to find the entry in the array or table that is nearest to, yet lower in sequence than, the search word. The first such entry found turns on the indicator assigned to low.
- The indicator assigned to high (columns 54 and 55) instructs the program to find the entry in the array or table that is nearest to, yet higher in sequence than, the search word. The first such entry found turns on the indicator assigned to high.

At least one resulting indicator must be assigned, but no more than two can be used. Resulting indicators can be assigned to equal and high or to equal and low. The program searches for an entry that satisfies either condition with equal given precedence; that is, if no equal entry is found, the nearest lower or nearest higher entry is selected. If resulting indicators are assigned both to high and to low, the indicator assigned to low is ignored.

When you use the LOKUP operation, consider the following:

- The search word and each array or table element must have the same length and the same format (alphabetic or numeric).
- A search can be made for high, low, high and equal, or low and equal only if the array or table is in sequence. The sequence must be indicated in column 45 of the extension specifications.
- No resulting indicator turns on if the search is not successful.
- If an index is used with an array, the search starts at that element.
- If a variable index is used with an array, the field used as the index is set to the number of the array element found if the search is successful. If the search is unsuccessful, the index is set to 1.

For more information, see “Searching Arrays and Tables” on page 408 in Chapter 14, “Using Arrays and Tables.”
### MHHZO (Move High to High Zone)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Optional</td>
<td>Blank</td>
<td>MHHZO</td>
<td>Required Required Blank Blank Blank</td>
</tr>
</tbody>
</table>

The MHHZO operation moves the zone from the leftmost position of factor 2 to the leftmost position of the result field. Factor 2 and the result field must be alphabetic.

### MHLZO (Move High to Low Zone)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Optional</td>
<td>Blank</td>
<td>MHLZO</td>
<td>Required Required Blank Blank Blank</td>
</tr>
</tbody>
</table>

The MHLZO operation moves the zone from the leftmost position of factor 2 to the rightmost position of the result field. Factor 2 must be alphabetic. The result field can be alphabetic or numeric.

### MLHZO (Move Low to High Zone)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Optional</td>
<td>Blank</td>
<td>MLHZO</td>
<td>Required Required Blank Blank Blank</td>
</tr>
</tbody>
</table>

The MLHZO operation moves the zone from the rightmost position of factor 2 to the leftmost position of the result field. Factor 2 can be numeric or alphabetic, but the result field must be alphabetic.
**MLLZO (Move Low to Low Zone)**

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Result Field</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>MLLZO</td>
<td>Required</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The MLLZO operation moves the zone from the rightmost position of factor 2 to the rightmost position of the result field. Factor 2 and the result field can be either alphameric or numeric.

Functions of the four move zone operations are shown in Figure 302.
The MOVE operation transfers characters from factor 2 to the rightmost positions in the result field. Moving starts with the rightmost character of factor 2. If factor 2 is longer than the result field, the excess leftmost characters of factor 2 are not moved. If the result field is longer than factor 2, the excess leftmost characters in the result field are unchanged.

When an alphameric field is moved to a numeric field, only the digit portion of each alphameric character is moved to the digit portion of the corresponding numeric character. The zone of the rightmost numeric character is set to hexadecimal D (negative) if the zone of the rightmost alphameric character is hexadecimal D (characters J through R). Otherwise, it is set to hexadecimal F.

The MOVE operation is summarized in Figure 303 on page 779.
### MOVE (MOVE)

**Figure 303 (Part 1 of 2). MOVE Operations**

#### Factor 2 Shorter Than Result Field

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a. Alphameric</strong>&lt;br&gt;Before MOVE</td>
<td>123456784&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>PH4SN</td>
<td>1234PH4SN</td>
</tr>
<tr>
<td><strong>b. Alphameric</strong>&lt;br&gt;Before MOVE</td>
<td>123456784&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>PH4SN</td>
<td>123478425</td>
</tr>
<tr>
<td><strong>c. Numeric</strong>&lt;br&gt;Before MOVE</td>
<td>1278425&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>1278425</td>
<td>12178425</td>
</tr>
<tr>
<td><strong>d. Numeric</strong>&lt;br&gt;Before MOVE</td>
<td>1278425&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>1278425</td>
<td>1278425</td>
</tr>
</tbody>
</table>

#### Factor 2 Longer Than Result Field

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a. Alphameric</strong>&lt;br&gt;Before MOVE</td>
<td>56784&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>ACEGH4SN</td>
<td>PH4SN</td>
</tr>
<tr>
<td><strong>b. Alphameric</strong>&lt;br&gt;Before MOVE</td>
<td>56784&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>ACEGH4SN</td>
<td>78425</td>
</tr>
<tr>
<td><strong>c. Numeric</strong>&lt;br&gt;Before MOVE</td>
<td>56748&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>56748</td>
<td>78425</td>
</tr>
<tr>
<td><strong>d. Numeric</strong>&lt;br&gt;Before MOVE</td>
<td>PH4SN&lt;br&gt;After MOVE</td>
</tr>
<tr>
<td>PH4SN</td>
<td>78425</td>
</tr>
</tbody>
</table>

---

---
### Factor 2 and Result Field Same Length

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a. Alphameric</strong></td>
<td>Before MOVE</td>
</tr>
<tr>
<td>PH4SN</td>
<td>After MOVE</td>
</tr>
<tr>
<td><strong>b. Alphameric</strong></td>
<td>Before MOVE</td>
</tr>
<tr>
<td>PH4SN</td>
<td>After MOVE</td>
</tr>
<tr>
<td><strong>c. Numeric</strong></td>
<td>Before MOVE</td>
</tr>
<tr>
<td>78425</td>
<td>After MOVE</td>
</tr>
<tr>
<td><strong>d. Numeric</strong></td>
<td>Before MOVE</td>
</tr>
<tr>
<td>78425</td>
<td>After MOVE</td>
</tr>
</tbody>
</table>

*Figure 303 (Part 2 of 2). MOVE Operations*
MOVEA (Move Array)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>MOVEA</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>7─8</td>
<td>9─17</td>
<td></td>
<td></td>
<td>Result</td>
<td>54─55 56─57 58─59</td>
</tr>
</tbody>
</table>

The MOVEA operation transfers characters from the leftmost positions of factor 2 to the leftmost positions of the result field. Factor 2 and the result field cannot reference the same array even if the array is indexed. Arrays and fields specified by a MOVEA operation can be alphameric or numeric. The length of the move is determined by the shorter of the lengths of factor 2 and the result field. If factor 2 is longer than the result field, the excess rightmost characters of factor 2 are not moved; if the result field is longer than factor 2, the rightmost characters in the result field are unchanged.

The length of factor 2 or the result field is the length of the entire array if the array is not indexed, or the length from the element specified to the end of the array if the array is indexed.

The MOVEA operation makes it possible to:

- Move several contiguous array elements to a single field.
- Move a single field to several contiguous array elements.
- Move contiguous elements of one array to contiguous elements of another array.

Movement of data starts with the first element of an array if the array is not indexed or with the element specified if the array is indexed. The movement of data ends when the last array element is moved or filled or when the number of characters moved equals the length of the shorter field specified by factor 2 and the result field. Therefore, the move could end in the middle of an array element. Because array boundaries are not considered when the MOVEA operation is done, all numeric data is treated as alphameric data. As a result, numeric data is moved without regard for the sign. When you are moving data to a numeric array or numeric field, you should ensure that the result field will contain valid numeric data.

If you use the MOVEA operation with a figurative constant (*BLANK, *BLANKS, *ZERO, or *ZEROS) in factor 2 and an array in the result field, the figurative constant is moved into the array. The figurative constant begins at the array element specified in the result field; it ends at the end of the array.

Figure 304 on page 782 illustrates the use of the MOVEA operation.
MOVEA (MOVE ARRAY)

Example: Alphameric array to alphameric array move. No indexing, different length arrays, same length elements.

Example: Alphameric array to alphameric array move. Index result field.

Figure 304 (Part 1 of 15). MOVEA Operation
Example: Alphameric array to alphameric array move. No indexing, different length array elements.

**Example: Alphameric array to alphameric array move. Index factor 2, different length array elements.**

**Figure 304 (Part 2 of 15). MOVEA Operation**
Example: Alphameric array to alphameric array move. No indexing on array.

```
<table>
<thead>
<tr>
<th>C</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVEA FIELD  A ARRY</td>
</tr>
</tbody>
</table>
```

FIELD A
1 2 3 4 5 6 7

Before MOVEA
9 8 6 5 4 3 2 1 0 A B C

After MOVEA
1 2 3 4 5 6 7 1 0 A B C

Example: Alphameric array to alphameric field move. Variable indexing.

```
<table>
<thead>
<tr>
<th>C</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVEAARRX, N FIELD</td>
</tr>
</tbody>
</table>
```

ARRX
0 1 0 A 0 2 0 B 0 3 0 1 C

Before MOVEA
0 1 0 A

After MOVEA
0 1 0 B

Figure 304 (Part 3 of 15). MOVEA Operation
**Example:** Numeric array to alphabetic array move. No indexing, different length arrays, same length array elements.

```
Before MOVEA

ARRX

   1 2 3 4 5 6 7 8 9 0

   One Element

After MOVEA

ARRY

   0 9 8 7 6 5 4 3 2 1 0 0

   One Element
```

**Example:** Alphabetic array to numeric array. No indexing, different length arrays, same length array elements.

```
Before MOVEA

ARRX

   1 2 3 4 5 6 7 8 9 0

   One Element

After MOVEA

ARRY

   A A B B C C D D E E F F

   One Element
```

Figure 304 (Part 4 of 15). MOVEA Operation
Example: Numeric array to numeric array move. No indexing, different length arrays, same length array elements.

Example: Alphameric array to numeric array move. No indexing, different length arrays, same length array elements.

The first element, AA, is invalid, because AA is not valid numeric characters.

Figure 304 (Part 5 of 15). MOVEA Operation
**Example:** Numeric array to alphameric array move. No indexing, different length array elements.

```
Before MOVEA

3 4 5 6 7 8 9 1 0 1 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 9 2 0 2 1 2 2 3 2 3 3 3 3 3 3 3 3 3 3
```

```
After MOVEA

1 2 3 4 5 6 7 8 9 1 0 1 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 9 2 0 2 1 2 2 3 2 3 3 3 3 3 3 3 3 3 3
```

**Example:** Alphameric array to numeric array move. No indexing, different length array elements.

```
Before MOVEA

AAA BBB CCC DDD
```

```
After MOVEA

000 111 222 333
```

*Figure 304 (Part 6 of 15). MOVEA Operation*
**MOVEA (MOVE ARRAY)**

**Example:** Numeric array to numeric array move. No indexing, different length array elements.

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Requiring Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>And And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Before MOVEA**

**ARRZ**

| 1234567890 |

**After MOVEA**

**ARRZ**

| 1234567890 |

**Example:** Numeric array to numeric array move. No indexing, different length array elements.

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Requiring Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>And And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Before MOVEA**

**ARRZ**

| 01 23 -45 67 -89 |

**After MOVEA**

**ARRZ**

| 01 23 -45 67 -89 |

-45 = F4D5

-89 = FBD9

**.hexadecimal values**

The hexadecimal value of the last element in ARRZ is D9F3F3. The element is invalid because the first digit of a number cannot carry the negative sign.

Figure 304 (Part 7 of 15). MOVEA Operation
Example: Numeric array to alphameric array move. Index result field, same length arrays and array elements.

Example: Alphameric array to numeric array move. Index result field, same length arrays and array elements.

Figure 304 (Part 8 of 15). MOVEA Operation
MOVEA (MOVE ARRAY)

Example: Numeric array to numeric array move. Index result field, same length arrays and array elements.

Example: Numeric array to alphameric array move. Index factor 2, different length array elements.

Figure 304 (Part 9 of 15). MOVEA Operation
**Example:** Alphameric array to numeric array move. Index factor 2, different length array elements.

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Indicator</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Name</th>
<th>Length</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Before MOVEA**

**ARRAY**

```
12 34 56 78 90
```

**After MOVEA**

```
78 06 54 32 10
```

Figure 304 (Part 10 of 15). MOVEA Operation

**Example:** Numeric array to numeric array move. Index factor 2, different length array elements.

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Indicator</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Name</th>
<th>Length</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Before MOVEA**

**ARRAY**

```
12 34 56 78 90
```

**After MOVEA**

```
78 06 54 32 10
```
Example: Numeric field to alphameric array move. No indexing on array.

<table>
<thead>
<tr>
<th>Line</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
<td></td>
<td></td>
<td>MOVEA FIELD</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td></td>
<td></td>
<td>ARRAY</td>
<td></td>
</tr>
</tbody>
</table>

**FIELDA**

<table>
<thead>
<tr>
<th>Before MOVEA</th>
<th>After MOVEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

**ARY**

<table>
<thead>
<tr>
<th>Before MOVEA</th>
<th>After MOVEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 8 6 5 4 3 2 1 0</td>
<td>1 2 3 4 5 6 7 1 0</td>
</tr>
</tbody>
</table>

Example: Alphameric field to numeric array move. No indexing on array.

<table>
<thead>
<tr>
<th>Line</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
<td></td>
<td></td>
<td>MOVEA FIELD</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td></td>
<td></td>
<td>ARRAY</td>
<td></td>
</tr>
</tbody>
</table>

**FIELDA**

<table>
<thead>
<tr>
<th>Before MOVEA</th>
<th>After MOVEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

**ARY**

<table>
<thead>
<tr>
<th>Before MOVEA</th>
<th>After MOVEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 8 6 5 4 3 2 1 0</td>
<td>1 2 3 4 5 6 7 1 0</td>
</tr>
</tbody>
</table>

Figure 304 (Part 11 of 15). MOVEA Operation
**Example:** Numeric field to numeric array move. No indexing on array.

<table>
<thead>
<tr>
<th>C</th>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MOVEA FIELD A ARRAY</td>
</tr>
</tbody>
</table>
Example: Alphabetic field to numeric array move. No indexing on array.

Example: Numeric array to alphabetic field move. Variable indexing.

Note: \( N = 3 \)

Figure 304 (Part 13 of 15). MOVEA Operation
Example: Alphameric array to numeric field move. Variable indexing.

<table>
<thead>
<tr>
<th>Line</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
MOVEA ARRX, N FIELD
```

Example: Numeric array to numeric field move. Variable indexing.

<table>
<thead>
<tr>
<th>Line</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
MOVEA ARRX, N FIELD
```

**Note:** N = 3

---

Figure 304 (Part 14 of 15). MOVEA Operation
Example: Numeric array to numeric field move. Variable indexing.

```
Example:
```

```
MOVEAA, N FIELD
```

```
ARRX

01 02 03 -04 05 06

Before MOVEA

01 02

One Element

After MOVEA

01 02 03 -04 05 06

-04

Note: N = 3

The last digit in the field carries the minus sign.
```

Example: Alphameric array to numeric field move. Variable indexing.

```
Example:
```

```
MOVEAA, N FIELD
```

```
ARRX

01 02 03 AA 05 06

Before MOVEA

01 02

One Element

After MOVEA

03 AA 05 06

Note: N = 3

The field is invalid because AA are not valid numeric characters.
```

Figure 304 (Part 15 of 15). MOVEA Operation
The MOVEL operation transfers characters from factor 2 to the leftmost positions in the result field. Moving begins with the leftmost character in factor 2. When a numeric field is moved into an alphameric field, both digit and zone portions of the rightmost character are transferred if that character is moved.

A summary of the rules for MOVEL operations for three conditions based on field lengths is as follows:

1. Factor 2 the same length as the result field:
   a. If factor 2 and the result field are numeric, the sign is moved with the rightmost position.
   b. If factor 2 is numeric and the result field is alphameric, the sign is moved with the rightmost position.
   c. If factor 2 is alphameric and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a D (characters J through R). However, if the zone from the rightmost position of factor 2 is not a D, a positive zone is moved into the rightmost position of the result field. Digit portions are converted to their corresponding numeric characters.
   d. If factor 2 and the result field are alphameric, all characters are moved.

2. Factor 2 longer than the result field:
   a. If factor 2 and the result field are numeric, the sign from the rightmost position of factor 2 is moved into the rightmost position of the result field.
   b. If factor 2 is numeric and the result field is alphameric, the result field contains only numeric characters.
   c. If factor 2 is alphameric and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a D (characters J through R). However, if the zone from the rightmost position of factor 2 is not a D, a positive zone is moved into the rightmost position of the result field. Other result field positions contain only numeric characters.
   d. If factor 2 and the result field are alphameric, only the number of characters needed to fill the result field are moved.
3. Factor 2 shorter than the result field:
   a. If factor 2 is either numeric or alphameric and the result field is numeric, the digit portion of factor 2 replaces the contents of the leftmost position of the result field. The sign in the rightmost position of the result field is not changed.
   b. If factor 2 is either numeric or alphameric and the result field is alphameric, the characters in factor 2 replace the equivalent number of the leftmost position in the result field. No change is made in the zone of the rightmost position of the result field.

The MOVEL operation is summarized in Figure 305.

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Numeric</td>
<td>Before MOVEL 5</td>
</tr>
<tr>
<td>b. Numeric</td>
<td>Before MOVEL A</td>
</tr>
<tr>
<td>c. Alphameric</td>
<td>Before MOVEL 5</td>
</tr>
<tr>
<td>d. Alphameric</td>
<td>Before MOVEL A</td>
</tr>
</tbody>
</table>

Figure 305 (Part 1 of 2). MOVEL Operations
### Factor 2 Longer Than Result Field

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Numeric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>000008425</td>
<td>56784</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>08000</td>
</tr>
<tr>
<td>b. Numeric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>903178425</td>
<td>AKT4D</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>90317</td>
</tr>
<tr>
<td>c. Alphameric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>BRCXH45N</td>
<td>56784</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>29637</td>
</tr>
<tr>
<td>d. Alphameric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>BRCXH45N</td>
<td>AKT4D</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>BRCX</td>
</tr>
</tbody>
</table>

The arrow ↓ between numbers indicates a decimal point.

### Factor 2 Shorter Than Result Field

<table>
<thead>
<tr>
<th>Factor 2</th>
<th>Result Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Numeric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>78425</td>
<td>130943210</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>784253210</td>
</tr>
<tr>
<td>Alphameric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>CPT5N</td>
<td>130943210</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>37353210</td>
</tr>
<tr>
<td>b. Numeric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>78425</td>
<td>BRCXH45A</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>7842NH45A</td>
</tr>
<tr>
<td>Alphameric</td>
<td>Before MOVEL</td>
</tr>
<tr>
<td>CPT5N</td>
<td>BRCXH45A</td>
</tr>
<tr>
<td>After MOVEL</td>
<td>CPT5NH45A</td>
</tr>
</tbody>
</table>

The arrow ↓ between numbers indicates a decimal point.

*Figure 305 (Part 2 of 2). MOVEL Operations*
## MULT (Multiply)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>Optional</td>
<td>Optional</td>
<td>MULT</td>
<td>Required</td>
<td>54–55 56–57 58–59</td>
</tr>
</tbody>
</table>

Factor 1 is multiplied by factor 2, and the product is placed in the result field. Factor 1 and factor 2 are not changed. If factor 1 is not present, the result field is multiplied by factor 2, and the product is placed in the result field. Be sure that the result field is large enough to hold the product. To determine the minimum length of the result field, use this rule: the length of the result field equals the length of factor 1 plus the length of factor 2.
MVR (Move Remainder)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-8</td>
<td>9-17</td>
<td>Blank</td>
<td>MVR</td>
<td>Blank</td>
<td>Required</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>MVR</td>
<td>Blank</td>
<td>Required</td>
</tr>
</tbody>
</table>

The MVR operation moves the remainder from the previous divide operation to a separate field named as the result field. Factor 1 and factor 2 must not be used. This operation must immediately follow the divide operation.

The maximum length of the remainder (including decimal positions) is 15. The number of significant decimal positions is the greater of:

- The number of decimal positions in factor 1 of the previous divide operation.
- The sum of the decimal positions in factor 2 and the result field of the previous divide operation.

The maximum number of whole number positions in the remainder is equal to the whole number positions in factor 2 of the previous divide operation. Figure 306 shows the specification for a move remainder operation.

```
  *.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
 CC1N01N02N03Factor1+++OpcedeFactor2+++ResultLenDHHiLoEqComments++++++*
  C 14N02 FIELD A DIV FIELD B SAVE 60
  C 14N02 MVR STORE 60
```

*Figure 306. Move Remainder Operation*
The NEXT operation code forces the next input to the program to come from the device specified in factor 1. If NEXT is specified more than once between input operations, only the last operation is processed. The NEXT operation code can be used only for a WORKSTN file.

**Note:** For WORKSTN files, a device can be either a display station or an ICF session.

To use this operation, enter NEXT in columns 28 through 32. In factor 1, enter the name of a two-character field that contains the device identification or a two-character alphabetic literal that is the device identification. In factor 2, enter the name of the WORKSTN file for which the operation is requested.

An indicator can be specified in columns 56 and 57. This indicator is set on if an exception or error occurs on the NEXT operation. If the INFSR subroutine is specified and columns 56 and 57 do not contain an indicator, the subroutine automatically receives control when an exception/error occurs. (For more information on the INFSR subroutine, see “Coding the INFSR Subroutine” on page 198 in Chapter 7, “Using a WORKSTN File.”) If the INFSR subroutine is not specified and columns 56 and 57 do not contain an indicator, the program halts when an exception or error occurs.

For more information on the NEXT operation code, see Chapter 7, “Using a WORKSTN File.”
The declarative PARM operation defines the parameters that compose a parameter list (PLIST).

PARM operations can appear anywhere in calculations, but they must immediately follow the PLIST or CALL operation, which they refer to. PARM statements must be in the order expected by the called program. One PARM statement, or as many as 255 PARM statements, can follow a PLIST or CALL.

The PARM operation can be specified anywhere within calculations, including total operations. The control level entry in positions 7 and 8 can be blank, or can contain an L1 through L9 indicator, the LR indicator, or the L0 entry to group the statement in the appropriate section of the program.

Factor 1 and factor 2 entries are optional. If specified, the entries must be the same type (character or numeric) as specified in the result field. A literal cannot be specified in factor 1. You can specify an indicator on factor 1 or factor 2 using the convention *INxx, where xx is the indicator.

Remember the following when specifying a PARM statement:

- If there are more parameters in the calling program than in the called program, the called program will fail.
- If there are more parameters in the called program than in the calling program, an error occurs when an unresolved parameter is used, and the called program will fail.
- Fields specified as parameters in an *ENTRY PLIST can be used at first-page (1P) time.
- Because the program accesses the parameter fields by address, not field name, the calling and called parameters do not have to use the same field names for fields that are passed. However, the attributes of the corresponding parameter fields in the calling and called programs must be the same. If they are not, undesirable results may occur.
- The result field must contain the name of a field, data structure, or array that is the parameter. Also, the result field of a non-*ENTRY PLIST can contain an array element. The result field can be numeric or character. The result field cannot contain a label, a literal, a table name. In addition, an array element, the name of a compile-time array, and a file information data structure (INFDS) are not allowed in the result field of the PARM specified for an *ENTRY PLIST. A field name can be specified only once in an *ENTRY PLIST.
- If an array is specified in the result field, the area defined for the array is passed to the called program.
• There is only one storage location for each parameter field. It is in the calling program. What is passed to the called program on a PARM operation is the address of the storage location of the result field. If the called program changes the value of a parameter, it changes the data at that storage location. Therefore, when control returns to the calling program, the value of the parameter in the calling program (that is, the result field) has changed. Even if the called program ends in error after it changes the value of a parameter, the changed value exists in the calling program. To preserve for later use the information passed to the called program, specify in factor 2 the name of the field that contains the information you want to pass to the called program. Factor 2 is copied into the result field, and the storage address of the result field is passed to the called program.

• Following occurs when a CALL operation is processed:

1. In the calling program, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation.

2. In the called program, after it receives control and after any normal program initialization, the contents of the result field of a PARM operation are copied into the factor 1 field (receiver field) of the same PARM operation.

3. In the called program, when control is returned to the calling program, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation. This move does not occur if the called program ends abnormally.

4. Upon return to the calling program, the contents of the result field of a PARM operation in the calling program are copied into the factor 1 field (receiver field) of the same PARM operation. This move does not occur if the called program ends abnormally or if an error occurs on the CALL operation.

**Note:** If the receiver field is numeric, the value of the field to be placed in the receiver field is added to a field of zeros and the sum is placed in the receiver field. If the receiver field is character, the field placed in the receiver field is moved (left-adjusted) into the receiver field, and the receiver field is padded with blanks.
PLIST (Identify a Parameter List)

The declarative PLIST operation defines a unique symbolic name for a parameter list specified in a CALL operation, or defines the entry parameter list (*ENTRY PLIST) in a called program.

A PLIST operation can be specified anywhere within operations, including within total operations and between subroutines.

The control level entry in positions 7 and 8 can be blank, or can contain an L1 through L9 indicator, the LR indicator, or the L0 entry to group the statement in the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 must contain the name of the parameter list. If the parameter list is the entry parameter list of a called program, factor 1 must contain *ENTRY. Only one *ENTRY parameter list can occur in a program. A parameter list is ended when an operation other than PARM is found.

Rules for Specifying PLIST
Remember the following when specifying a PLIST statement:

- If PLIST is specified, it must immediately be followed by the PARM(s) that apply to it. If no PARM statements follow a PLIST statement, the PLIST statement is invalid.
- Multiple PLIST statements can appear in a program.
- Only one *ENTRY PLIST can occur in a program.
The POST operation allows you to retrieve status information for a specified display station that is using a WORKSTN file. The status information is placed in the INFDS data structure that was specified in the result field. The program must contain the INFDS data structure for the WORKSTN file to use POST.

Factor 1 must contain a variable or an alphameric literal that identifies the display station whose status is being requested. The result field contains the name of the INFDS data structure in which this information is posted. Columns 56 and 57 can specify an indicator that is set on if an error occurs on the POST operation. An error occurs if the specified work station identifier is not using the file for which the INFDS data structure is specified.

If columns 56 and 57 do not specify an indicator but the program contains the INFSR subroutine, the subroutine automatically receives control when an error occurs. If the INFSR subroutine is not present and columns 56 and 57 do not contain an indicator, the program halts when an exception or error occurs. If the display station is not using the WORKSTN file, the device will be not found, and an error will occur on POST. (For more information on the INFSR subroutine, see “Coding the INFSR Subroutine” on page 198 in Chapter 7, “Using a WORKSTN File.”)

Columns 33 through 42, 49 through 55, and 58 and 59 must be blank for a POST operation.

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>POST</td>
<td>Blank</td>
<td>Required</td>
</tr>
<tr>
<td>7–8</td>
<td>9–17</td>
<td></td>
<td></td>
<td>54–55</td>
<td>56–57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>58–59</td>
<td>Opt: ER</td>
</tr>
</tbody>
</table>

The POST operation allows you to retrieve status information for a specified display station that is using a WORKSTN file. The status information is placed in the INFDS data structure that was specified in the result field. The program must contain the INFDS data structure for the WORKSTN file to use POST.

Factor 1 must contain a variable or an alphameric literal that identifies the display station whose status is being requested. The result field contains the name of the INFDS data structure in which this information is posted. Columns 56 and 57 can specify an indicator that is set on if an error occurs on the POST operation. An error occurs if the specified work station identifier is not using the file for which the INFDS data structure is specified.

If columns 56 and 57 do not specify an indicator but the program contains the INFSR subroutine, the subroutine automatically receives control when an error occurs. If the INFSR subroutine is not present and columns 56 and 57 do not contain an indicator, the program halts when an exception or error occurs. If the display station is not using the WORKSTN file, the device will be not found, and an error will occur on POST. (For more information on the INFSR subroutine, see “Coding the INFSR Subroutine” on page 198 in Chapter 7, “Using a WORKSTN File.”)

Columns 33 through 42, 49 through 55, and 58 and 59 must be blank for a POST operation.
READ (Read)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Result</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>READ</td>
<td>Required</td>
<td>Blank 54–55</td>
</tr>
<tr>
<td>58–59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Blk Opt: ER 56–57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Blk R: EOF 58–59</td>
</tr>
</tbody>
</table>

The READ operation calls for immediate input from a demand or full-procedural file during the calculation phase of the program cycle. This operation differs from the FORCE operation because FORCE calls for certain input on the next program cycle, not on the present one.

The READ operation code must appear in columns 28 through 32. Factor 2 contains the name of the file from which a record should be read immediately. Full-procedural files must contain an indicator in columns 58 and 59. This indicator turns on when an end-of-file condition is reached for the demand file or the full-procedural file or for each READ operation after an end-of-file condition is reached. If the file in factor 2 is a demand file, the indicator in columns 58 and 59 does not turn off before the READ operation is processed. If the file is a full-procedural file, the indicator does turn off. If columns 58 and 59 are blank, a halt occurs on an end-of-file condition and on subsequent READ operations after the end-of-file condition is reached. Indicators can be specified in columns 7 through 17.

Note: In a READ operation, input occurs during calculation time and record-identifying indicators do not turn off.

An indicator can be specified in columns 56 and 57 if the READ operation is issued to a WORKSTN file. This indicator turns on if an exception condition occurs (that is, if the operator pressed one of the command keys: Roll Up, Roll Down, Clear, Print, Record Backspace, or Help) or if an input error occurs. If columns 56 and 57 do not contain an indicator and either of these conditions occurs, the program halts unless the INFSR subroutine is specified. If the INFSR subroutine is specified, the subroutine automatically receives control, and an exception or error occurs. (For more information on the INFSR subroutine, see “Coding the INFSR Subroutine” on page 198 in Chapter 7, “Using a WORKSTN File.”)

An AS/400 RPG/400 program may receive modified data back from a workstation after a roll key is entered on a READ operation due to the special handling of status codes 1121 - 1126 in the RPG Exception/Error Handling routine.

The following columns must remain blank for a READ operation: columns 18 through 27 (factor 1), columns 43 through 48 (result field), columns 49 through 51 (field length), column 52 (decimal positions), column 53 (half-adjust), and columns 54 and 55 (resulting indicators).

If a READ operation is not successful, you must reposition the file by using either a SETLL or a CHAIN operation. If the file is not repositioned, all following READ, READE, and READP operations will fail.
The following files can appear as factor 2 in a READ operation:

- Sequential DISK files processed consecutively and specified as input or update files
- Direct DISK files processed consecutively as input or update files
- Indexed DISK files processed sequentially by key field and specified as input or update files
- Indexed DISK files processed sequentially by limits and specified as input or update files
- Sequential, direct, and indexed DISK files processed randomly and sequentially as input or update files
- WORKSTN files
- SPECIAL files
- CONSOLE files
- BSCA files.
The READE operation retrieves the next sequential record from an indexed full-procedural DISK file (identified by an F in position 16 of the file description specifications) if the key field of the record matches factor 1.

Factor 1 identifies the record to be retrieved. It is required, and it can be a field name, an array or table element, or a literal. Factor 1 must have the same field type and length as the key field of the file being read.

Factor 2 must contain the name of the file read as specified on the file description specification. This file must be full-procedural.

A resulting indicator must be specified in positions 58 and 59 of the calculation specification. This indicator turns on if the next sequential record does not have a key field equal to factor 1, or if end of file occurs. If this indicator turns on, a record is not read by the program.

If a READE operation is not successful, you must reposition the file by using either a SETLL or a CHAIN operation. If the file is not repositioned, all following READ, READE, and READP operations will fail.
The READP operation reads the prior record from a full-procedural DISK file (identified by an F in position 16 of the file description specifications). For example, if record X was just read using the READ operation code, READP reads the record prior to record X.

Factor 2 must contain the name of the file read as specified on the file description specifications. This file must be full-procedural.

Positions 58 and 59 must contain an indicator. This indicator turns on if no prior record exists in the file (beginning-of-file condition). It turns off before each READP operation. If a READP operation is not successful, you must reposition the file by using either a SETLL or CHAIN operation. If the file is not repositioned, all following READ, READE, and READP operations will fail.
The REL operation releases the device specified in factor 1 from the program. Either a requesting or an acquired device can be released with the REL operation code. The specified device is released when the REL operation occurs during the calculations unless the device is the requester of a single requester terminal (SRT) program. If the device specified in factor 1 is the requester of a SRT program, the device is released at end of job, not when the operation code is read in the calculations. (If the device is a display station, it is no longer available to the program, but it is available for system log messages.)

Factor 2 must contain the file name for the WORKSTN file.

If an exception or error occurs on the attempt to release the device, the indicator specified in columns 56 and 57 turns on. If no indicator is specified, the program halts unless the INFSR subroutine is specified in the program. If the INFSR subroutine is specified, the INFSR subroutine automatically receives control when an exception or error occurs and no indicator is specified in columns 56 and 57.

When all devices are released from a primary WORKSTN file, the file goes to end of file, and RPG turns on the LR indicator. If the program containing the primary file is a never-ending program, the program continues until the ENDJOB CL command or the ENDSBS (End Subsystem) CL command is entered.

When all devices are released from a demand WORKSTN file and the program is not a never-ending program, the first READ operation after the last REL operation causes the READ end-of-file indicator to turn on (columns 58 and 59). You can then turn on the LR indicator, unless the LR indicator was specified as the end-of-file indicator. If the program containing the demand WORKSTN file is a never-ending program, the end-of-file indicator turns on when the user enters the ENDJOB or ENDSBS CL command. You can then turn on the LR indicator, unless the LR indicator was specified as the end-of-file indicator.

If RESTORE-NO is specified on the OCL WORKSTN statement, a display format from the program may appear on the screen after the display station has been released. If RESTORE-YES is specified on the WORKSTN statement, the command display appears on the screen immediately when the display station is released.

For more information on the REL operation code, see Chapter 7, “Using a WORKSTN File.”
The RETRN operation causes a return to the caller. It works as follows:

1. RPG checks the halt indicators. If a halt indicator is on, the program ends abnormally. All open files are closed, an error return code is set to indicate to the caller that the program has ended abnormally, and control returns to the caller.

2. If no halt indicators are on, RPG checks the LR indicator. If it is on, the program ends normally.

3. If no halt indicators are on, and if LR is not on, the program returns to the caller.
The RLABL operation allows the subroutine specified in an EXIT operation to refer to a field, data structure, array, table, or indicator defined in the RPG program. RLABL operations must be specified immediately after the EXIT operation that refers to the subroutine using the field, data structure, array, table, or indicator in the RLABL specification.

The number of RLABL operations following the EXIT operation must correspond to the number of items referred to in the subroutine. The order of the RLABL operations must correspond to the order of references in the subroutine.

The entries used with RLABL on the calculation specifications are as follows:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation (28-32)</td>
<td>RLABL</td>
</tr>
<tr>
<td>Result field 43-48</td>
<td>Field name, data structure name, array name, table name, or indicator (INxx, where xx is the indicator)</td>
</tr>
<tr>
<td>Field length (49-51)</td>
<td>Length of field (optional)</td>
</tr>
<tr>
<td>Decimal positions (52)</td>
<td>Decimal indication (optional)</td>
</tr>
</tbody>
</table>

The control level entry in positions 7 and 8 can be blank, or can contain an L1 through L9 indicator, the LR indicator, or the L0 entry to group the statement in the appropriate section of the program.

Only RLABL operations specifying a field, array, or table name can have entries for field length (columns 49 through 51) and decimal positions (column 52).

The following columns must be blank for an RLABL operation: columns 9 through 17 (indicators), columns 18 through 27 (factor 1), columns 33 through 42 (factor 2), column 53 (half adjust), and columns 54 through 59 (resulting indicators).

A name defined by a TAG, BEGSR, or ENDSR specification cannot be used in an RLABL specification.

The parameters defined on the RLABL operations are passed to the subroutine by address. The RPG compiler also creates an extra parameter at the end of the parameter list that contains the attributes of each of the parameters. Because of the attribute parameter, the number of parameters in the called program should be one more than the number of RLABLs in the calling program. For each parameter, the following attributes are generated:

- A 1-byte character field: ‘C’ When the parameter is alphabetic; ‘Z’ When the parameter is numeric.
RLABL (RPG LABEL)

- A 4-byte zoned field: Length of the parameter in bytes.
- A 2-byte zoned field: Number of decimal digits if the parameter is numeric.
- Four byte zoned field: Number of elements if the parameter is an array.

Therefore, you need to declare an array of length equal to the number of variables that are declared using the RLABL opcode. Each element in the array must have the following structure:

<table>
<thead>
<tr>
<th>Type of element (C or Z)</th>
<th>Length of parameter</th>
<th>Number of decimal digits</th>
<th>Number of array elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>ZND(4,0)</td>
<td>ZND(2,0)</td>
<td>ZND(4,0)</td>
</tr>
</tbody>
</table>

The array itself will be based on the extra parameter that is declared in the subroutine, because RPG passes the parameters to the subroutine by address.

*..1...+..2...+..3...+..4...+..5...+..6...+..7...*  
CC1NO1N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++  
C*  
C* RPG passes control to the subroutine SUBRA.  
C*  
C* The subroutine SUBRA refers to the field HERE in the RPG program  
C* and returns control to the RPG program.  
C*  
C*  
C EXIT SUBRA  
C RLABL HERE 1  
C*  
C* The subroutine returns control to the RPG program, which processes  
C* a compare operation to determine which character was placed in  
C* the field HERE.  
C*  
C 'A' COMP HERE 082257

*Figure 307. RPG Linkage to a Subroutine*
Message-Retrieving Subroutine (SUBR23)

The message-retrieving subroutine (SUBR23) allows you to retrieve messages from a user message member. After the message has been retrieved, it can be modified and written to an output file.

Note: Previously, # signs were ignored by the compiler, but as of the present release, # signs included in the message text will be returned to the program as part of the text message.

If you want the # signs to be excluded from the returned message, use level 7 for first level text messages, and level 8 for second level retrieval.

Linkage to SUBR23 is by the EXIT operation code, and input parameters are passed to SUBR23 by RLABL operation codes. To use SUBR23, specify EXIT in columns 28 to 31 and SUBR23 in columns 33 to 38. Four RLABL operation codes must be specified after the EXIT operation with the following result-field entries:

<table>
<thead>
<tr>
<th>Table 21 (Page 1 of 2). Result-Field Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result Field</td>
</tr>
<tr>
<td>MIC number</td>
</tr>
</tbody>
</table>
The text area, which is specified by the second RLABL operation, is blanked before each attempt to retrieve a message; therefore, a blank text area is returned to the user program when the return code value is 2 or greater. A total of 225 positions in the text area are blanked unless the text area is less than 225 characters in length.

**Note:** User messages must be prefixed by ‘USR’.

<table>
<thead>
<tr>
<th>Result Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text area</td>
<td>Name of the alphameric field or data structure into which the message text is read. The maximum length of a level-1 message is 75 characters and of a level-2 message is 225 characters.</td>
</tr>
</tbody>
</table>

| Level | Name of a one-digit numeric field that designates the user message member level. A value of 1 in this field indicates a message level of 1; a value of 2 indicates a message level of 2. |

<table>
<thead>
<tr>
<th>Rcode</th>
<th>Name of a one-digit numeric field that contains the return codes. The return codes and their meanings are as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Code</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td>Message was successfully retrieved with no truncation.</td>
</tr>
<tr>
<td>1</td>
<td>Message was successfully retrieved; however, 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 invalid.</td>
</tr>
<tr>
<td>4</td>
<td>An invalid MIC value was diagnosed.</td>
</tr>
<tr>
<td>5</td>
<td>Message member was not found or message text length exceeds the level-1 maximum.</td>
</tr>
</tbody>
</table>
## SET (Set)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Field</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>SET</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54–55 56–57 58–59</td>
</tr>
</tbody>
</table>

The SET operation can be used only with input files assigned to the device KEYBORD, or with a CONSOLE file if the result field contains the word ERASE. Both KEYBORD and CONSOLE are specified in columns 40 through 46 of the file description specifications. All SET operations are directed to the display station that loaded the program.

The SET operation allows any one or any combination of the following:

- Pressing of function keys identified in columns 54 through 59
- Displaying on the display screen the field, literal, or array or table element specified in factor 1
- Displaying user messages (from USER1 message member) 0001 to 0099 when numbers 01 to 99, respectively, are specified in the nn portion of the SETnn and KEYnn operation codes
- blanking the buffer for a CONSOLE file if ERASE is specified in the result field of the SET operation.

Factor 1 can contain the constant, literal, field name, or table or array element to be displayed.

Factor 2 must contain the name of the CONSOLE file if ERASE is coded in the result field. For all other SET operations, leave factor 2 and the result field blank. See “Calculation Specifications for a SET Operation” on page 304 in Chapter 10, “Using a CONSOLE, KEYBORD, or CRT File” for more information on coding the SET operation.
# SETLL (Set Lower Limits Operation)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>factor 1</td>
<td>operation</td>
<td>factor 2</td>
<td>blank</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>required</td>
<td>SETLL</td>
<td>required</td>
<td>blank</td>
</tr>
</tbody>
</table>

The SETLL operation allows the lower limits for indexed full-procedural and indexed demand files being processed sequentially within limits set during calculations. Factor 1 must contain a field name or literal representing the value of the lower limit being set. The length of the field or literal must be equal to the total length of the key field specified for the file named in factor 2. Factor 2 must contain the name of the file for which the lower limit is set.

Figure 309 shows an example of SETLL coding.

```
* 1 2 3 4 5 6 7
filenameIPEAFBlenRLenLKLAI0vKlocEDevice+....Exit++....A....U+. *
FLMTFILE ID F 512 256L 8AI 1 DISK
* 1 2 3 4 5 6 7
CC1NSz
C FIELDA SETLLMTFILE
C READ LMTFILE 10
```

Figure 309. SETLL Operation Code

FIELDA is defined on input specifications as an eight-position alphabetic field.

**Notes:**

1. When a lower limit is specified by SETLL, the end-of-file indicator specified for the READ operation to the file being processed is not set off by the RPG cycle.
2. If a READ operation is processed to the file prior to a SETLL operation, the record with the lowest key field in the file is fetched.
**SETOF (Set Off)**

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Field</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>SETOF</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54–55</td>
</tr>
</tbody>
</table>

The SETOF operation turns off any indicators specified in columns 54 through 59. At least one resulting indicator must be specified in columns 54 through 59.

---

**SETON (Set On)**

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Factor 1</td>
<td>Operation</td>
<td>Factor 2</td>
<td>Field</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>SETON</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54–55</td>
</tr>
</tbody>
</table>

The SETON operation turns on any indicators specified in columns 54 through 59. At least one resulting indicator must be specified in columns 54 through 59.
The SHTDN operation turns on the resulting indicator specified in columns 54 and 55 if the system operator has requested shutdown. The indicator can then be used to condition ending the program in an orderly manner, such as printing some partial totals and going to the normal end of job.

Columns 28 through 32 must contain SHTDN, and columns 54 and 55 must contain one of the following valid indicators: 01 through 99, L1 through L9, U1 through U8, H1 through H9, or LR.
SORTA (Sort an Array)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7─8</td>
<td>9─17</td>
<td></td>
<td></td>
<td></td>
<td>54─55 56─57 58─59</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>SORTA</td>
<td>Required</td>
<td>Blank Blank Blank</td>
</tr>
</tbody>
</table>

The SORTA operation allows you to sequence the elements of an array while a program is running. You can ensure that the elements of the array are in the proper sequence for a LOKUP operation by processing a SORTA operation.

The array specified in factor 2 is sorted into the sequence specified in the extension specifications for the array. If no sequence is specified, the array is sorted into ascending sequence. The standard EBCDIC collating sequence is used for the SORTA operation. If an alternative collating sequence has been defined, it is not used. Related arrays, if existing, are not sorted. Only the array specified in factor 2 is sorted.

For examples of the SORTA operation, see Figure 310.

**Note:** Columns 18 through 27 (factor 1) and 43 through 59 (result field, half-adjust, and resulting indicators) must be blank if a SORTA operation is specified.

In Figure 310, the array ARY is sorted into ascending order because no entry is specified for sequence (columns 45) in the extension specifications. ARYA is sorted into ascending order because column 45 of the extension specifications contains A; ARYD is sorted into descending order because column 45 contains D.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
E....FromFileToFile++ArrnamEntParrLenPDSAItnamLenPDSComments+++++++++
E   ARY   2   10   3
E   ARYA  15   5   A
E   ARYD  10   4   0D
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++ C*
C* Sort ARY and ARYA into ascending order.
C*
C   SORTAARY
C   SORTAARYA
C*
C* Sort ARYD into descending order.
C*
C   SORTAARYD
```

*Figure 310. Example of SORTA Operation Code*
SQRT (Square Root)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Blank</td>
<td>SQRT</td>
<td>Required</td>
<td>Blank</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54–55</td>
<td>56–57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>58–59</td>
<td></td>
</tr>
</tbody>
</table>

The SQRT operation derives the square root of the field named in factor 2. The square root of factor 2 is placed in the result field. Factor 1 is not used. An entire array can be used in a SQRT operation if factor 2 and the result field contain array names.

The number of decimal places in the result field can be either less than or greater than the number of decimal places in factor 2. However, the result field should not have less than half the number of decimal places in factor 2. The result of a SQRT operation is always half-adjusted.

If the value of the factor 2 field is negative, the job halts. The person using the display station can continue processing by responding to the error message. When processing continues, the result field is set to zero.
### SUB (Subtract)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Required</td>
<td>Required</td>
<td>54–55 56–57 58–59</td>
<td></td>
</tr>
</tbody>
</table>

Factor 2 is subtracted from factor 1. The difference is placed in the result field. Factor 1 and factor 2 are not changed by the operation. Subtracting a field from itself is a method of setting the result field to zeros. If factor 1 is not present, factor 2 is subtracted from the result field, and the difference is placed in the result field.
TAG (Tag)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Blank</td>
<td>Required</td>
<td>TAG</td>
<td>Blank</td>
<td>Blank</td>
</tr>
</tbody>
</table>

The TAG operation names the operation to which the program branches in the GOTO operation. If the TAG appears within a subroutine, the associated GOTO must appear within the same subroutine.

Factor 1 contains the label, which must begin in column 18. The same label cannot be used for more than one TAG operation (or elsewhere as a subroutine name or ENDSR label).

Factor 2 and the result field are not used. No indicators can be entered in columns 9 through 17 for a TAG operation.

**Note:** When the label for a TAG statement is used as a breakpoint of the ADDBK (ADD BREAK POINT) command, you must precede the label with T_. Otherwise, the message MSGCPF1920 will be issued.

See Figure 299 on page 769 and Figure 300 on page 770 for examples of the TAG operation.
The TESTB operation compares the bits identified in factor 2 with the corresponding bits in the field named as the result field. The resultant field must be a one-position character field. Resulting indicators in columns 54 through 59 reflect the status of the result-field bits. Factor 2 is always a source of bits for the result field. The result field is the field in which corresponding bits are compared with the bits specified in factor 2.

Factor 2 can contain:

- Bit numbers 0-7: From one to eight bits can be tested per operation. The bits tested are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in column 33. For example, to test bits 0, 2, and 5, enter '025' in factor 2.

- Field name: The name of a one-position alphabetic field, array element, or table element can be specified in factor 2. In this case, the bits that are on in the field, array element, or table element are tested in the result field; bits that are off are not tested.

See Figure 311 on page 826 for a summary of TESTB operations.

Indicators assigned in columns 54 through 59 reflect the status of the result-field bits. At least one indicator must be assigned, and as many as three can be assigned for one operation. Two indicators can be the same for a TESTB operation, but not three. For TESTB operations, the resulting indicators turn on as follows:

**Columns 54-55:**

An indicator in these columns turns on if each bit specified in factor 2 or each bit that is on in the factor 2 field is off in the result field.

**Columns 56-57:**

An indicator in these columns turns on if the bits specified in factor 2 or the bits that are on in the factor 2 field are of mixed status (some on, some off) in the result field.

**Columns 58-59:**

An indicator in these columns turns on if each bit specified in factor 2 or each bit that is on in the factor 2 field is on in the result field.

**Note:** If the field in factor 2 has no bits on, then this indicator turns on.

The operation code TESTB must appear in columns 28 through 32. Conditioning indicators can be used in columns 7 through 17. At least one resulting indicator must be assigned in columns 54 through 59. As many as three resulting indicators
can be assigned, but not more than two can be the same. Factor 1, decimal positions, and the half-adjust columns must be blank.

\*.. 1 \*+... 2 \*+... 3 \*+... 4 \*+... 5 \*+... 6 \*+... 7 \*+...
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++C*
C* The following TESTB operation compares bits 0 and 7 with corresponding
C* bits in the field named BITS. If bits 0 and 7 are off in the field
C* named BITS, indicator 20 turns on. If bits 0 and 7 are of mixed
C* status (one on, one off) in the field named BITS, indicator 21 turns
C* on. If bits 0 and 7 are on in the field named BITS, indicator 22
C* turns on.
C*
C  TESTB'07'        BITS       202122
C*
C* The following operation compares the bits that are on in the field
C* named ALPHA with corresponding bits in the field named BITS. If the
C* bits that are on in the field named ALPHA are off in the field named
C* BITS, indicator 20 turns on. If the bits that are on in the field
C* named ALPHA are of mixed status (some on, some off) in the field named
C* BITS, indicator 21 turns on. If the bits that are on in the field
C* named ALPHA are on in the field named BITS, indicator 22 turns on.
C*
C  TESTBALPHA       BITS       202122
C*
C* The following operations use a one-position array element either as a
C* source of bits or as a result field, or both. In the first
C* operation, the bits that are on in the field named ALPHA are compared
C* to corresponding bits in the array element ARR,NX. For example,
C* assume that bits 1 and 4 are on in the field named ALPHA. If bits
C* 1 and 4 are off in array element ARR,NX, indicator 20 turns on. If
C* bits 1 and 4 are of mixed status (one on, one off) in array element
C* ARR,NX, indicator 21 turns on. If bits 1 and 4 are on in array
C* element ARR,NX, indicator 22 turns on.
C*
C  TESTBALPHA       ARR,NX      202122
C*
C  TESTB'24'        ARE,12      202122
C*
C  TESTBARE,12      ARR,NX      202122

Figure 311. Summary of TESTB Operations
### TESTZ (Test Zone)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–8</td>
<td>9–17</td>
<td>Blank</td>
<td>TESTZ</td>
<td>Blank</td>
<td>Required</td>
</tr>
</tbody>
</table>

The TESTZ operation tests the zone of the leftmost character in the result field. The result field must be alphameric because this operation can be done only on alphameric characters. At least one resulting indicator must be specified in columns 54 through 59. Resulting indicators turn on according to the results of the test. The characters &; A through I, and any other character with the same zone as the character A turn the plus indicator on. The characters - (minus), J through R, and any other character with the same zone as the character J turn the minus indicator on. Characters with any other zone turn the zero indicator on. Factor 1 and factor 2 are not used in this operation.
TIME (Time of Day)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>TIME</td>
<td>Blank</td>
<td>Required</td>
</tr>
</tbody>
</table>

The TIME operation accesses the system time of day and, if specified, the system date.

The system date accessed is not the same field as UDATE (the user date), so the system date can differ from UDATE. See the System Reference for the System/36 Environment manual for a complete description of the system date and the OCL DATE statement.

Columns 28 through 32 must contain the operation code TIME, and columns 43 through 48 (the result field) must specify the name of a numeric field with zero decimal positions into which the time of day or the time of day and the system date are written.

To access the time of day only, specify the result field as a six-digit numeric field.

To access both the time of day and the system date, specify the result field as a 12-digit numeric field. The time of day is always placed in positions 1 through 6 of the result field in the format hhmmss, where hh is hours, mm is minutes, and ss is seconds. If the system date is included, it is placed in positions 7 through 12 of the result field. The date format depends on the system date format and can be mmddyy, ddmmyy, or yymmdd.
XFOOT (Summing the Elements of an Array)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>XFOOT</td>
<td>Required</td>
<td>Field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54─55 56─57 58─59</td>
</tr>
</tbody>
</table>

The XFOOT operation can be used only on numeric arrays. XFOOT adds the elements of the array together and places the sum into the field specified as the result field. Factor 1 is not used. Factor 2 contains the name of the array.

Z-ADD (Zero and Add)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>Z-ADD</td>
<td>Required</td>
<td>Field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54─55 56─57 58─59</td>
</tr>
</tbody>
</table>

Factor 2 is added to a field of zeros. The sum is placed in the result field. Factor 1 is not used.

Z-SUB (Zero and Subtract)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Resulting Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Blank</td>
<td>Z-SUB</td>
<td>Required</td>
<td>Field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54─55 56─57 58─59</td>
</tr>
</tbody>
</table>

Factor 2 is subtracted from a field of zeros. The difference, which is actually the negative of factor 2, is placed in the result field. You can use this operation to change the sign of a field. Factor 1 is not used.
Z-SUB (ZERO AND SUBTRACT)
Chapter 28. Using Double-Byte Data

RPG can process double-byte data when you use the double-byte version of OS/400 and the double-byte hardware devices that the double-byte version supports. Display stations with double-byte capability are supported by the WORKSTN file only.

Double-byte support allows the RPG compiler to process IBM-supplied or your user-defined double-byte character sets. Very little error checking is done on double-byte data. Double-byte data is transparent to the RPG II compiler. You must be sure that the double-byte data is processed properly by your program.

Double-byte characters can be present in literals, constants, fields, tables, and arrays. The transparent literal option must be specified in column 57 of the control specification if transparent literals or constants are present in the program. (For more information on the transparent literal option, see “Column 57 (Transparent Literal)” on page 480.) A field, table, or array containing double-byte data is considered to contain alphanumeric data by the RPG compiler. No error checking occurs for double-byte data in fields, tables, or arrays.

Double-byte data has a two-byte representation, rather than a one-byte representation as in the EBCDIC character set. This can cause the RPG operation codes that process data one byte at a time (COMP, MOVE, and so on) to produce incorrect results. In addition, double-byte data is enclosed by the shift-out (S/O) character (hexadecimal 0E) and the shift-in (S/I) character (hexadecimal 0F). These control characters must be taken into consideration when an operation that processes double-byte data is performed. (For more information on considerations that apply to processing double-byte data, see “Processing Considerations” on page 833.)

Specifying Double-byte Data

Double-byte Literals and Constants

Double-byte characters can be specified as literals in factor 1 or factor 2 or the calculation specifications. Double-byte characters can be specified in the constant or edit word section of the output specifications (columns 45 through 70). Double-byte literals and constants must begin with an apostrophe immediately followed by the S/O character. Double-byte literals and constants must end with the S/I character immediately followed by the ending apostrophe.

Note: When double-byte literals or constants are processed by RPG, the S/O and S/I characters are considered part of the literal or constant data. When the constant is displayed or printed on a double-byte device, these control characters appear as blanks.

When a double-byte literal or constant is used, the transparent literal option must be specified in column 57 of the control specification. When this option is specified, the compiler checks for literals or constants that begin with an apostrophe followed by the S/O character. If a literal or constant is found that begins with an apostrophe followed by the S/O character, the compiler checks to see if the literal or constant is valid.
A literal or constant is not valid if:

- A second S/O character is found before the S/I characters.
- An odd number of one-byte characters are found between the S/O and S/I characters.
- The S/I character is not immediately followed by the ending apostrophe.

An invalid transparent literal or constant is rechecked to see if it is a valid alphabetic literal or constant. If the literal or constant is a valid transparent literal or constant, it is not checked for embedded apostrophes.

Any double-byte character can be entered in a double-byte literal or constant. Each double-byte character has a two-byte hexadecimal representation. A double-byte blank also occupies two bytes. Because each character occupies two bytes in storage, double-byte constants can only be from 1 to 11 characters long (this also allows for the control characters), and literals can only be up to three characters long (this also allows for the control characters).

Note: A double-byte literal or constant can be composed only of double-byte data. Mixing double-byte and EBCDIC data in the same constant means the literal or constant will be checked as alphabetic.

Double-byte Fields, Tables, and Arrays

Double-byte characters can be present in fields, tables, and arrays. The RPG compiler does not recognize these characters as double-byte. The compiler treats double-byte characters as alphabetic. Double-byte fields, tables, and arrays must therefore conform to the rules for alphabetic fields, tables, and arrays.

When double-byte data is present in a field, table, or array, the data must be enclosed in the S/O and S/I characters. These control characters are considered part of the field, table element, or array element. Therefore, when the length of the field, table element, or array element is defined, space must be left for the control characters. For example, if you want to define a field so that it can contain four double-byte characters, you must specify a field length of 10 (two positions for each double-byte character, and one position for each control character). If you do not specify a large enough length, the field, table element, or array element is truncated, causing one of the control characters to be lost.

You must also consider the control characters when the field, table element, or array element is processed. For example, if a field is being printed or displayed on a double-byte device, the control characters appear as blanks. If blank after (column 39 of the output specifications contains a B) is specified for a field, the control characters are also blanked out and must be reconstructed if the field is to still contain double-byte data.

Note: When a field, table, or array contains double-byte data, it should contain only double-byte data. Mixing double-byte and EBCDIC data in the same field, table, or array can cause incorrect results.
Double-byte Comments

Double-byte characters can be entered as comments in source statements. If you enter double-byte characters as comments, but do not enclose them by control characters they are not displayed as double-byte characters. The source statements that allow comments are the extension specifications (columns 58 through 74) and the calculation specifications (columns 60 through 74). Double-byte characters can also be specified on a comment line (column 7 contains an asterisk).

Processing Considerations

Double-byte data can produce incorrect results when used with certain RPG operation codes. Because double-byte data has a two-byte hexadecimal representation, operations that compare data byte by byte are not meaningful unless they check for an equal condition. Care must also be taken when double-byte data is moved. If the lengths of the data being moved and the area that the data is being moved to are not correctly specified, the S/O or S/I characters can be lost.

A number of RPG operations and functions operate by comparing data one byte at a time. The COMP and LOKUP operations compare for high, low, and equal conditions. These operations compare the one-byte EBCDIC values that correspond to the data that is present and produce a result based on the standard one-byte collating sequence. Because of this, the only valid test when double-byte data is being processed is for an equal condition. If all the bytes in a field are equal to all the bytes in another field, the fields are equal whether they contain double-byte or EBCDIC data.

Match fields and sequence checking are also invalid for double-byte data. Match fields compare data from different records, one byte at a time. This produces incorrect results for double-byte data. Sequence checking compares data in different fields to see if the fields are in ascending or descending order. This comparison is done one byte at a time and therefore produces incorrect results for double-byte data.

The SETLL operation is another one-byte comparison operation that cannot be used with double-byte data. This operation compares the key of each record with a lower limit value. If the key of the record is higher than the lower limit, the record is selected for processing. As this comparison is carried out using one-byte EBCDIC values, the SETLL operation can produce incorrect results when used with double-byte data.

RPG allows you to define an alternate collating sequence for EBCDIC data. In other words, you can redefine the order in which one-byte segments of data will be sorted. This is meaningless for double-byte data.

Care must be taken when the various move operations (MOVE, MOVEA, MOVEL) are used with double-byte data. The length of the field, table element, or array element to which the double-byte data is being moved must be defined as being exactly the same length as the literal, field, table element, or array element being moved. If the lengths are not the same, the data will not be recognized as double-byte. For example, if the field to which the data is being moved is shorter than the length of the double-byte data, the data is truncated, losing one of the control characters. If the field to which the data is being moved is longer than the double-byte
Moving Double-byte Data and Deleting Control Characters (SUBR40)

SUBR40 is a move and edit routine that moves the contents of one field to another field. If the S/O and S/I characters are found as the first and last characters in the field, SUBR40 deletes them.

SUBR40 is called as shown in Figure 312.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C EXIT SUBR40
C RLABL EMPNO 10 SENDING FIELD
C RLABL SOCSEC 8 RECEIVING FIELD
C RLABL RETCDE 1 RETURN CODES
C RLABL SNDLEN 30 SENDING LENGTH
C RLABL RECLEN 30 RECEIVING LEN

If you want the receiving field to contain all the data that was 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 deleting the S/O and S/I characters (and the two positions they occupied). 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 right with blanks (one-byte EBCDIC blanks). If the receiving field is shorter than the sending field minus two positions, the data being moved is truncated on the right.

Five RLABL fields must be specified when SUBR40 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 RLABLs for the call to SUBR40 (in Figure 312, you would need to load the lengths of EMPNO and SOCSEC). The return code field must be defined as a one-position alphameric field; the length fields must be defined as three-position numeric fields with zero decimal positions.

SUBR40 produces return codes to indicate the status of the move operation. The following list contains these return codes and their meanings:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data moved; no errors.</td>
</tr>
<tr>
<td>1</td>
<td>Data moved; padding occurred.</td>
</tr>
<tr>
<td>2</td>
<td>Data moved; truncation occurred.</td>
</tr>
<tr>
<td>3</td>
<td>Data moved; S/O and S/I characters were not found.</td>
</tr>
</tbody>
</table>

data, one of the control characters will be embedded in the field. This control character is considered part of the data.
If more than one return code can be issued, only the highest return code is returned.

### Moving Double-byte Data and Adding Control Characters (SUBR41)

SUBR41 is a move and edit routine that moves the contents of one field into another field. If the S/O and S/I characters are not found in the first and last positions of the field, SUBR41 adds them to the field when it is moved.

SUBR41 is called as shown in Figure 313.

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CC1N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C EXIT SUBR41
C RLABL SOCSEC 8
C RLABL EMPNO 10
C RLABL RETCDE 1
C RLABL SNDLEN 30
C RLABL RECLEN 30

*Figure 313. Calling SUBR41*

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 two positions longer than the length of the sending field (to hold the S/O and S/I characters). If you specify a receiving field that is longer than the sending field plus two, the data is padded on the right 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 right when it is moved. If the receiving field is specified either longer or shorter than the sending field plus two positions, the S/I character is still placed in the correct position (the rightmost position).

Five RLABL fields must be specified when SUBR41 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 RLABLs for the call to SUBR41 (in Figure 313, you would need to load the lengths of SOCSEC and EMPNO). The return code field must be defined as a one-position alphabetic field; the length fields must be defined as three-position numeric fields with zero decimal positions.
MESSAGES

SUBR41 produces return codes to indicate the status of the move. The following list contains these return codes and their meanings:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data moved; no errors.</td>
</tr>
<tr>
<td>1</td>
<td>Data moved; padding occurred to left of S/I character.</td>
</tr>
<tr>
<td>2</td>
<td>Data moved; data truncated to left of S/I character.</td>
</tr>
<tr>
<td>3</td>
<td>Data moved; S/O and S/I already present.</td>
</tr>
<tr>
<td>6</td>
<td>Data not moved. Either odd field length found, length of zero found, length greater than 256, or invalid character found in field length. Length specified in fourth and fifth parameters is greater than the field length of the first and second parameters respectively. Actual length of the sending or receiving field is less than the passed length specification in the fourth or fifth parameter.</td>
</tr>
</tbody>
</table>

If more than one return code can be issued, only the highest return code is issued.

Double-byte Device Support

The *SIZE, *MODE, *INP, and *OUT keywords identify subfields in the INFDS data structure that contain values for double-byte devices. For more information on these keywords, see "Coding the INFDS Data Structure" on page 194 in Chapter 7, "Using a WORKSTN File."

Messages

The RPG-displayed messages (both compile time and run time) are displayed in either the standard character set or the double-byte character set. The messages are displayed in a double-byte character set if double-byte support was requested when the user signed on.

The RPG compiler messages are printed in either the standard character set or a double-byte character set. The messages are printed in a double-byte character set if double-byte support was requested when the user signed on.
Appendix A. Differences between System/36 RPG II and AS/400 System/36-Compatible RPG II

If you are currently running an AS/400 system but have previous experience with IBM System/36 RPG II, you should be aware of the differences between the two systems. In addition to the information in this appendix, see Chapter 11, “Using a BSCA File” on page 313 and Chapter 22, “Telecommunications Specifications” on page 537 for more details on specific differences between System/36 RPG II and the AS/400 system.

Changes in Capacity

<table>
<thead>
<tr>
<th>Description</th>
<th>System/36 RPG II</th>
<th>AS/400 System/36-Compatible RPG II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of files in program</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of CHAIN and DEMAND files</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of EXCPT names</td>
<td>64</td>
<td>No limit</td>
</tr>
<tr>
<td>Maximum number of arrays, tables, and data structures</td>
<td>75</td>
<td>Up to 200 arrays and tables. No limit on the number of data structures.</td>
</tr>
<tr>
<td>Maximum number of internal subroutines (BEGSR)</td>
<td>No limit</td>
<td>9999</td>
</tr>
<tr>
<td>Number of OR lines on I- and O-specifications</td>
<td>20</td>
<td>No limit</td>
</tr>
<tr>
<td>Match field and control field length</td>
<td>144</td>
<td>256</td>
</tr>
<tr>
<td>Output file for DEBUG operation code:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Minimum record length</td>
<td>No minimum</td>
<td>80</td>
</tr>
<tr>
<td>• CRT device allowed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Source record comments included on compiler listing: (If specified, columns beyond 91 will be ignored.)</td>
<td>Columns 81-96</td>
<td>Columns 81-91</td>
</tr>
</tbody>
</table>

RPGC and AUTOC Procedures

Certain parameters are ignored on the RPGC and AUTOC procedures in the System/36 Environment. To find the parameters that are ignored see “Using RPGC to Compile an RPG II Source Program in the System/36 Environment” on page 45 for RPGC and “Using AUTOC to Compile an Auto Report Source Program in the System/36 Environment” on page 682 for AUTOC.

Calling RPG II Programs from an RPG II Program

In the S/36 Execution Environment of the AS/400, when an RPG II program calls another RPG II program and the first workstation operation in the called program is a READ, a Major Return Code of 01 will result.

On a System/36, when an RPG II program calls another RPG II program and the first workstation operation in the called program is a READ, a Major Return Code of 00 will result.
Control Specification

The following columns on the control specification are ignored on AS/400 System/36-Compatible RPG II:

- Column 10, Object Output
- Column 11, Listing Options
- Columns 12-14, Size to Execute.

Operation Codes

The following operation codes have been added to AS/400 System/36-Compatible RPG II:

- CALL
- PARM
- PLIST
- FREE
- RETRN.

SUBR22

SUBR22 is not supported on AS/400 System/36-Compatible RPG II.

Program Name

You can specify a program name of up to eight characters when you are using the CL commands CRTS36RPG or CRTS36RPT.

Multiple Field Definition

AS/400 System/36-Compatible does not accept some cases of multiple field definitions for which the System/36 RPG II compiler would only have issued a warning message. The AS/400 System/36-Compatible RPG II compiler will issue terminal errors for the following:

- The name used on a data structure subfield definition was previously used on a data structure subfield definition.
- A field name definition is used on a data structure subfield with a different length or decimal position specification than was specified on a previous definition (i.e. field previously defined on an input record format).

SETLL

There are fundamental differences between System/36 RPG II and AS/400 System/36-Compatible RPG II in terms of handling the SETLL operation. On the System/36, file positioning is not done on the SETLL operation; it is deferred until the next input operation, such as READ or READE. If there is an output operation between the SETLL code and the next input operation in the same file, the SETLL code becomes void, and the SETLL position is ignored for any preceding input operations. On the AS/400 system, the SETLL operation results in an actual I/O call that is made to position the file. Any update operation after a SETLL operation is not valid. This is consistent for all RPG products on the AS/400 system, and it avoids possible confusion in programs using file-sharing of open data paths.

To be compatible with System/36 RPG II, an input operation must be the first I/O operation following a SETLL operation. If there is an output operation between SETLL code and the input operation, (for example an EXCPT output), removing the
SETLL code causes the program to run exactly as in the System/36 environment. If the user still wants file positioning by using the SETLL code, the SETLL code must be moved after the EXCPT operation code. If the SETLL code is before the EXCPT output when updating a record, the AS/400 System/36-Compatible RPG II compiler issues the error `RPG9Sz` indicating that before attempting to update a record, the record must be retrieved and locked.

**Note:** The error message is issued only for an update with fields. No error is issued for an update without fields.

If changes are made to the same physical file between the SETLL code and the input operation by either the current RPG II program using another logical view of the file or by another program running concurrently and sharing the same file, move the SETLL code directly before the input operation. You may need to use an extra indicator and save the key value to ensure that the SETLL operation is done under exactly the same conditions as before.

The two examples in Figure 314 on page 840 show you how to change the source program.
Example 1 old code:
There is an EXCPT output to FILE between SETLL and the READ.
Assume first read will get record 2. KEY is set to record 5.
In System/36 the second read will get record 3, because SETLL is ignored in this case.
In AS/400 the second read will get record 5, positioned by record 5.

```
C          READ FILE  Assume rec 2
C          KEY      SETLLFILE  Key is rec 5
C          EXCPT
C          READ FILE
```

Example 1 new code: -- To run exactly as on System/36:
Remove SETLL code, as it is ignored in System/36.

```
C          READ FILE
C          EXCPT
C          READ FILE
```

Example 1 new code: -- Positioning by SETLL is still desired:
Move SETLL code, as it is ignored in System/36.

```
C          READ FILE
C          EXCPT
C          KEY      SETLLFILE
C          READ FILE
```

Figure 314 (Part 1 of 2). Examples to Change the Examples
C*
C* Example 2 old code:
C* Two logical views of the same file FILE by OCL STATEMENTS.
C*
C* FILE1 FILE2 (Logical views)
C* \ / \ /
C* FILE (Physical file)
C*
C* Between SETLL and READ, new records are added to FILE2.
C*
C* 1 ...+ ... 2 ...+ ... 3 ...+ ... 4 ...+ ... 5 ...+ ... 6 ...+ ... 7 ...
CSETS1N01N02N03 Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C L1 03 KEY SETLLFILE1
C
C
C
C
C READ FILE
C*" = Records are added to FILE2
C*
C* Example 2 new code:
C* Assume key value KEY has not changed in between.
C*
C* 1 ...+ ... 2 ...+ ... 3 ...+ ... 4 ...+ ... 5 ...+ ... 6 ...+ ... 7 ...
CSETS1N01N02N03 Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++*
C L1 03 SETON 55
C
C
C
C
C 55 KEY SETLLFILE1
C READ FILE1
C SETOF 55
C*
C*" = Records are added to FILE2

Figure 314 (Part 2 of 2). Examples to Change the Examples

“Y” Edit Code

AS/400 System/36-Compatible RPG II generates different end positions when a field with length less than 3 or greater than 6 is written out using the “Y” edit code and the end position is not specified by the user.
**EXCPT Operation Conditioned By Printer Overflow Indicators**

When a calculation specification follows an EXCPT operation and both are conditioned by the same printer overflow indicator, a System/36 RPG II program may not repeat the testing of the overflow indicator for the operation following the EXCPT operation. This failure to test the overflow indicator may cause the operations (following the EXCPT operation) to be performed even after the EXCPT operation has changed the value of the printer overflow indicator. Operations, following the EXCPT operation, conditioned by the same printer overflow indicator as the EXCPT operation, may be affected.

An AS/400 System/36-Compatible RPG II program will properly test the conditioning indicators. This testing of the conditioning indicators may cause the operations (following the EXCPT operation) not to be performed on the AS/400 system. Under the same circumstances, the same operations (following the EXCPT operation) are normally performed on the System/36.

If an incompatibility as described above is found, the AS/400 System/36-Compatible RPG II compiler will diagnose the operation immediately following an EXCPT operation, and a warning message is issued in the compile listing. Receiving the warning message on the compile listing does not necessarily mean that the program will work differently than it does on the System/36.

To prevent the problem, change the source program as follows:
- Place the EXCPT operation and the following operations that are conditioned with the same printer overflow indicators between DO and END operations.
- Condition the DO operation with the printer overflow indicator.
- Delete the individual overflow indicators conditioning the operations between the DO and END operations.

**DUP Key With Numeric Fields**

On the System/36, when the DUP key is pressed at a numeric field, the system fixes the sign portion of the X‘1C’ generated to an F so that the program receives an X‘FC’.

This value is used in many System/36 applications to detect pressing of the DUP key at a numeric field.

Because of the ‘fix decimal data error’ processing on the AS/400 system (See FIXDECDTA on page 44), both the sign and the digit portion are fixed. As a result, the program receives an X‘F0’ in the numeric field.

On the System/36-Compatible RPG II, to detect pressing of the DUP key at a numeric field, you must define a character field that overlays the numeric field in the input record specification. Then check the character field for X‘1C’ instead of the numeric field for X‘FC’.
## Appendix B. Glossary of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPC</td>
<td>Advanced Program-to-Program Communications</td>
<td>Data communications support that allows programs on other systems having compatible communications support.</td>
</tr>
<tr>
<td>BSC</td>
<td>Binary Synchronous Communications</td>
<td>A form of communications line control that uses transmission control characters to control the transfer of data over a communications line.</td>
</tr>
<tr>
<td>CL</td>
<td>Control Language</td>
<td>A language used to identify a job and its processing requirements to OS/400.</td>
</tr>
<tr>
<td>DBCS</td>
<td>Double-Byte Character Set</td>
<td>Two-byte characters enclosed by the shift-out (S/O) character (hex 0E) and the shift-in (S/I) character (hex 0F).</td>
</tr>
<tr>
<td>DDM</td>
<td>Distributed Data Management</td>
<td>A feature of the system that allows an application program to work on files that reside on a remote system.</td>
</tr>
<tr>
<td>DDS</td>
<td>Data Description Specifications</td>
<td>A description of the user’s data base or device files that is entered into the system using a fixed-form syntax. The description is then used to create files.</td>
</tr>
<tr>
<td>DLE</td>
<td>Data Link Escape character</td>
<td>In BSC, a transmission control character usually used in transparent text mode to indicate that the next character is a transmission control character.</td>
</tr>
<tr>
<td>DTE</td>
<td>Data Terminal Equipment</td>
<td>The data processing unit that uses communications lines.</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Extended Binary-Coded Decimal Interchange Code</td>
<td>A set of 256 eight-bit characters.</td>
</tr>
<tr>
<td>EOT</td>
<td>End-of-Transmission character</td>
<td>In binary synchronous communications, the transmission control character usually used to end communications.</td>
</tr>
<tr>
<td>ETB</td>
<td>End-of-Transmission-Block character</td>
<td>In binary synchronous communications, the transmission control start-of-text character.</td>
</tr>
<tr>
<td>ETX</td>
<td>End-of-Text character</td>
<td>In binary synchronous communications, the transmission control character used to end a logical set of records that began with the start-of-text character.</td>
</tr>
<tr>
<td>ICF</td>
<td>Intersystem Communications Function</td>
<td>A feature of OS/400 that allows a program to interactively communicate with another program or system.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
<td>Pertaining to either input or output, or both.</td>
</tr>
<tr>
<td>IRP</td>
<td>Intermediate Representation of a Program</td>
<td>A listing of a program that has been translated, but requires more processing to become a program object.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Term</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>ITB</td>
<td>Intermediate-Text-Block character</td>
<td>In binary synchronous communications, the transmission control character used to indicate the end of a section of checked data.</td>
</tr>
<tr>
<td>MIC</td>
<td>Message Identification Code</td>
<td>A four-digit number that identifies a record in a message member. This number can be part of the message identifier.</td>
</tr>
<tr>
<td>MRT program</td>
<td>Multiple Requester Terminal Program</td>
<td>A program that can process requests from more than one display station or ICF session at the same time using a single copy of the program.</td>
</tr>
<tr>
<td>NEP</td>
<td>Never-Ending Program</td>
<td>A long-running program that does not share system resources, except for shared files and the spool file.</td>
</tr>
<tr>
<td>OCL</td>
<td>Operation Control Language</td>
<td>A language used to identify a job and its processing requirements to OS/400.</td>
</tr>
<tr>
<td>RPG</td>
<td>Report Program Generator</td>
<td>A programming language designed for developing business application programs, especially reports.</td>
</tr>
<tr>
<td>RVI</td>
<td>Reverse-Interrupt character</td>
<td>In binary synchronous communications, a request by the receiving station to the sending station to stop sending and receive a message.</td>
</tr>
<tr>
<td>SDA</td>
<td>Screen Design Aid</td>
<td>The part of the Application Development Tools that helps the user design, create, and maintain displays and menus. SDA can also generate specifications for RPG workstation programs.</td>
</tr>
<tr>
<td>SEU</td>
<td>Source Entry Utility</td>
<td>One of the Application Development Tools, used to enter and update source and procedure members.</td>
</tr>
<tr>
<td>$SFGR</td>
<td>Screen Format Generator</td>
<td>A utility program for compiling SFGR specifications, which are used to design displays, into a display file.</td>
</tr>
<tr>
<td>SNA</td>
<td>Systems Network Architecture</td>
<td>A set of rules for controlling the transfer of information in a data communications network.</td>
</tr>
<tr>
<td>SRT</td>
<td>Single Requester Terminal program</td>
<td>A program that can process requests from only one display station or ICF session from each copy of the program.</td>
</tr>
<tr>
<td>STX</td>
<td>Start-of-Text character</td>
<td>In binary synchronous communications, a transmission control character used to begin a logical set of records that will be ended by the end-of-text character or end-of-transmission-block character.</td>
</tr>
</tbody>
</table>
Bibliography

For additional information about topics related to System/36-Compatible PG II programming on the AS/400 system, refer to the following IBM AS/400 publications:

- Information Directory, GC41-9678 Short title: Publications Guide
- Communications: Operating System/400* Communications Configuration Reference, SC41-0001 Short title: OS/400* Communications Configuration Reference
- Programming: Data Management Guide, SC41-9658 Short title: Data Management Guide
- IBM AS/400: System/36 to AS/400 Migration Aid User’s Guide and Reference, SC09-1166 Short title: System/36 to AS/400 Migration Aid User’s Guide and Reference
- Programming: Control Language Reference, SC41-0030 Short title: CL Reference
- Programming: Reference Summary, SX41-0028 Short title: Programming Reference Summary
- Licensed Programs Installation Guide, SC41-9878 Short title: Licensed Programs and New Release Installation Guide
- Programming: Data Description Specifications Reference, SC41-9620 Short title: DDS Reference

© Copyright IBM Corp. 1994
Glossary of Terms

The AS/400 system glossary defines terms that are used in the customer documentation for the AS/400 system. The *Dictionary of Computing, SC20-1699*, contains definitions of general data processing terms.

The AS/400 system glossary includes definitions developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Terms have been reproduced for the *American National Dictionary for Information Processing*, copyright 1982 by the Computer and Business Equipment Manufacturers Association. Copies of this dictionary may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions from this dictionary are identified by (ANSI) at the beginning of the definition.

**#LIBRARY.** The library, provided with the system, that is the default current library.

**access method.** A method used to read a record from, or to write a record into a file. Access can be sequential (records are referred to one after another in the order in which they appear in the file), or it can be random (the individual records can be referred to in any order).

**acquire.** To assign a display station or session to a program.

**adapter.** See *communications adapter*.

**address.** A name, label, or number that identifies a location in storage, a device in a network, or any other data source.

**address output file.** Record address file, containing relative numbers, that is produced by a sort program.

**addrout file.** See *address output file*.

**allocate.** To assign a resource, such as a disk file or a diskette file, to perform a specific task.

**alphabetic character.** Any one of the letters A through Z (uppercase and lowercase), or any one of the special characters #, $, and @.

**alphameric.** Consisting of letters, numbers, and often other symbols, such as punctuation marks and mathematical symbols.

**alternative collating sequence.** A user-defined collating sequence that replaces the standard EBCDIC collating sequence.

**alternative index.** An index that is built after an indexed file is created and that provides a different order for reading or writing records in the file. Contrast with *primary index*.

**application.** (1) A particular business task, such as inventory control or accounts receivable. (2) A group of related programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

**Application Development Tools.** A licensed program that contains the source entry utility (SEU), and the screen design aid (SDA).

**application program.** A program used to perform an application or part of an application.

**array.** A series of elements with like characteristics. An array can be searched for a uniquely identified element, or elements in an array can be accessed by their position relative to other elements. Contrast with *table*.

**array element.** A single data item in an array.

**ascending key sequence.** The arrangement of data in order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence*.

**asynchronous transmission.** In data communications, a method of transmission in which sending and receiving of data is controlled by control characters instead of by a timing sequence. Contrast with *synchronous transmission*.

**automatic answer.** In data communications, the ability of a station to receive a call over a switched line without operator action. Contrast with *manual answer*.

**automatic call.** In data communications, the ability of a station to place a call over a switched line without operator action. Contrast with *manual call*.

**automatic report.** An RPG option that simplifies the defining of formats for printed reports and that allows the previously written statements included in new programs.

**batch processing.** A processing method in which a program or programs process records with little or no operator action. Contrast with *interactive processing*.

**binary.** (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.
**binary synchronous communications (BSC).** A form of communications line control that uses transmission control characters to control the transfer of data over a communications line. Compare with *synchronous data link control*.

**bit.** Either of the binary digits 0 or 1. See also *byte*.

**block.** (1) A group of records that are recorded or processed as a unit. Same as *physical record*. (2) In data communications, a group of records that is recorded, processed, or sent as a unit.

**branch instruction.** An instruction that changes the sequence in which the instructions in a computer program are performed. The sequence of instructions continues at the address specified in the branch instruction.

**buffer.** (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read or from which data is written.

**byte.** A group of eight bits. In the EBCDIC coding system, one byte can represent a character. In the double-byte coding system, two bytes represent a character.

**call.** (1) To activate a program or procedure at its entry point. Compare with *load*. (2) In data communications, the action necessary in making a connection between two stations on a switched line.

**cancel.** To end a task before it is completed.

**chained file.** An input, output, or update disk file from which records can be read randomly.

**character.** Any letter, digit, or other symbol in the data character set that is part of the organization, control, or representation of data.

**code.** (1) Instructions for the computer. (2) To write instructions for the computer. Same as *program*. (3) A representation of a condition, such as an error code.

**collating sequence.** The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

**combined file.** A file used as both an input and an output file. The fields are not necessarily the same in the input and output records.

**command keys.** A keyboard key that has been assigned a particular function and are generally labelled as such, (for example HELP, PRINT).

**comment.** Words or statements in a program or procedure that serve as documentation rather than as instructions.

**communications adapter.** A hardware feature that enables a computer or device to become a part of a data communications network.

**compilation time.** The time during which a source program is translated from a high-level language to a machine language program.

**compile.** To translate a program written in a high-level programming language into a machine language program.

**compile-time array.** An array that is compiled with the source program and that becomes a permanent part of the compiled program. Contrast with *run-time array* and *prerun-time array*.

**compile-time table.** A table that is built into the source program and that becomes a permanent part of the compiled program. Contrast with *prerun-time table*.

**compiler.** A program that translates instructions written in a high-level programming language into machine language.

**compression.** (1) A technique for removing strings of duplicate characters and for removing trailing blanks before transmitting data. (2) In data communications, a technique for removing strings of duplicate characters.

**conditioning.** The use of indicators to control when calculations or output operations are done.

**conditioning indicator.** An indicator used to indicate when calculations are done or which attributes apply to a format or format field.

**configuration.** The arrangement of devices and programs that make up a data processing system. See also *system configuration*.

**configure.** (1) To describe the interconnected arrangement of the devices, programs, communications, and optional features installed on a system. (2) To describe setting up of auxiliary storage pools and check sum protection.

**consecutive processing.** The processing of records in the order in which they exist in a file. Same as *sequential processing*. See also *random processing*.

**console.** A display station from which an operator can control and observe the system operation.

**constant.** A data item with a value that does not change. Contrast with *variable*.  

---

848 System/36-Compatible RPG II User’s Guide and Reference
continuation line. Additional lines specified on the file description specifications to provide more information about the file being defined.

data management. The system programs that provide access to data, perform or monitor storage of data, and control input/output devices.

data terminal equipment (DTE). The part of the data link that sends data, receives data, and provides the data communications control function.

data type. A characteristic used for defining data as numeric or character.

decimal. Pertaining to a system of numbers to the base ten; decimal digits range from 0 through 9.

delete-capable file. A file from which records can be logically removed without compressing the file.

delta position. The position in the edit word that corresponds to the leftmost position in the data field.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with ascending key sequence.

detail record. A record that contains the daily activities or transactions of a business. For example, the items on a customer order are typically stored in detail records. Contrast with header record.

detail time. A portion of the RPG program cycle during which calculation and output operations for specified fields are performed for each record read.

direct file. A disk file in which records are referenced by the relative record number. Contrast with indexed file and sequential file.

disk. A storage device made of one or more flat, circular plates with magnetic surfaces on which information can be stored.

disk file. A set of related records on disk that are treated as a unit.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copied from the disk.

display. (1) A visual presentation of information on a display screen. (2) To show information on the display screen.

control break. A change in the contents of a control field that indicates all records from a particular control group were read and a new control group is starting.

current library. The first library searched for any required members. The current library can be specified during sign-on or while running programs and procedures.

current record. The record that is currently available to the program.

data communications. The transmission of data between computers and/or remote devices (usually over a long distance).

data dictionary. (IDDU) An object for storing fields, record format, and file definitions.

data type. A characteristic used for defining data as numeric or character.

decimal. Pertaining to a system of numbers to the base ten; decimal digits range from 0 through 9.

delete-capable file. A file from which records can be logically removed without compressing the file.

delta position. The position in the edit word that corresponds to the leftmost position in the data field.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with ascending key sequence.

detail record. A record that contains the daily activities or transactions of a business. For example, the items on a customer order are typically stored in detail records. Contrast with header record.

detail time. A portion of the RPG program cycle during which calculation and output operations for specified fields are performed for each record read.

direct file. A disk file in which records are referenced by the relative record number. Contrast with indexed file and sequential file.

disk. A storage device made of one or more flat, circular plates with magnetic surfaces on which information can be stored.

disk file. A set of related records on disk that are treated as a unit.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copied from the disk.

display. (1) A visual presentation of information on a display screen. (2) To show information on the display screen.
display format. Data that defines (or describes) a display.

display layout sheet. A form used to plan the location of data on the display.

display screen. The part of the display station on which information is displayed.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent or receive information from the system.

distributed data management (DDM). A function of the operating system that allows an application program or user on one system to use data files stored on remote systems. The systems must be connected by a communications network, and the remote systems must also be using DDM.

Double-Byte Character Set. A set of characters in which each character is represented by two bytes. There are four double-byte character sets supported by the system: Japanese, Korean, Simplified Chinese, and Traditional Chinese.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

EBCDIC character. Any one of the symbols included in the eight-bit EBCDIC set.

edit. (1) To modify the form or format of data; for example, to insert or remove characters such as for dates or decimal points. (2) To check the accuracy of information that has been entered, and to indicate if an error is found.

edit code. A number or letter indicating that editing should be done according to a defined pattern.

element. The smallest addressable unit of data in a table or array.

embedded blanks. Blanks that are surrounded by any other characters.

end-of-text (ETX) character. In binary synchronous communications, the transmission control character used to end a logical set of records that began with the start-of-text character.

end-of-transmission-block (ETB) character. In binary synchronous communications, the transmission control character usually used to end a block of records.

enter. To type in information on a keyboard and press the Enter key in order to send the information to the computer.

EXCEPT group name. A name used in the place of indicators to identify a record or group of records written at exception output time.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

external indicators. Indicators that can be set by another program before a program is run or changed while a program is running. The external indicators are U1 through U8.

feature. A programming or hardware option, usually available at an extra cost.

field. One or more characters of related information (such as a name or an amount).

file. A set of related records treated as a unit.

file name. The name used by a program to identify a file. See also label.

first-page indicator. Coded as 1P. An indicator that specifies which lines (such as headings) should be printed on the first page only.

fixed currency symbol. A currency symbol that appears in a specified position of an edited field. Contrast with floating currency symbol.

floating currency symbol. A currency symbol that appears immediately to the left of the leftmost position that is used in an edited field. Contrast with fixed currency symbol.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) To arrange such things as characters, fields, and lines.

full-procedural file. A file that uses input operations controlled by programmer-specified operation codes instead of by the program cycle. Contrast with primary file.

function key. A keyboard key designated Fn, where n is 1-24. These keys are assigned a function that can be changed. They allow the user to select keyboard functions or programmer functions.
**function-key indicator.** Coded as KA through KL, and KQ through KY. An indicator that is set on when an operator presses the corresponding function key.

**group indication.** The printing of control information for only the first record of a group of records containing identical control information.

**half-adjust.** A method of rounding off a number by adjusting the last significant digit.

**hardware.** Physical equipment, rather than programs, procedures, rules, and associated information.

**header record.** A record that contains information, such as customer name and customer address, that is common to following detail records. Contrast with detail record.

**hexadecimal.** Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

**host system.** The primary or controlling computer in the communications network. See also control station.

**I/O.** See input/output.

**ID.** Identification.

**index.** A table containing the key value and location of each record in an indexed file.

**indexed file.** A file in which the key and the position of each record is recorded in a separate portion of the file called an index. Contrast with direct file and sequential file.

**indicator.** An internal switch that communicates a condition between parts of a program or procedure.

**informational message.** A message that provides information to the operator, but does not require a response.

**initialize.** To prepare for use. For example, to initialize a diskette.

**input.** Data to be processed.

**input/output (I/O).** Data provided to the computer or data resulting from computer processing.

**input/output processor (IOP).** One or more circuits that process programmed instructions and control one or more input or output adapters.

**inquiry.** (1) A request for information in storage. (2) A request that puts a display station into inquiry mode. (3) In data communications, a request for information from another system.

**inquiry mode.** A mode during which the job currently running from a display station is interrupted so that other work can be done. The operator puts the display station in inquiry mode by pressing the System Request key.

**inquiry program.** (1) A program that allows an operator to get information from a disk file. (2) A program that runs while the system is in inquiry mode.

**interactive.** Pertains to activity involving requests and replies as, for example, between an operator and a program or between two programs.

**interactive processing.** A processing method in which each operator action causes a response from the program or the system. Contrast with batch processing.

**intermediate block check.** In binary synchronous communications, an option that permits checking each record, instead of checking the contents of the total buffer, when large buffers of data are received.

**intermediate-text-block (ITB) character.** In binary synchronous communications, the transmission control character used to indicate the end of a section of data to be checked. See intermediate block check.

**intermediate representation of program (IRP).** A listing of a program in an intermediate language, into which a source program or statement is translated prior to interpretation or further translation.

**Intersystem Communications Function (ICF).** A feature of the operating system that allows a program to interactively communicate with another program or system.

**interrupt.** To stop a process in such a way that it can be continued.

**job.** (1) A unit of work done by a system. (2) One or more related procedures or programs grouped into a procedure.

**job queue.** A list, on disk, of jobs waiting for processing by the system.

**job step.** A unit of work represented by a single program or a procedure that contains a single program. A job consists of one or more job steps.

**K-byte.** 1024 bytes.

**Katakana.** A native Japanese character set that is used to write foreign words phonetically in Japanese.

**key.** One or more characters used to identify the record and establish the record’s order within an indexed file.
**label.** A symbolic name that represents a specific location in a program. A label can serve as the destination point for one or more branching operations.

**level indicator.** Two characters (L0 through L9 and LR) that control calculation and output processing during total time.

**library.** (1) An object on disk that serves as a directory to other objects. A library groups related objects, and permits finding objects by name. (2) The set of publications for a system.

**licensed program.** An IBM-written program that performs functions related to processing user data.

**limits file.** A file that contains upper and lower values of the record keys that can be used to read from an indexed file.

**linkage.** The coding that passes control and parameters between two routines.

**literal.** A character string whose value is defined by the characters themselves.

**load.** (1) To move data or programs into storage. (2) To place a diskette into a diskette drive or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

**local.** Pertaining to a device that is directly connected to your system without the use of a communications line. Contrast with **remote.**

**local data area.** A 1024-byte area that can be used to pass information between programs in a job. A separate local data area is automatically created for each job. RPG II programs access 512 bytes.

**location name.** In interactive communications, the identifying name associated with a particular remote system or device.

**logical file member.** A named logical grouping of data records from one or more physical file members. See also **member.**

**look-ahead field.** A field that allows the program to look at information in a field on the next record that is available for processing in any input or update file.

**loop.** A sequence of instructions that is performed repeatedly until an ending condition is reached.

**machine language.** A language that can be used directly by a computer without intermediate processing.

**main storage.** The part of the processing unit where programs are run. Contrast with **control storage.**

**manual answer.** In data communications, the operator actions required to receive a call over a switched line. Contrast with **automatic answer.**

**manual call.** In data communications, the operator actions required to place a call over a switched line. Contrast with **automatic call.**

**master file.** A collection of permanent information, such as a customer address file.

**match fields.** In primary or secondary multilose processing, fields within a record type that are to be used for checking the order of a single file, or for matching records of one file with those of another file.

**match level.** The value identified by the match field indicators M1 through M9. The match level identifies match fields.

**member.** Different sets of data within one file.

**menu.** A displayed list of items from which an operator can make a selection.

**message.** Information sent to an operator or programmer from a program. A message can be either displayed or printed.

**message identifier.** A seven-character code that identifies a predefined message, and is used to get the message description from a message file.

**message identification code (MIC).** A four-digit number that identifies a record in a message member. This number can be part of the message identifier.

**message member.** A library member that defines the text of each message and its associated message identification code.

**MRT program.** See Multiple Requester Terminal program.

**Multiple Requester Terminal (MRT) program.** A program that can process requests from more than one display station or ICF session at the same time using a single copy of the program. Contrast with **single requester terminal (SRT) program.**

**never-ending program (NEP).** A long-running program that does not share system resources, except for shared files and the spool file.

**noncontiguous key.** A key made up of characters or character strings which occur in separate areas of a record.
nonswitched line. A connection between computers or devices that does not have to be established by dialing. Contrast with switched line.

not-found indicator. An indicator that is set on when the specified record cannot be found.

numeric. Pertaining to any of the digits 0 through 9.

offline. Neither controlled directly by, nor communicating with, the computer, or both. Contrast with online.

online. Being controlled directly by, or directly communicating with, the computer, or both. Contrast with offline.

operation. A defined action, such as adding or comparing, performed on one or more data items.

Operation Control Language (OCL). A language used to identify a job and its processing requirements to the System/36 Environment.

output. Information or data received from a computer that is shown on a display, printed on the printer, or stored on disk, diskette, or tape.

overflow indicator. An indicator that signifies that the last line on a page has been printed or skipped.

overflow line. The line specified as the last line printed on a page.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

packed decimal format. A format in which each byte (except the rightmost byte) within a field represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with zoned decimal format.

packed key. An index key in packed decimal format.

parameter. A value supplied to a procedure or program that either is used as input or controls the actions of the procedure or program.

physical file. (1) A description of how data is to be presented to or received from a program and how data is actually stored in the database. A physical file contains one record format and one or more members. (2) An indexed file containing data for which one or more alternative index files have been created.

point-to-point line. A communications line that connects a single remote station to a computer.

poll. To determine if any remote device on a communications line is ready to send data.

polling. (1) The process whereby stations are invited, one at a time, to transmit. (2) The process whereby a controlling station contacts the attached devices to avoid contention, to determine operational status, or to determine readiness to send or receive data.

prerun-time array. An array that is loaded at the same time as the user program, before the program begins. Contrast with compile-time array and run-time array.

prerun-time table. A table that is loaded at the same time as the user program, before the program begins. Contrast with compile-time table.

primary file. If specified, the main file from which RPG first reads a record in the program cycle. In multile file processing, the primary file is used to determine whether the MR indicator is set on. Contrast with full-procedural file.

primary index. The index that is built when an indexed file is created. Contrast with alternative index.

printout. Information from the computer that is produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

procedure. A set of related operation control language statements (and, possibly, utility control statements and procedure control expressions) that cause a specific program or set of programs to be performed.

processing unit. The part of the system unit that performs instructions and contains main storage.

program. (1) A sequence of instructions for a computer. See source program and program object. (2) To write a sequence of instructions for a computer. Same as code.

program cycle. In RPG, the series of operations performed by the computer for each record read.

program date. The date associated with a program (job step). See also creation date, session date, and system date.
**program object.** The result of compiling a source program is a program object that is coded in machine language and can therefore run more efficiently.

**program stack.** A list of programs linked together as a result of programs calling other programs with the CALL instruction, or implicitly from some other event, within the same job.

**program template.** A stream of hexadecimal data which includes the program instructions, variables and data; it is the object program.

**prompt.** A displayed request for information or operator action.

**queue.** A line or list formed by items waiting for processing.

**random access.** An access method in which records can be read from, written to, or removed from a file in any order.

**random by key.** A processing method for chained files in which record keys identify records to be processed.

**random by relative record number.** A processing method for chained files in which relative record numbers identify the records processed.

**random processing.** The processing of records in an order other than the order that they exist in a file. See also consecutive processing and sequential processing.

**record.** A collection of fields that is treated as a unit.

**record address file.** An input file that indicates to a program which records are read from a disk file, and the order in which these records are read from the disk file.

**record identification code.** Characters placed in a record to identify that record type.

**record identifying indicator.** An indicator that identifies the record just read.

**record type.** The classification of records in a file.

**relative record number.** A number that specifies the location of a record in relation to the beginning of the file.

**remote.** Pertaining to a system or device that is connected to your system through a communications line. Contrast with local.

**requester.** A display station or interactive communications session that requests a program to be run.

**resulting indicator.** An indicator that is set depending on the result of an operation.

**return code.** In data communications, a value generated by the system or subsystem that is returned to a program to indicate the results of an operation issued by that program.

**reverse-interrupt character (RVI).** In binary synchronous communications, a request by the receiving station to the sending station to stop sending and receive a message.

**routine.** A set of statements in a program that causes the system to perform an operation or a series of related operations.

**RPG II.** Report Program Generator. A commercially oriented programming language specifically designed for writing application programs intended for business data processing.

**run.** To cause a program, utility, or other machine function to be performed.

**run-time array.** An array that is loaded after the program begins. Contrast with compile-time array and prerun-time array.

**scratch file.** A file, usually used as a work file, that exists until the program that uses it ends.

**screen design aid (SDA).** A function of the AS/400 system Application Development Tools licensed program that helps the user design, create, and maintain displays and menus.

**search word.** Data used to find a match in a table or array.

**secondary file.** Any input file other than the primary file.

**sequential access.** An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

**sequential by key.** A method of indexed file processing in which records are read or written in the order of the record keys.

**sequential file.** A file in which records occur in the order in which they were entered. Contrast with direct file and indexed file.

**sequential processing.** The processing of records in the order in which they exist in a file. Same as consecutive processing. See also random processing.

**session.** (1) The length of time that starts when a user signs on and ends when the user signs off at a display.
station. (2) The logical connection by which an AS/400 program or device can communicate with a program or device at a remote location.

session date. The date associated with a session. See also creation date, program date, and system date.

session library. The library specified, or assigned as a default, when signing on or while running a program.

shift-in (S/I) character. A control character that indicates the end of a string of double-byte characters. The shift-in character is represented by hexadecimal 0F.

shift-out (S/O) character. A control character that indicates the start of a string of double-byte characters. The shift-out character is represented by hexadecimal 0E.

single requester terminal (SRT) program. A program that can process requests from only one display station or ICF session from each copy of the program. Contrast with multiple requester terminal program.

source entry utility (SEU). A function of the AS/400 system Application Development Tools licensed program that is used to create and change source members.

source member. A library member that contains information in the form in which it was entered, such as RPG specifications. Contrast with load member.

source program. A set of instructions that are written in a programming language and that must be translated to machine language before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

declaration sheets. Forms on which a program is coded and described.

spool file. A disk file that contains output that has been saved for later printing.

spooling. The system function that saves data in a disk file for later processing or printing.

SRT program. See single requester terminal program.

start-of-text (STX) character. In binary synchronous communications, a transmission control character used to begin a logical set of records that will be ended by the end-of-text character or end-of-transmission-block character.

subroutine. A group of instructions that can be called by another program or subroutine.

switched line. In data communications, a connection between computers or devices that is established by dialing. Contrast with nonswitched line.

synchronous. Occurring in a regular or predictable sequence.

synchronous data link control (SDLC). A form of communications line control that uses commands to control the transfer of data over a communications line. Compare with binary synchronous communications.

synchronous transmission. In data communications, a method of transmission in which the sending and receiving of characters is controlled by timing signals. Contrast with asynchronous transmission.

system. The computer and its associated devices and programs.

system configuration. A process that specifies the machines, devices, and programs that form a particular data processing system.

system console. A display station from which an operator can keep track of and control system operation.

system date. The date assigned by the system operator during the initial program load procedure. See also creation date, program date, and session date.

system library. The library, provided with the system, that contains the operating system for the System/36 Environment. It is called QSSP.

System/36 Environment. A part of OS/400 that emulates the System/36 operating system, mapping a subset of System/36 statements to OS/400 CL commands. The System/36 Environment also supports utility programs that perform common service functions.

systems network architecture (SNA). A set of rules for controlling the transfer of information in a data communications network.

table. (ANSI) A collection of data in which each item is uniquely identified by a label, by its position relative to the other items, or by some other means.

total operations. Calculation and output operations performed only after a group of records has been processed.

total time. The part of the RPG program cycle in which calculation and output operations specified for a group of records are done.

transaction. (1) An item of business. The handling of customer orders and customer billing are examples of transactions. (2) In interactive communications, the
communication between the application program and a specific item (usually another application program) at the remote system.

**transaction file.** A file containing data, such as customer orders, that is usually used only with a master file.

**transmission control characters.** In data communications, special characters that are included in a message to control communication over a data link. For example, the sending station and the receiving station use transmission control characters to exchange information; the receiving station uses transmission control characters to indicate errors in data it receives.

**transparent data.** Data that can contain any hexadecimal value.

**transparent literal.** A literal (or constant) that begins with an apostrophe followed immediately by the shift-out (S/O) character (hexadecimal 0E), and up to 3 double-byte characters for a literal or up to 11 double-byte characters for a constant, and ends with the shift-in (S/I) character (hexadecimal 0F) followed immediately by an apostrophe.

**transparent text mode.** A mode that allows BSC to send and receive messages containing any of the 256 character combinations in hexadecimal, including transmission control characters.

**tributary station.** In data communications, a secondary device on a multipoint line.

**truncate.** To shorten a field or statement to a specified length.

**update file.** A disk file from which a program reads a record, updates fields in the record, and writes the record back into the location it came from.

**valid.** (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

**variable.** A name used to represent a data item whose value can change while the program is running. Contrast with constant.

**work file.** A file that is used for temporary storage of data being processed.

**work station.** A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

**World Trade.** (1) Pertains to the distinction between the countries of the world. (2) Pertains to the IBM World Trade Corporation and its subsidiaries.

**X.21.** In data communications, a specification of the CCITT that defines the connection of data terminal equipment to an X.21 (public data) network.

**zero suppression.** The substitution of blanks for leading zeros in a number. For example, 00057 becomes 57 when using zero suppression.

**zoned decimal format.** A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte; bits 0 through 3 of all other bytes contain 1's (hexadecimal F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with packed decimal format.
Index

Special Characters
/COPY statement specifications 633
See also auto report
changing copied specifications 634
file description specifications 634
input field specifications 636
format of 633
placement in auto report source program 633
/EJECT
See columns 7 through 12 in chapters 17 through 24
/SPACE
See columns 7 through 12 in chapters 17 through 24
/TITLE
See columns 7 through 12 in chapters 17 through 24
(b) (blank) with edit words 450
- (minus), sign for negative balance 451
See also negative balance
$ control specification entry 475
use in edit word (fixed and floating) 451
use with edit code (fixed and floating) 453
$MAINT utility 180
$SFRG utility
for a CONSOLE file 287
for a WORKSTN file 180
* (asterisk)
asterisk fill 450
comment line
See column 7 in chapters 17 through 24
indication of auto report total line 651
use with edit code 446
** alternate collating sequence coding sheet 465
compile-time arrays or tables 402
file translation table 467
look-ahead fields 558
*AUTO output specifications
accumulating totals 651
resetting total fields to zero 651
asterisk indication 651
created calculation specifications 668
created total fields 649
description 644
detail printing 645
field description specifications
1-9 or R in column 39 653
A in column 39 648
B in column 39 646
C in column 39 652
field or constant on created total line 653
group printing
description 655
*AUTO output specifications (continued)
group printing (continued)
examples 656
record description entries 645
report format 660
restrictions in naming fields 650
spacing and skipping 660
total rolling 648
*AUTO page-heading specifications
date 640
description 623
eedit codes 643
field description specifications 642
output indicators 641
record description specifications 641
spacing and skipping 641
*AUTO specifications 623
*BLANK 595
*BLANKS 595
*CANCL keyword 200
*DETC keyword 200
*GETIN keyword 200
*INP keyword 197
*LIKE DEFN operation 742
*MODE keyword 197
*OPCODE keyword 197
*OUT keyword 197
*PLACE special word 613
*RECORD keyword 197
*SIZE keyword 197
*STATUS keyword 196
*Suppress (auto report) 632
*ZERO 595
*ZEROS 595
& (ampersand)
auto report copy function 636
use in edit word 451

Numerics
1P (first-page indicator) 375

A
A$SUM subroutine (auto report) 669
accessing RPG II from OS/400 2
accumulating totals (auto report) 649
ACQ (acquire) operation code
description 722
used with a WORKSTN file 204
acquiring a device 203
ADD (add) operation code
   description 723
adding entries to arrays and tables 415
adding records to a file
   file description specifications
      entry 513
      RECNO continuation-line option 510
output specifications entry 608
to a direct file 133
to a sequential file
   at the end of a file 105
   between records in a file 107
to an indexed file
   randomly by key field 162
additional input/output area 504
address output (addrout) file
   definition 847
   file description specification charts 483
   reading a direct file 124
   reading a sequential file 96
   reading an indexed file 153
   updating a direct file 132
   updating a sequential file 104
   updating an indexed file 161
addressing characters (BSCA file) 318, 546
addrout file
   See address output file
adjusting arithmetic results (half-adjust) 600
allowing command keys to be used 190, 308
   with a WORKSTN file 190
alphabetized operation codes 722
alphabetic literal 594
alternating format (related arrays and tables) 399
alternative collating sequence
   changes to 462
   coding sheet 461
   control specification entry 461
   record format 465
alternative index for an indexed file
   creating 137
   using 139
ALTSEQ 465
ampersand (&)
   auto report copy function 636
   use in edit word 451
AND or OR lines
   *AUTO page-heading specifications 641
   calculation specifications 587
   indicators conditioning output 393
   input specifications 550
   output specifications 263, 607
   record-identifying indicators 357
apostrophe
   used with output constant 619
AR230R, sample program (inquiring into an accounts
   receivable file) 221
AR330R, sample program (maintaining an accounts
   receivable file) 226
AR935R, sample program (requesting a printout of
   accounts receivable) 241
AR936R, sample program (printing accounts
   receivable) 270
AREA field (SUBR21) 205
arithmetic operations 716
array LOKUP 774
array name 522
arrays
   See also tables
   adding entries to 415
   alternating
      definition of 399
      extension specification entry 530
      binary format 529
      changing the contents of
         permanently 415
         temporarily 414
   compile-time
      definition of 397
      loading 402
      placement in source program 403
   creating input records for 399
   decimal positions 529
   defining 400
   differences between arrays and tables 397
   dynamic array
      See run-time array
   editing 417
   examples of using 418
   filename (when required) 492
   index 407
   input specifications entry 577
   kinds of 397
   loading
      compile-time 402
      prerun-time 403
      run-time 404
   maximum allowed per program 517
   moving (MOVEA operation code) 781
   name
      and index 407
      as factor 1 in calculation specifications 590
      as factor 2 in calculation specifications 596
      as field name in *AUTO output 647, 654
      as field name in *AUTO page headings 642
      as result field in calculation specifications 597
      extension specifications entry 521
   number of entries per array 525
   number of entries per record 524
   packed-decimal format 529
   prerun-time
      definition of 397
      loading 403
arrays (continued)
  printing elements, examples of 424
  processing specifications 517
  referring to in calculations 413
  referring to in RLABL operation 813
  related, definition of 397
run-time
  definition of 397
  loading 404
  searching 408
  sequence (extension specifications entry) 529
  similarities between arrays and tables 397
  specifying arrays 413
  summing the elements of (XFOOT operation) 829
  using an array name and index 407
writing 416
AS/400 system, differences from System/36 837
AS/400-to-3740 program 332
ASCII-EBCDIC Character Translation 319
assembler-language subroutine
  EXIT operation 761
  RLABL operation 813
  SUBR01 278
assigning control-level indicators 366
assigning field-record-relation indicators 382
assigning overflow indicators 355
asterisk (*) comment line
  See column 7 in chapters 17 through 24
asterisk fill 450
asterisk indication on auto report total lines 651
auto report
  A$$SUM subroutine 669
  auto report page headings 640
  copy function 633
  created specifications
    calculation 668
    format of 628
    order of 628
    output 669
  examples of using auto report 686
    example 1 686
    example 2 690
    example 3 692
    example 4 694
    example 5 696
    example 6 699
    example 7 701
input for 625
option specifications 630
overriding specifications 631
programming aids 670
report format 660
  See also /COPY statement specifications
  placement of headings and fields 662
  source program 625
  spacing and skipping 660
auto report program
  changing (RPGSEU procedure) 30
  compiling (AUTOC procedure) 682
  creating (RPGSEU procedure) 30
  AUTOC procedure 682
  automatic overflow 265
B
backspace (Record Backspace) key
  *STATUS keyword code 196
  as exception 191
BEGSR (begin subroutine) operation code
  description 724
  in INFSR subroutine 199
binary field
  comparison with packed-decimal and zoned-decimal fields 573
  for table or array file 529
  format 573
  input specifications entry 571
  output specifications entry 619
binary synchronous communications (BSC) functions 323
binary synchronous communications adapter (BSCA)
  See also BSCA file
  defining 313
  device-dependent considerations 325
  programming considerations 318
  sample programs 328
  systems that use BSC 325
  translating T-SPEC 320
bit operations 719
  See also BITON, BITOF, and TESTB operations
bit testing (TESTB operation code) 825
BITOF (set bit off) operation code
  description 725
BITON (set bit on) operation code
  description 728
blank after
  *AUTO output 648
  *AUTO page headings 643
  output specifications 618
blanks
  imbedded, removing (compressing data) 318
  trailing, removing (truncating data) 319
BLDINDEX procedure 138
block length
  file description specifications entry 498
  relation to record length entry 498
branching (GOTO operation code) 768
branching to assembler-language subroutine 721
branching within RPG 720
breakpoints 64
BSC (binary synchronous communications) 314
BSCA files

addressing characters 546
configuring your system for BSC 320
translating T-SPEC to AS/400 system configuration 320
defining 314
file description specifications 314
telecommunications specifications 316
descriptions of BSC functions
receive-only function 323
send only function 324
send-and-receive function 324
device-dependent considerations
IBM 3740 data entry system 325
file description specifications 314
IBM 3750 (world trade only) 328
last file 545
multiple-file support 326
permanent-error indicator 544
polling characters 546
programming considerations
ASCII-EBCDIC character translation 318
compressing the data 318
configuring your system for BSC 320
column headings created by auto report 662
diagnostics 320
first program cycle 318
reclaim resources 320
record-available indicator 324
removing strings of embedded blanks 318
removing trailing blanks 319
truncating the data 319
sample programs
AS/400 to 3740 332
send only 329
systems that use BSC 325
telecommunications specifications 316
wait time 545
buffer for CONSOLE file, erasing 298
bypassing a KEY operation 302

calculations (continued)
operation codes
summary chart 713
specifications 585
subroutines
coding of 763
total time 6
CALL (call a program) operation code
description 730
CAsx (case) operation code
description 732
rules of comparing factor 1 with factor 2 718
xx portion of 22, 718
CFILE continuation-line option 184
CHAIN (chain) operation code
description 734
random processing 734
chained files
description 496
random processing 734
changing the contents of arrays and tables 414
character
collating sequence 461
grouping by zone or digit 569
hexadecimal values 460
in record identification code 567
unprintable 479
CL 1
Clear key
*STATUS keyword code 196
as exception 191
Cmd
See function keys
codes
edit 440
exception/error 196
operation 713
record identification 567
WORKSTN return 197
coding a data structure 429
coding an RPG program 25
coding edit words 449
coding subroutines 763
collating sequence 459
column headings created by auto report 662
overflow of print lines 663
reformatting 662
spacing and skipping 660
combined files 494
command keys 191
See also function keys
comments
See column 7 in chapters 17 through 24
common processing variations 189
communications
AS/400 to 3740 332

C

C/Z/D (character/zone/digit) entry in input
specifications 568
calculation specifications 28, 585
calculations
created by auto report 628
detail 6
factor 1 590
factor 2 596
indicators
conditioning 391
control-level 365
in AND relationship 392
resulting 372

860  System/36-Compatible RPG II User’s Guide and Reference
communications (continued)  
BSC functions 323  
sample programs 328  
send interspersed with receive 334  
send only 329  
telemcommunications specifications 316, 537  
translating T-SPEC 320  
using a BSCA file 313  
COMP (compare) operation code  
description 737  
compare and testing operations 717  
rules of comparing fields 717  
compile-time arrays  
  defining 400  
  description of 397  
  loading 402  
compile-time tables  
  defining 400  
  description of 397  
  loading 402  
compiler listing  
  browsing through using SEU 48  
  output options for AUTO procedure 683  
  output options for RPGC procedure 46  
  output options for RPGR procedure 289  
compiling  
  auto report program (AUTO procedure) 682  
  auto report source program using the CR36RPT command 674  
  interpreting 52  
  MRT program 206  
  program object for display formats 287  
  RPG program (RPGC procedure) 45  
  RPG program using the CR36RPGR command 33  
compressing data 318  
conditional branching 14  
conditioning indicators  
  conditioning calculations 391  
  conditioning output 393  
configuration (telecommunications specifications entry) 540  
consecutive processing  
  of a direct file  
    reading 116  
    updating 128  
  of a sequential file  
    reading 93  
    updating 99, 128  
  of an indexed file 141, 142  
CONSOLE file  
  buffer, erasing 298  
  changing the display format 297  
  creating display formats 287  
  creating display formats (CR36RPGR procedure) 290  
  CR36RPGR procedure 290  
CONSOLE file (continued)  
    erasing the buffer 298  
    RPG procedure 287  
    subfields 286  
    with KEYBORD and CRT files 298  
constants  
  *AUTO output specifications  
    1-9 or R in column 39 654  
    A in column 39 648  
    B or blank in columns 39 646  
    C in column 39 652  
  *AUTO page headings 640  
    on output specifications 619  
    with edit words 450  
continuation-line options 182  
  CFILE 184  
  FMTS 184  
  ID 184  
  IND 183  
  INFDS 184  
  INFSR 184  
  NUM 182  
  RECNO  
    adding records between records in a sequential file 107  
    adding records to a direct file 133  
    creating a direct file that allows deletions 113  
  SAVDS 183  
  SLN 183  
  control break 6, 365  
  control break, unwanted 368  
  control field 6, 365  
  control group 6, 365  
  control language 1  
  control language commands to run a program 63  
  control specification 26, 471  
control-level indicators  
  as field-record-relation indicators 581  
  as record-identifying indicators 557  
  assigning 366  
  description of 365  
  on calculation specifications 588  
  split control fields 370  
  with auto report  
    *AUTO output specifications 644  
    effect on group printing 651  
    with subroutines 721  
controlled loop 16  
  copy function, auto report 633  
  CR (sign for negative balance) 451  
  CR and - in edit words 451  
  created specifications, auto report 628  
  created total fields, auto report 649  
  creating a direct file  
    that allows deletions 113  
    that does not allow deletions 111
creating a sequential file 90
creating an alternative index 137
creating an indexed file 135
creating display formats for a CONSOLE file 287
creating display formats for a WORKSTN file 180
Creating Display Formats for CONSOLE Files with CRTS36RPGR 290
creating edit words with printer spacing chart 457
creating input records for arrays or tables 399
creating or changing an RPG program (RPGSEU procedure) 30
creating or changing display formats (RPGSDA procedure) 23
cross-reference listing 50
creating (RPGX procedure) 50
option on AUTOC procedure 683
option on RPGC procedure 46
CRT file
coding the specifications 310
displaying data 312
CRTS36RPG command
completing the first CRTS36RPG display 36
completing the second CRTS36RPG display 41
elements of the command file 35
entering only certain parameters 35
entering only parameter values 35
parameters, description 37
using 34
CRTS36RPT command
completing the first CRTS36RPT display 675
completing the second CRTS36RPT display 676
completing the third CRTS36RPT display 678
currency symbol
control specification entry 475
with an edit code 444
with an edit word 451
cycle, program
detailed 81
overview 6
processing matching records 346
processing WORKSTN files 211
data structures (continued)
examples (continued)
defining subfields within a field 433
reorganizing fields in an input record 435
local data area for a display station (SUBR21) 205
overlapping subfields 590
referred to in RLABL operation 813
special data structures
CFILE continuation-line option 184
file information data structure (INFDS) 194, 436
INFDS continuation-line option 184
local data area for a display station 436
SAVDS continuation-line option 183
SAVDS data structure 435
subfields 430
date created for *AUTO page headings 640
suppressing the date 640
date edit (control specification) 475
date editing with edit codes 447
date editing with edit word (example) 455
date fields 616
date format 475
date option 475
date suppression (auto report) 640
DBCS
See double-byte character set
DEBUG (debug) operation code 76, 740
control specification entry (column 15) 474
overriding in AUTOC procedure 684
overriding in RPGC procedure 47
records written by the DEBUG operation 741
debugging a program that uses a WORKSTN file 77
debugging an RPG program object 76
decimal comma 476
decimal data format
packed 572
zoned 571
decimal period 476
decimal positions
calculation specifications entry 601
created total fields (auto report) 649
extension specifications entry 529
input specifications entry 576
with move remainder (MVR) operation 801
with square root (SQRT) operation 822
defining a BSCA file 314
defining a field in calculations (result field) 597
defining arrays and tables 400
defining one area of storage more than one way 431
definition of terms 847
DEFN (field definition) operation 742
DEL entry in output specifications 608
delete-capable file 113
deleting records from a direct file 127
deleting records from a sequential file 99
deleting records from an indexed file 156
demand files
   See also READ operation code
description 497
device types and file types that can use 495
possible processing methods 500
demand WORKSTN file
   file description specifications 181
   reaching end of file 208
descriptions of BSC functions 323
designing your RPG program 11
designing the input 14
designing the output 13
designing the processing 13
detail lines (*AUTO output specifications) 644
detail records 607
detail time 6
detailed RPG program cycle 81
device entry of file description specifications 505
associated file types 505
   BSCA 509
   CONSOLE 509
   CRT 509
   DISK 508
   KEYBORD 509
   PRINTER 509
   SPECIAL 509
   WORKSTN 508
device-dependent considerations for BSCA files
   IBM 3740 data entry system 325
   blocked records 327
   multiple-file support 326
   restrictions 326
   RPG specifications 327
   single-file support 326
device, definition of 171
DFILE parameter on OCL FILE statement 113
differences between arrays and tables 397
differences between System/36 RPG II and AS/400
   System/36-Compatible RPG II 837
digit
   character grouping by 569
   portion of character used as record identification
code 568
direct files
   adding records to 133
   consecutively 128
   creating a direct file that allows deletions 113
   creating a direct file that does not allow
deletions 111
   deleting records from 127
   reading
      consecutively 116
      randomly by address output (addrout) file 124
      randomly by relative record number 94
      randomly by relative record number and/or con-
   secutively 95
direct files (continued)
   updating
      consecutively 128
      randomly by address output (addrout) file 132
      randomly by relative record number 129
      randomly by relative record number and/or con-
      secutively 132
DISK files
   direct files
      adding records to 133
      creating a direct file that allows deletions 113
      creating a direct file that does not allow
deletions 111
      deleting records from 127
      reading 116
indexed files
      adding records to 161
      creating 135
      deleting records from 156
      reading 141
      updating 156
sequential files
      adding records to 104
      creating 90
      deleting records from 99
      reading 92
      updating 98
updating 127
display formats
   creating 180
   creating or changing (RPGSDA procedure) 23
   for CONSOLE file
      changing 297
      creating (CRTS36RPGR procedure) 290
      creating (RPGR procedure) 287
      format of 294
   for WORKSTN file 180
   from IDDU data dictionaries 216
   from user-defined formats 216
   relationship between input specifications and 186
   relationship between output specifications and 188
display station
   acquiring 203
   requesting the program by 204
display station local data area 436
display station local data area (SUBR21) 205
DIV (divide) operation code
   See also MVR operation code
description 743
DO (do) operation code
description 745
   rules of comparing factor 2 with the result field 718
   with conditional indicators 745
do group 22, 718
Do Until structure 18
Do While structure 16
dollar sign in edit codes 444
dollar sign in edit words 451
double-byte character set 831
double-byte comments 833
double-byte device support 836
double-byte fields 832
double-byte literals and constants 831
double-byte tables and arrays 832
messages 836
moving double-byte data and adding control characters (SUBR41) 835
moving double-byte data and deleting control characters (SUBR40) 834
processing considerations 833
DOUxx (do until) operation code
description 748
rules of comparing factor 1 with factor 2 718
with conditional indicators 748
xx portion of 22, 718
DOWxx (do while) operation code
description 750
rules of comparing factor 1 with factor 2 718
with conditional indicators 750
xx portion of 22, 718
duplicate character value 208
dynamic array
See run-time array

E
edits
*AUTO output specifications
A in column 39 651
B in column 39 647
*AUTO page-headings specifications 643
asterisks used with 446
currency symbol used with 444
date field 447
editing numeric fields 440
effect on end position 447
examples of usage 442
inverted print 476
output specifications entry 617
summary chart of 441
zero suppress 441
edit words
*AUTO output specifications 655
& (Ampersand) 451
asterisk (fill) 450
coding for 449
constants 450
CR and 451
creating (with printer spacing chart) 457
currency symbols ($ and other) 451
examples of 452
edit words (continued)
b (Blank) 450
output specifications entry 619
zero suppress 450
editing arrays 417
editing considerations (edit words) 449
editing numeric fields 439
ELSE (else do) operation code
description 752
embedded blanks 318
dynamic array
See run-time array
end of file
CONSOLE file 294
file description specifications entry 497
WORKSTN file 179, 208
END operation code
description 753
END/CASxx 753
END/DO 753
END/DOUxx 753
END/DOWxx 753
END/IFxx 753
end position in output record, *AUTO output specifications
A in column 39 652
B in column 39 648
ENDSR (end subroutine) operation code
description 754
in INFSR subroutine 199
entering and compiling an RPG program 33
entering source statements
RPGSEU procedure 30
ERASE entry, calculation specifications 597
erasing the CONSOLE file buffer 298
error codes, INFDS subroutine
*STATUS keyword 196
WORKSTN return codes 197
error handling
description 191
INFDS data structure 194
INFSR subroutine 198
error, definition of 191
exception output records 607
See also EXCPT operation code
exception, definition of 191
exception/error handling
description 191
INFDS data structure 194
INFSR subroutine 198
EXCPT (exception output) operation code 755
EXCPT group name 617
definition 850
EXCPT names 617
execute subroutine (EXSR) operation code 763
EXIT (exit to an external subroutine) operation code 761
See also RLABL (RPG label) operation code
EXSR (execute subroutine) operation code
  description  763
extension specifications  27, 517
external indicators
  as file-conditioning indicator  514
  as output indicators  611
description  374
setting and restoring (SUBR20)  204
used to condition calculations  391
used to condition output  393

F
factor 1
  description  590
  figurative constant  595
  in arithmetic operations  716
  in LOKUP operation  774
  literals  594
factor 2
  description  596
  in arithmetic operations  716
fetch overflow routine  269
field description entries on input specifications  571
field indicators  371
field length
  arithmetic operations  716
  calculation specifications  598
  COMP (compare) operation code  737
  input specifications  575
  key field  501
  output specifications  618
field location, input specifications  575
field name
  created by auto report  647
  in RLABL operation  813
  input specifications  576
  in OR relationships  575
  output specifications  612
  result field, calculation specifications  597
  rules for  612
  special words  612
field-record-relation indicators
  assigning  382
  description  381
  input specifications entry  581
figurative constants  595
file
  address output (addrout)
    reading a direct file by  96
    reading a sequential file by  96
    reading an indexed file by  153
    updating a direct file by  132
    updating a sequential file by  104
    updating an indexed file by  161
array  496

file (continued)
  BSCA  313
  chained  496
  combined  494
  CONSOLE  283
  CRT  310
demand  497
direct  110
  DISK  89
  full-procedural  496
  indexed  135
input  494
  KEYBORD  298
output  494
primary  341
  PRINTER  259
record address  496
secondary  341
sequential  89
  SPECIAL  277
table  496
update  494
  used by RPG II  5
  WORKSTN  171
file addition
  direct file  133
  file description specifications entry  513
indexed file
    randomly by key field with chaining  162
    randomly by key field without chaining  165
    sequentially by key field  165
sequential file
  at the end of a file  104
  between records in a file  107
file condition  514
FILE control language statement
  adding records to a direct file  133
  adding records to a sequential file  104
  adding records to an indexed file  161
  creating a direct file that allows deletions  113
  creating a sequential file  90
  creating an alternative index for an indexed file  137
deleting records from a direct file  127
deleting records from a sequential file  99
deleting records from an indexed file  156
  updating DISK files in a MRT program  206
file description charts  483
file description specifications  27, 483
file designation
  See also chained file, demand file, full-procedural
  file, primary file, record address file, secondary file,
  and table file
  file description specifications entry  494
file format  498
file information data structure (INFDS)  191
file organization
   See also direct files, indexed files, and sequential files
   DISK files 89
   file description specifications entry 504
file processing methods
   See also consecutive processing, random processing, and sequential processing
   file description specifications entry 500
file sharing 477
file translation 466
file type
   See also combined file, input file, output file, and update file
   file descriptions specifications entry 493
file-conditioning indicators 381
filename
   extension specifications
      from filename 520
to filename 521
   input specifications 550
   line counter specifications 533
   output specifications 605
   telecommunications specifications 540
files that allow deletion of records 113
first RPG program cycle 10
first-page forms position 478
first-page indicator 375
fixed currency symbol
   with an edit code 445
   with an edit word 451
fixed-length format 498
floating currency symbol
   with an edit code 444
   with an edit word 451
flowchart
   detailed program cycle 81
   handling exceptions and errors in WORKSTN files 193
   overview of program cycle 8
   processing WORKSTN input files 214
   program cycle for WORKSTN files 211
FMTS continuation-line option 184
FORCE (force) operation code
   description 765
form length, line counter specifications 534
form type
   See column 6 in chapters 17 through 24
format of specifications created by auto report 628
format, data 571
format, date 475
format, display
   for CONSOLE file 287
   for WORKSTN file 180
format, file 498
FREE (deactivate a program) operation code
   description 767
from filename, extension specification 521
full arrays and tables 525
full-procedural files
   file description specifications entry 496
   reading a direct file 123
   reading a sequential file 94
   reading an indexed file 151
   updating a direct file 132
   updating a sequential file 103
   updating an indexed file 161
function keys
   as exceptions 191
   corresponding indicators 388
   with a KEYBORD file 308
   with a WORKSTN file 190
function-key indicators 388
G
GOTO (branch to) operation code
   See also TAG operation code
   description 768
group indication, auto report 692
group printing, auto report 655
grouping characters by zone and digit 569
H
H-*AUTO record description 640
half-adjust 600
halt indicators 389
halt messages 63
handling exceptions and errors 191
handling overflow 265
header (control) specification 471
heading records, output specifications 607
Help key
   *STATUS keyword code 196
      as exception 191
hexadecimal value of characters
   changing the collating sequence 461
duplicate character values 208
   translating a file 466
Home key
   *STATUS keyword code 196
      as exception 191
I
IBM 3740 data entry system 325
IBM 3750 328
IBM-written subroutines
   SUBR01 (system input for SPECIAL files) 279
   SUBR20 (setting and restoring external indicators) 204
IBM-written subroutines (continued)
   SUBR21 (reading and writing the local data are for a
   display station) 205
   SUBR23 (retrieving messages) 815
   SUBR95 (inline inquiry) 477
ICF (Intersystem Communications Function)
   See also Intersystem Communications Function
   (ICF)
   compared with batch BSC feature 313
ICF-library name 42
ID continuation-line option 184
identification
   of programs 480
   of record types 567
ideographic data
   See double-byte character set
If-Then-Else structure 14
IFxx (if then) operation code
description 14, 771
rules of comparing factor 1 with factor 2 718
with conditioning indicators 771
xx portion of 22, 718
IND continuation-line option 183
index search, description of xxviii
index, array 407
indexed files
   adding records to
      randomly by key field 162
   alternative indexes for
      creating 137
      using 141
   creating 135
   deleting records from 156
reading
   randomly and/or sequentially by key field 151
   randomly by address output (addrout) file 153
   randomly by key field 150
   sequentially by key field 142
   sequentially within key-field limits 143
updating
   randomly and/or sequentially by key field 161
   randomly by address output (addrout) file 161
   randomly by key field 158
   sequentially by key field 157
   sequentially within key-field limits 158
indicators
conditioning 381
conditioning calculations 391
control-level 365
defined on RPG specifications 355
field 371
field-record-relation 381
file-conditioning 381
first-page 375
indicators (continued)
   function-key 388
   halt 389
   internal 375
   last-record 377
   level-zero 385
   matching-record 379
   overflow 355
   permanent-error 544
   record-available 545
   record-identifying 357
   resulting 372
INFDS continuation-line option 184
INFDS data structure
   coding 194
   handling exceptions and errors 191
INFSR continuation-line option 184
INFSR subroutine
   coding 198
   handling exceptions and errors 191
   inline inquiry subroutine (SUBR95) 477
input and output, programmed control of 721
input file 494
input for AUTO report 625
input record
   number allowed in a sequenced group 555
   reorganizing fields in 435
input specification 28, 547
   field description entries 571
   file and record-type identification entries 548
inquiry
   accounts receivable (sample program AR230R) 221
   inline (SUBR95) 477
inserting new records
   See adding records to a file
intermediate block checking (ITB)
   telecommunications specifications entry 543
internal indicators
   first-page indicator 375
   last-record indicator 377
   matching-record indicator 379
Intersystem Communications Function (ICF)
   alternative to batch BSC feature 313
   using a WORKSTN file 171
inverted print 476
ITB (intermediate block checking)
   telecommunications specifications entry 543
K
K edit code created by auto report
   A in column 39 651
   B in column 39 647
KA through KN, KP through KY (function-key indicators)
description 388
used with a KEYBORD file 308
KA through KN, KP through KY (function-key indicators) (continued) used with a WORKSTN file 190

KEY (key) operation code
bypassing 302
calculation specifications for 299
description 773
examples of 301
using 302
with SET operation 309

key field
adding records randomly by 162
length of 501
limits
reading sequentially within 143
updating sequentially within 158
part of index entry 135
reading randomly and/or sequentially by 151
reading randomly by 150
reading sequentially by 142
starting location 504
updating randomly and/or sequentially by 161
updating randomly by 158
updating sequentially by 157

KEYBORD file 298
allowing function keys to be pressed 308
KEY operation 299
bypassing 302
calculation specifications for 299
using 302
SET operation 304
using a message member 303

keys, non-contiguous
file description specifications 139
in alternative index file 137

keywords
*CANCL 200
*DETC 200
*GETIN 200
*INP 197
*MODE 197
*OPCODE 197
*OUT 197
*RECORD 197
*SIZE 197
with the POST operation 210
*STATUS 197

kinds of arrays and tables 397

L
L0 (level-zero) indicator 385
label exit 510
last program cycle 10
last-record indicator 377

level-zero indicator 385
limits record 145
limits, key field
reading sequentially within 143
updating sequentially within 158
line counter specifications 27, 531
link editing
option on AUTOC procedure 685
linking to assembler subroutines
EXIT operation code 761
for SPECIAL file 278
RLABL operation code 813

literals 594
load module
See program object
loading arrays
compile-time 402
prerun-time 403
run-time 404
loading tables
compile-time 402
prerun-time 403
local data area for a display station, reading and writing (SUBR21) 205

LOKUP (lookup) operation code
array LOKUP 774
description 774
table LOKUP 774
look-ahead field 558
lookup operation (LOKUP operation code) 774
LR (last-record) indicator 377

M
main storage
size to compile 472

match fields
assigning matching-record indicators to 379
coding 341
in primary and secondary files 341
input specifications entry 579
rules for coding 343

matching-record indicator 379
matching-records, processing 346
message identification codes (MIC)
using 209
message-retrieving subroutine (SUBR23) 815

messages
description 62
displaying and printing 63
using 61

MHHZO (move high to high zone) operation code
description 776

MHLZO (move high to low zone) operation code
description 776
MIC (message identification code) using 209
MLHZO (move low to high zone) operation code description 776
MLLZO (move low to low zone) operation code description 777
mode of processing 500
modifier statement, auto report 634
MOVE (move) operation
description 778
move remainder (MVR) operation
description 801
move zone operations 717
MOVEA (move array) operation code
description 781
MOVEL (move left) operation code
description 797
moving double-byte data and adding control characters (SUBR41) 835
moving double-byte data and deleting control characters (SUBR40) 834
MRT (multiple requester terminal) program
acquiring a display station 203
compiling 206
reaching end of file 208
releasing a display station 203
requesting the program by one or more display stations 204
running 206
updating DISK files in 206
using 201
MULT (multiply) operation code
description 800
multiple-file processing 341
multiple-index files 137
MVR (move remainder) operation code
description 801

N
N (not)
calculation specifications 588
input specifications 567
output specifications 611
name
array 522
field
factor 1 on calculation specifications 590
factor 2 on calculation specifications 596
input specifications 576
result field on calculation specifications 597
program 480
table 522
name of label exit 510
negative balance
edit codes 440
negative balance (continued)
edit words 451
negative indicator 611
negative numbers 571
negative square root 822
never-ending program
option on AUTOCo procedure 683
option on RPGC procedure 46
specified with CRTS36RPG command 38
NEXT (next) operation code
description 802
in MRT program 202
nonprint characters 479
normal collating sequence 459
NUM continuation-line option 182
number of entries per record 524
number of entries per table or array 525
number of lines per page 533
numbering report pages (PAGE special word)
input specifications 577
output specifications 612
numeric fields
data format 571
moving 778
numeric literals 594

O
OE140R, sample program (entering orders from customers) 245
OE400R, sample program (updating an indexed file randomly by key field) 158
OP field
in SUBR20 204
in SUBR21 205
operating system (OS/400) 1
operation codes 713
*LIKE DEFN (field definition) 742
ACQ (acquire) 722
ADD (add) 723
BEGSR (begin subroutine) 724
BITOF (set bit off) 725
BITON (set bit on) 728
CALL (call a program) 730
CASxx (case) 732
CHAIN (chain) 734
COMP (compare) 737
DEBUG (debug) 740
DIV (divide) 743
DO (do) 745
DOUxx (do until) 748
DOWxx (do while) 750
ELSE (else do) 752
END (end) 753
ENDSR (end subroutine) 754
EXCPT (exception output) 755
operation codes (continued)
EXIT (exit to an external subroutine) 761
EXSR (execute subroutine) 763
FORCE (force) 765
FREE (deactivate a program) 767
GOTO (branch to) 768
IFxx (if/then) 771
KEY (key) 773
LOKUP (lookup) 774
MHHZO (move high to high zone) 776
MLHZO (move low to high zone) 776
MLLZO (move low to low zone) 777
MOVE (move) 778
MOVEA (move array) 781
MOVEL (move left) 797
MULT (multiply) 800
MVR (move remainder) 801
NEXT (next) 802
PARM (identify parameters) 803
PLIST (identify a parameter list) 805
POST (post) 806
READ (read) 807
READE (read equal key) 809
READP (read prior record) 810
REL (release) 811
RETRN (return to caller) 812
RLABL (RPG label) 813
SET (set) 817
SETLL (set lower limits) 818
SETOF (set off) 819
SHTDN (shutdown) 820
SORTA (sort an array) 821
SQRT (square root) 822
SUB (subtract) 823
TAG (tag) 824
TESTB (test bit) 825
TESTZ (test zone) 827
TIME (time of day) 828
XFOOT (summing the elements of an array) 829
Z-ADD (zero and add) 829
Z-SUB (zero and subtract) 829
output (continued)
detail
output specifications entry 607
exception 607
heading 607
part of program cycle 6
total
output specifications entry 607
output fields
end position 618
name 612
repeating with *PLACE 613
output files 494
output indicators 611
output records
detail 607
end position 618
exception 607
heading 607
total 607
output specifications 28, 603
created by auto report 629
output, indicators conditioning 393
overflow
automatic 265
fetch 269
handling 265
line 534
using indicators 265
overflow indicators 355
overflow line 534
overlapping data structure subfields 590
overriding fields in a display format 209
overriding the debug option in the source program
AUTOC procedure 684
RPGC procedure 47
overriding the print option in the source program
AUTOC procedure 684
RPGC procedure 47
overriding the size-to-execute option in the source program
AUTOC procedure 684
RPGC procedure 47
overview of auto report 624
overview of program cycle 6

P
packed-decimal format
input specifications 572
record address type 502
page (columns 1 and 2) 472
page headings, auto report
placement of 662
reformatting 662
page numbering (PAGE, PAGE1-PAGE7)
input specifications entry  577
output specifications entry  612
PARM (identify parameters) operation code
description  803
permanent-error indicator
telecommunications specifications  544
physical file  137
placement of headings and fields, auto report  662
PLIST (identify a parameter list) operation code
description  805
polling characters  546
positioning printer forms  478
positions 23 through 26 of INFDS data structure  197
POST (post) operation code
description  806
with WORKSTN file  210
prerun-time arrays and tables
defining  400
description of  397
loading  403
primary files
file description specifications entry  495
used with secondary files  341
Print key
*STATUS keyword code  196
as exception  191
print option in source program
overriding in AUTOC procedure  684
overriding in RPGC procedure  47
print, inverted  476
PRINTER files
description  259
file description specifications  259
handling overflow  265
automatic overflow  265
fetch overflow routine  269
overflow indicators  265
line counter specifications  260
output specifications  261
AND and OR lines  263
field-description entries  263
file and record-identification entries  261
sample program  270
spacing and skipping  269
printing an RPG cross-reference listing (RPGX procedure)  49
procedures
AUTOC  682
CRTS36RPGR  290
RPGC  45
RPGR  287
RPGSDA  23
RPGEU  30
RPGX  49
procedures, System/36 RPG II  4
processing methods  500
processing the duplicate character value  208
processing, multfile  341
program cycle
detailed  81
for a WORKSTN file  211
flowchart  211
processing of input files  214
overview  6
flowchart  8
program identification  480
program object (load module)
compiling
AUTOC procedure  682
CRTS36RPG command  33
RPGC procedure  45
RPGR procedure  287
debugging  76
running  63
programmed control of input and output  721
programming aids, auto report  670
programming operation codes, structured  22, 718
programming, structured  22
PROMPT control language statement  201
R
random processing
adding records by key field with chaining  162
adding records by key field without chaining  165
definition of  94
do
random processing *(continued)*
  reading by key field and/or sequentially 151
RCODE field
  for SUBR20 204
  for SUBR21 205
reaching end of file 179
reaching end of file for a MRT program
demand file 208
  primary file 208
READ (read) operation code
description 807
read under format 201
READE (read equal key) operation code
description 809
reading a direct file
  consecutively 116
  randomly by address output (addrout) file 124
  randomly by relative record number 119
  randomly by relative record number and/or consecutively 123
reading a sequential file
  consecutively 93
  randomly by address output (addrout) file 96
  randomly by relative record number 94
reading an indexed file
  randomly and/or sequentially by key field 151
  randomly by address output (addrout) file 153
  randomly by key field 150
  sequentially by key field 142
  sequentially within key-field limits 143
reading and writing the local data area for a display station
  (SUBR21) 205
reading data from a display shown by a previous program 201
READP (read prior record) operation code
description 810
receive-only BSC function 323
RECN0 continuation-line option
  adding records to a direct file 133
description 510
record addition
  file description specifications entry 513
  RECN0 continuation-line option 510
output specifications entry 608
to a direct file 133
to a sequential file
  at the end of a file 105
  between records in a file 107
to an indexed file
  randomly by key field with chaining 162
  randomly by key field without chaining 165
  sequentially by key field 165
record address file
  See also address output (addrout) file
description 496
record address file *(continued)*
  reading a direct file randomly by 124
  reading a sequential file randomly by 96
  reading an indexed file randomly by 153
  summary charts 485
  updating a direct file randomly by 132
  updating a sequential file randomly by 104
  updating an indexed file randomly by 161
record address type 502
record identification code
  for CONSOLE file 288
  input specifications entry 567
record length 498
record type
  See record identification code
record-description specifications
  *AUTO output specifications 644
  *AUTO page-heading specifications 640
  input specifications 548
  output specifications 603
record-identifying indicators
  AND relationship 358
  conditioning calculations 391
  conditioning output 393
description 357
  OR relationship 359
records written by the DEBUG operation 77, 741
referring to a data structure 813
referring to a field 813
referring to an array or table 813
referring to an indicator 813
REL (release) operation code
description 811
  used with a WORKSTN file 203
related arrays and tables
  creating 399
  defining separately or in alternating format 398
description of 397
  searching 411
relative record number
definition of 94
direct file 119
  reading randomly by
    sequential file 94
  reading randomly by and/or consecutively
    direct file 123
    sequential file 95
  updating randomly by
    direct file 129
    sequential file 103
  updating randomly by and/or consecutively
    direct file 132
    sequential file 103
releasing a display station (REL operation code)
description 811
  in a MRT program 203
remainder, move (MVR operation code) 801
removing embedded blanks (compressing data) 318
removing trailing blanks (truncating data) 319
repeating an operation
Do Until structure 18
Do While structure 16
repeating output fields (*PLACE) 613
REPLACE parameter 42
result field
field name, table name, array name, array element,
or data structure 597
resulting indicators 372
RETRN (return to caller) operation code
description 812
RLABL (RPG label) operation code
description 813
referring to a data structure 813
referring to a field 813
referring to an array or table 813
referring to an indicator 813
Roll Down key
*STATUS keyword code 196
as exception 191
Roll Up key
*STATUS keyword code 196
as exception 191
rolling totals, auto report 644
rounding numbers in the result field (half-adjust) 600
RPG halt messages 63
RPG procedures
AUTOC 682
CRTS36RPGR 290
RPGC 45
RPGR 287
RPGSDA 23
RPGSEU 30
RPGX 49
RPG program cycle
detailed 81
effect of indicators on 353
overview 6
RPGC procedure
calling 45
first RPGC display 45
second RPGC display 47
using System/36 Environment 45
RPGSDA procedure 23
RPGSEU 30
RPGX procedure 49
run-time array
defining 400
description of 397
loading 404
running an RPG program object 63

S
sample programs
AR230R (inquiring into an accounts receivable
file) 221
AR330R (maintaining an accounts receivable
file) 226
AR935R (requesting a printout of accounts receivable)
241
AR936R (printing accounts receivable) 270
auto report 686
BSCA file programs
AS/400 to 3740 332
send interspersed with receive 334
send only 329
DISK file programs
adding records at the end of a sequential
file 105
adding records between records in a sequential
file 109
adding records to an indexed file sequentially by
key field 167
creating a direct file that allows deletions 115
creating a direct file that does not allow
deletions 112
creating a sequential file 91
creating an alternative index 139
creating an indexed file 137
deleting records from a sequential file 100
randomly by key field with chaining 164
reading a direct file consecutively 117
reading a direct file randomly by relative record
number 119
updating a direct file randomly by relative record
number 129
updating an indexed file randomly by key
field 158
updating and deleting records from a sequential
file 100
using an alternative index 141
OE140R (entering orders from customers) 245
OE400R (updating an indexed file randomly by key
field) 158
PRINTER file program 270
requesting a printout of accounts receivable
(AR935R) 241
WORKSTN file programs
entering orders from customers (OE140R) 245
inquiring into an accounts receivable file
(AE230R) 221
maintaining an accounts receivable file
(AE330R) 226
SAVDS continuation-line option 183
SAVDS data structure 435
screen design aid (SDA)
creating displays for a WORKSTN file 180
screen design aid (SDA) (continued)
  RPG procedure (RPGSDA) 23
search word for LOKUP operation 774
searching an array 409
searching arrays and tables 408
searching one table 411
searching related tables 411
secondary files
  file description specifications entry 496
  used with a primary file 341
send interspersed with receive 324, 545
send-and-receive BSC function 324
  receive a file, then send a file 324
  send a file, then receive a file 324
  send records interspersed with receive records 324
send-only BSC function 324
sequence checking
  file description specifications entry 497
  input specifications entry 550
sequence, collating 459
sequential files 89
  adding records to 104
    at the end of a file 105
    between records in a file 107
  creating 90
  deleting records from 99
  randomly by address output (addrout) file 96
  reading
    consecutively 93
    randomly by relative record number 94
    randomly by relative record number and/or consecutively 95
  updating
    consecutively 99
    randomly by address output (addrout) file 104
    randomly by relative record number 103
    randomly by relative record number and/or consecutively 103
sequential processing of indexed files
  adding records 165
  reading by key field 142
  reading by key field and/or randomly 151
  reading within key-field limits 143
  updating by key field 157
  updating by key field and/or randomly 161
  updating within key-field limits 158
SET (set) operation code
  description 817
  with a KEYBORD file 304
  with the KEY operation code 310
SETLL (set lower limits) operation code
  description 818
  reading an indexed file 147
SETOF (set off) operation code
  description 819
SETON (set on) operation code
  description 819
SETON and SETOF operations 720
setting and restoring external indicators (SUBR20) 204
SEU split display 48
shift-in (S/I) character 831
shift-out (S/O) character 831
short arrays and tables 520, 525
SHTDN (shut down) operation code
  description 820
sign (positive or negative)
  binary format 573
  packed-decimal format 572
  zoned-decimal format 571
similarities between arrays and tables 397
single requester terminal (SRT) program 201
size to compile 472
skip after 610
skip before 610
skipping
  description 269
  output specifications entry 610
  with auto report 660
SLN continuation-line option 183
SORTA (sort an array) operation code
  description 821
source entry utility
  browsing through a compiler listing 48
  change an RPG II program 29
  enter an RPG II program 29
source program
  compiling
    AUTOC procedure 682
    CRTS36RPG command 33
    CRTS36RPT command 674
    RPGC procedure 45
  creating or changing (RPGSEU procedure) 30
  entering
    RPGSEU procedure 30
space after 610
space before 610
spacing
  description 269
  output specifications entry 610
  with auto report 660
special data structures
  CFILE continuation-line option 184
  file information data structure 194, 436
  INFDS continuation-line option 184
  local data area for a display station 436
  local data area for a display station (SUBR21) 205
  SAVDS continuation-line option 183
  SAVDS data structure 435
special device support 277
SPECIAL file
  restrictions for 278
SPECIAL file (continued)
  SUBR01  279
    using a subroutine for input and output  278
    your own subroutine  280
special words
  *PLACE  613
    PAGE, PAGE1-PAGE7  547, 612
    UDATE, UDAY, UMONTH,UYEAR  616
specifications
  calculations  585
  control  471
  description  25
  extension  517
  file description  483
  input  547
  line counter  531
  option  629
  output  603
specifications created by auto report  628
specifying a MRT program  206
split control fields  370
SQRT (square root) operation code  822
description  822
SR entry on calculation specifications  587
SRT (single requester terminal) program  201
SRyzzz (subroutine name for SPECIAL file)  278
steps in using a WORKSTN file  180
structure programming  22
  conditional branching  14
description  14
  repeating an operation  16
  sequential operation  14
structured programming operation codes
  CASxx (case)
description  732
  rules of comparing factor 1 with factor 2  718
  with conditional indicators  732
  xx portion of  22, 718
  DO (do)
description  745
  rules of comparing factor 2 with the result
    with conditional indicators  745
  DOUxx (do until)
description  18, 748
  rules of comparing factor 1 with factor 2  718
  with conditional indicators  748
  xx portion of  22, 718
  DOWxx (do while)
description  16, 750
  rules of comparing factor 1 with factor 2  718
  with conditional indicators  750
  xx portion of  22, 718
  ELSE (else do) operation code
description  752
  END operation code
description  753
System/36, differences from AS/400 system 837

T

T-*AUTO, auto report 644
table LOKUP 774
table name 522
table or array files 496
tables
See also arrays
adding entries to 415
alternating
definition of 399
extension specifications entry 530
as factor 1 590
changing the contents of
permanently 415
compile-time
definition of 397
loading 402
order in source program 402
creating input records for 399
decimal positions 529
defining 400
differences between tables and arrays 397
element 397
example of using tables 426
extension specifications 517
filename (when required) 492
kinds of 397
length of entry 526
loading
compile-time 402
placement in source program 402
prerun-time 403
LOKUP operation 774
maximum allowed for each program 517
name
as factor 2 596
as field name in *AUTO output 650
as field name in *AUTO page headings 642
as result field 597
extension specifications entry 522
output specifications entry 612
number of entries per table or array 525
packed-decimal format 529
prerun-time
defined in alternating format 399
defined separately 398
definition of 397
loading 403
referring to in RLABL operation 813
related 397
searching
description 408
one table 411
tables (continued)
searching (continued)
related tables 411
sequence (extension specifications entry) 529
short 520, 525
similarities between tables and arrays 397
writing 416
TAG (tag) operation code 824
techniques for efficient coding 14
telecommunications specifications 27, 537
  translating to AS/400 system configuration 320
  with BSCA file 313
TESTB (test bit) operation code
description 825
testing an RPG program 61
testing and comparing fields 717
  rules of comparing fields 717
TESTZ (test zone) operation code
description 827
TIME (time of day) operation code
description 828
TNAME field
for SUBR20 204
for SUBR21 205
to filename (extension specifications) 521
total fields, auto report
  asterisk indication 651
  description 649
  resetting to zero 651
total operations 6
total output records 607
total rolling, auto report 644
total time in program cycle 6
traces 68
trailing blanks, removing 319
translating a file
coding the records that translate a file 467
coding the translation 466
example 468
translation table and alternating collating sequence
coding sheet 461
transparent literal 480
truncating data (removing trailing blanks) 319

U

UDATE 616
UDAY 616
UMONTH 616
update files 494
updating a direct file
  consecutively 128
  deleting records 127
  randomly by address output (addrout) file 132
  randomly by relative record number 129
  randomly by relative record number and/or consecutively 132
updating a sequential file
  consecutively 99
  deleting records 99
  randomly by address output (addrout) file 104
  randomly by relative record number 103
  randomly by relative record number and/or consecutively 103
updating an indexed file
  deleting records 156
  randomly and/or sequentially by key field 161
  randomly by address output (addrout) file 161
  randomly by key field 158
  sequentially by key field 157
  sequentially within key-field limits 158
updating DISK files in a MRT program
  avoiding 207
  possible errors 206
using a BSCA file 313
using a CONSOLE file 283
using a CONSOLE file with KEYBORD and CRT files 298
using a CRT FILE 310
using a DISK file 89
using a KEY operation 302
using a KEYBORD file 298
using a message member 303
using a MRT program 201
using a PRINTER file 259
using a SPECIAL file 277
using a SRT program 201
using a subroutine for input and output 278
using a WORKSTN file 171
using an alternate index 141
using an array name and index 407
using arrays and tables 397
using auto report 624
using CRTS36RPG to compile an RPG source program 33
Using CRTS36RPGR to create display formats for CONSOLE files 290
using CRTS36RPRT to compile an auto report source program 674
using data structures 429
using displays with a CONSOLE file 294
using double-byte data 831
using function keys
  with a KEYBORD file 308
  with a WORKSTN file 190
using IBM's subroutine, SUBR01 279
using indicators 353
using message identification codes 209
using one or more display stations 201
using primary and secondary files 341
using the DEBUG operation 76
using the POST operation 210

using the SET and KEY operations together 309
using your own subroutine 280
using, displaying, and printing messages 61
UYEAR 616

W
wait time for BSCA file 545
when arrays and tables can be loaded 397
WORKSTN files
  ACQ operation 204
  acquiring one or more devices by the program 203
  advanced topics 208
  CFILE continuation-line option 184
  coding the INFDS data structure 194
  coding the INFSR subroutine 198
  coding the RPG specifications 181
  common processing variations 189
  compiling and running a MRT program 206
  continuation-line options 182
    CFILE 184
    FMTS 184
    ID 184
    IND 183
    INFDS 184
    INFSR 184
    NUM 182
    SAVDS 183
    SLN 183
  creating the display formats 180, 183
  end of file
    MRT program 208
    reaching 179
  exception/error-processing subroutine (INFSR) 198
  file information data structure (INFDS) 194
  FMTS continuation-line option 184
  function keys
    *STATUS keyword codes 196
    as exceptions 191
  handling exceptions and errors 191
  how WORKSTN files are processed 211
  INFDS continuation-line option 184
  INFDS data structure 194
  *INP keyword 197
  *MODE keyword 197
  *OPCODE keyword 197
  *OUT keyword 197
  *RECORD keyword 197
  *SIZE keyword 197
  *STATUS keyword 196
  NEXT operation code 202
  overriding fields in a display format 209
  processing the duplicate character value 208
  program cycle 211
  reaching end of file 179
  reaching end of file for a MRT program 208
WORKSTN files (continued)
read under format 201
reading and writing the local data area for a display station 205
reading data from a display shown by a previous program 201
REL (release) operation code 203
releasing a display station 203
requesting the program by one or more display stations 204
sample programs 221
SAVDS continuation-line option 183
setting and restoring external indicators (SUBR20) 204
SLN continuation-line option 183
steps in using a WORKSTN file 180
SUBR20 (setting and restoring external indicators) 204
SUBR21 (reading and writing the local data area for a display station) 205
updating DISK files in a MRT program 206
using a MRT program 201
using a SRT program 201
using function keys 190
using message identification codes 209
using one or more display stations 201
using the POST operation 210
WORKSTN input file processing 214
WORKSTN operations 721
WORKSTN return codes 197
writing arrays and tables 416

X
X edit code 440
XFOOT (summing the elements of an array) operation code 829

Y
Y edit code 440, 447

Z
Z (zone portion of character) 568
Z-ADD (zero and add) operation code description 829
Z-SUB (zero and subtract) operation code description 829
zero suppression
  with edit codes 440
  with edit words 450
zone
  character grouping by 569
  move operation codes 776

zoned-decimal format 571
Communicating Your Comments to IBM

IBM Application System/400
System/36-Compatible RPG II
User's Guide and Reference
Publication No. SC09-1818-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 416-448-6161
  - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
  - Internet: torrcf@vnet.ibm.com
  - IBMLink: toribm(torrcf)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwrt9)
Readers' Comments — We'd Like to Hear from You

IBM Application System/400
System/36-Compatible RPG II
User's Guide and Reference
Publication No. SC09-1818-00

Overall, how satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th>Overall satisfaction</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

How satisfied are you that the information in this book is:

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Complete</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Easy to find</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Well organized</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.