CA IDMS - 19.0
Using CA Culprit for CA IDMS

Date: 15-Jan-2018
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
# Table of Contents

**What You Should Know Before You Begin** .................................................. 14
What a File Description Tells You ........................................................................ 14
How Information Is Organized in a Computer .................................................... 15
How to Use Information in a File Description ................................................... 16

**Building and Using Data Tables** ................................................................. 17
Creating Tables ................................................................................................. 17
   How to Do It ................................................................................................. 18
   Demonstration (1) Creating a Table from a Sequential File ............................ 20
   Demonstration (2) Creating a Table from the Database ................................. 21
   Demonstration (3) Creating a Table from an Existing Table .......................... 22
   Demonstration (4) Creating a Totals-only Table .......................................... 23
Retrieving Data Tables ...................................................................................... 24
   How to Do It ................................................................................................. 24
   Demonstration 12 .......................................................................................... 25
Modifying Data Tables ....................................................................................... 25
   How to Do It ................................................................................................. 26
   Demonstration (1) Adding Rows to an Existing Table .................................... 26
   Demonstration (2) Replacing Rows of an Existing Table ............................... 27
   Demonstration (3) Deleting a Table ............................................................. 28
   Demonstration (4) Regenerating a Table ...................................................... 29
Consolidating Tables .......................................................................................... 31
   How to Do It ................................................................................................. 31
   Demonstration 13 .......................................................................................... 33

**How Totals Processing Works** ..................................................................... 35
The Extract Processing Phase (CULL) ............................................................... 38
The Output Phase (CULE) ................................................................................ 38
The Processing Steps when an Extract File Is Read .......................................... 39

**How to Produce a CA Culprit Report** ......................................................... 40
Step 1. Planning the Report .............................................................................. 40
CA Culprit Reports .................................................................................................................. 42
Add Programming Logic to a Report ...................................................................................... 42
Code Programming Logic ........................................................................................................ 42
Use Counters ........................................................................................................................ 43
  Use Conditional Counters .................................................................................................. 43
  Use Automatic Counters .................................................................................................. 43
Example 1: Print the Count of each Occurrence .................................................................. 43
Example 2: Print a Total Count ......................................................................................... 44
Example 3: Use the Value of a Work Field ......................................................................... 45
Perform Arithmetic Operations .......................................................................................... 46
  Perform Arithmetic Operations ....................................................................................... 46
Direct Processing Flow ......................................................................................................... 48
  Example: Code a Repeated Procedure .......................................................................... 48
Additional Standard Options ................................................................................................. 49
Produce Multiple Reports .................................................................................................. 49
  Example: Produce Multiple Reports in a Single Run ....................................................... 50
Sequence Report Output ..................................................................................................... 51
  How to Do It .................................................................................................................... 51
  Example: List Employees in Alphabetical Order .............................................................. 51
Use Selective Processing ........................................................................................................ 52
  How to Do It .................................................................................................................... 52
  Example 1: Select Records with Single Criteria ............................................................... 53
  Example 2: Select Records with Multiple Criteria ........................................................... 54
Assign Information to Multiple Detail Lines ....................................................................... 55
  How to Do It .................................................................................................................... 55
  Example: Use Multiple Detail Lines .............................................................................. 55
Control Total Lines ................................................................................................................ 56
Selectively Print Total Amounts .......................................................................................... 56
  Example .......................................................................................................................... 57
Label Total Lines .................................................................................................................. 58
  Example .......................................................................................................................... 58
Generate Only Total Amounts ............................................................................................. 59
  Example .......................................................................................................................... 59
Suppress All Totals ................................................................................................................ 60
  Example .......................................................................................................................... 60
Generate Subtotals ................................................................................................................ 61
  Example .......................................................................................................................... 62
Select Subtotal and Total Line Placement ................................................................. 63
   Example ................................................................................................................. 64
Selectively Print Output Lines ................................................................................... 65
   Print Output Lines Using Type 8 Parameters ......................................................... 66
   Example ................................................................................................................. 66
   Print Output Lines Using a Work Field .............................................................. 67
   Example ................................................................................................................. 67
Obtain Sort Field Values .......................................................................................... 69
   Example ................................................................................................................. 69
Obtain Sorted Work Field Values .............................................................................. 70
   Example ................................................................................................................. 71
Enhance the Basic Report ......................................................................................... 73
Add a Title to a Report .............................................................................................. 73
   Add a Title ............................................................................................................. 73
   Example: Add a Title to a Report ........................................................................ 73
Add Column Headings ............................................................................................. 74
   How to Do It ........................................................................................................ 74
   Example 1: Format Multiple Line Column Headings ....................................... 75
   Example 2: Add Subtitles .................................................................................. 76
Modify Column Widths ............................................................................................ 78
Format Numeric Data ............................................................................................... 79
   Example 1: Use Formatting Codes .................................................................... 79
   Example 2: Use Formatting Patterns .......................................................... 80
Adjust Line Length and Page Depth ........................................................................ 81
   Example 1: Specify Output Parameters for a Report ................................... 81
Generate a Basic Report from Standard Files ............................................................ 83
   Locating the Data ............................................................................................... 83
   Coding the Report ............................................................................................. 83
   How to Define the Input .................................................................................... 84
   How to Define the Output .................................................................................. 84
Generate Reports From Database Files ...................................................................... 85
   List Data ............................................................................................................. 86
   How to Do It ..................................................................................................... 86
   Example: List Employee Data ....................................................................... 87
Use Logical Records ............................................................................................... 88
   How to Do It ..................................................................................................... 88
   Example: List Employee Data from a Logical Record .................................. 88
Select Data for a Report .......................................................................................... 89
   How to Do It ..................................................................................................... 89
   Example: Specify Data Selection Criteria for a Report ............................... 89
Debugging a CA Culprit Report ................................................................. 90
Reviewing CA Culprit Listings and Messages .............................................. 91
  Sequential Parameter Listing .................................................................. 91
  Input Parameter Listing ........................................................................ 91
  Runtime Messages .................................................................................. 91
    Total DML Syntax Error Codes .............................................................. 93
    Error Conditions Detected by the CA IDMS/DB .................................. 93
    Transparency Errors ........................................................................ 95
  The Report ............................................................................................ 97
Using CULLUS48 .................................................................................... 98
Printing Additional Report Information ...................................................... 99
  Example: Printing Additional Report Information .................................. 99

Totals Processing Techniques ................................................................. 100
Calculations on Accumulated Totals .......................................................... 100
  How to Do It ....................................................................................... 100
  Demonstration 14 ............................................................................. 101
Multiple-level Subtotals ......................................................................... 102
  How to Do It ....................................................................................... 102
  Demonstration 15 ............................................................................. 102
Creating a Sparse Listing ....................................................................... 105
  How to Do It ....................................................................................... 105
  Demonstration 16 ............................................................................. 105
Obtaining Work Field Values .................................................................. 106
  Demonstration 17 ............................................................................. 107

Using Subscripts .................................................................................... 107
Explicit Subscripts ................................................................................ 108
  How to Do It ....................................................................................... 108
  Demonstration (1) Using a Literal Subscript ......................................... 108
  Demonstration (2) Using a Counter as a Subscript .............................. 110
Zero Subscripts .................................................................................... 111
  How to Do It ....................................................................................... 111
  Demonstration 18 ............................................................................. 111
Fixed Repeating Fields ........................................................................ 113
  How to Do It ....................................................................................... 113
  Demonstration 19 ............................................................................. 114
Variable Repeating Groups .................................................................. 115
  How to Do It ....................................................................................... 115
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Fields</td>
<td>116</td>
</tr>
<tr>
<td>How to Do It</td>
<td>117</td>
</tr>
<tr>
<td>Demonstration 21</td>
<td>117</td>
</tr>
<tr>
<td>Obtaining Accumulated Totals</td>
<td>118</td>
</tr>
<tr>
<td>How to Do It</td>
<td>118</td>
</tr>
<tr>
<td>Demonstration 22</td>
<td>119</td>
</tr>
<tr>
<td>Obtaining Specific Field Values</td>
<td>120</td>
</tr>
<tr>
<td>How to Do It</td>
<td>120</td>
</tr>
<tr>
<td>Demonstration 23</td>
<td>120</td>
</tr>
<tr>
<td>Obtaining Sort-key Values</td>
<td>122</td>
</tr>
<tr>
<td>How to Do It</td>
<td>122</td>
</tr>
<tr>
<td>Demonstration 24</td>
<td>122</td>
</tr>
<tr>
<td>File Matching</td>
<td>123</td>
</tr>
<tr>
<td>How to Code Match-file Runs</td>
<td>123</td>
</tr>
<tr>
<td>How File Matching Works</td>
<td>124</td>
</tr>
<tr>
<td>Matching Single-Occurrence Files</td>
<td>128</td>
</tr>
<tr>
<td>Demonstration 25</td>
<td>128</td>
</tr>
<tr>
<td>Matching Multiple Transactions with a Single-Entry Master</td>
<td>129</td>
</tr>
<tr>
<td>Demonstration (1) Dropping Unmatched Records</td>
<td>129</td>
</tr>
<tr>
<td>Demonstration (2) Selecting Matching Account Numbers</td>
<td>131</td>
</tr>
<tr>
<td>Listing Accounts Without Transactions</td>
<td>133</td>
</tr>
<tr>
<td>Demonstration 26</td>
<td>133</td>
</tr>
<tr>
<td>Listing Transactions not on the Master File</td>
<td>135</td>
</tr>
<tr>
<td>Demonstration 27</td>
<td>135</td>
</tr>
<tr>
<td>Using the Match-file Facility for Table Initialization</td>
<td>137</td>
</tr>
<tr>
<td>How to Do It</td>
<td>137</td>
</tr>
<tr>
<td>What You Can Do 30</td>
<td>137</td>
</tr>
<tr>
<td>How to Do It</td>
<td>137</td>
</tr>
<tr>
<td>Demonstration 28</td>
<td>137</td>
</tr>
<tr>
<td>Using Files Defined to the Data Dictionary</td>
<td>140</td>
</tr>
<tr>
<td>What You Can Do 31</td>
<td>140</td>
</tr>
<tr>
<td>How to Do It</td>
<td>140</td>
</tr>
<tr>
<td>Demonstration 29</td>
<td>140</td>
</tr>
<tr>
<td>Creating Unique Match-key Names</td>
<td>140</td>
</tr>
<tr>
<td>How to Do It</td>
<td>141</td>
</tr>
<tr>
<td>What You Can Do 32</td>
<td>141</td>
</tr>
<tr>
<td>Demonstration 30</td>
<td>141</td>
</tr>
<tr>
<td>Qualified Fields</td>
<td>141</td>
</tr>
</tbody>
</table>
Demonstration (2) Converting to Packed Decimal Format .................................................. 166
 Demonstration (3) Writing Complete Records ................................................................. 167
 Demonstration (4) Writing Totals-only ............................................................................ 168
 Demonstration (5) Writing Variable-length Records ...................................................... 169
 Demonstration (6) Writing from the Database ............................................................... 169
 Creating Variable Headings ............................................................................................. 170
 How to Do It .................................................................................................................. 170
 Demonstration (1) Using the SORT Parameter ............................................................... 170
 Demonstration (2) Using the SORT/NOSORT Parameters ............................................ 172
 Accessing Non-database Files Defined in the Data Dictionary ....................................... 173
 How to Do It .................................................................................................................. 173
 What You Can Do 44 ...................................................................................................... 173

Additional CA IDMS/DB Facilities ............................................................. 174

How to Prepare for Record Retrieval ............................................................................. 174

Retrieving Partial Paths ................................................................................................. 174
 How to Do It .................................................................................................................. 174
 Example: Retrieving Partial Paths .............................................................................. 175

Retrieving Record Types by Name ............................................................................... 176
 How to Do It .................................................................................................................. 176
 Example: Retrieving Record Types by Name ............................................................... 176

Retrieving Record Types by Key .................................................................................. 178
 How to Do It .................................................................................................................. 178
 What You Can Do 47 ...................................................................................................... 179
 Demonstration (1) Accessing Records by Key ............................................................. 179
 Demonstration (2) Accessing Records with a Key File ............................................. 180

Retrieving Stand-alone Records ................................................................................... 181
 How to Do It .................................................................................................................. 181
 What You Can Do 48 ...................................................................................................... 182
 Example: Retrieving Stand-alone Records .................................................................. 182

Retrieving All Record Occurrences ............................................................................... 183
 How to Do It .................................................................................................................. 183
 What You Can Do 49 ...................................................................................................... 183
 Example: Retrieving All Record Occurrences ............................................................. 184

Testing for Record Occurrences .................................................................................... 185
 How to Do It .................................................................................................................. 185
 What You Can Do 50 ...................................................................................................... 185
 Example: Testing for Record Occurrences .................................................................. 185

Accessing Bill-of-Materials Structures ......................................................................... 186
 How to Do It .................................................................................................................. 186
What You Can Do 51 .......................................................................................................................... 187
Example: Accessing Bill-of-Materials Structures ................................................................................... 187
Accessing Multiple-Member Sets ............................................................................................................ 188
How to Do It ........................................................................................................................................ 188
What You Can Do 52 ............................................................................................................................ 189
Example: Accessing Multiple-Member Sets ............................................................................................. 189

Retrieving Data With SQL ....................................................................................................................... 190
Getting the Exact Column Names for an SQL Table ................................................................................ 190
Using the AS Clause to Rename SQL Columns ...................................................................................... 191
Retrieving Floating Point Data from SQL Columns ............................................................................... 191
Displaying an SQL Column with Data Type BINARY .......................................................................... 192
How to Test and Display Null Values .................................................................................................... 194
Handling Null Values at Total Time ....................................................................................................... 194
Understanding CA Culprit Decimal Point Handling ............................................................................ 196
Interpreting Common Error Messages in SQL Retrieval ........................................................................ 197

Creating New SQL Data Tables ............................................................................................................. 198
Create Table Syntax Generated by CA Culprit ..................................................................................... 198
Converting ASF Tables to SQL Tables ................................................................................................. 198
Assigning Null Values to an SQL Column ............................................................................................. 199
How CA Culprit Handles Data Insertion Errors ................................................................................... 200

Adding, Replacing, and Dropping Data on SQL Tables ......................................................................... 201
Updating SQL Columns of Type DATE ................................................................................................ 201
Getting the Exact Syntax for Updating a SQL Table ............................................................................. 202
Drop Table Example ............................................................................................................................... 202

The Personnel File Description ............................................................................................................. 203

Coding for Efficiency ............................................................................................................................. 204
Eliminating Unnecessary Records .......................................................................................................... 204
When to Use SELECT and When to Use Type 7 Logic ......................................................................... 205
Avoiding Unnecessary Logic ................................................................................................................... 205
Combining Tasks for Multiple Reports ................................................................. 205
Avoiding Unwanted Buffer Dumps ........................................................................ 206
Creating Subtotal Work Fields for Presorted Input Files ...................................... 206
CA Culprit JCL Considerations ............................................................................. 207
Miscellaneous Hints ............................................................................................. 208

Precoded CA Culprit User Modules ................................................................. 208
Procedure Modules ............................................................................................. 208
Output Modules .................................................................................................. 210
  Output Modules ................................................................................................. 210

Employee Database Subschema ................................................................. 210
Subschema Listing .............................................................................................. 210
Data Structure Diagram ...................................................................................... 217

Summary of CA Culprit Parameters ................................................................. 217
Processing Operations ......................................................................................... 218
Summary of Advanced Capabilities for Standard Files ........................................... 219
Summary of Parameters for Using Database Data ................................................ 221
Summary of Advanced Capabilities for Accessing the Database ............................ 222
Minimum Coding Requirements for Using Data Tables ....................................... 222
Minimum Coding Requirements for Using SQL Tables ....................................... 224
CA Culprit is a batch utility used to generate files, print reports, and create data tables from conventional and database files. You can print these reports or write them to conventional files. Tables are stored in the database as part of CA IDMS/DB and CA-ICMS.

CA Culprit provides a means for writing detailed or summary reports with a minimum of coding requirements. By using CA Culprit parameters, you can:

- Retrieve data from conventional file and database structures.
- Read and process up to 32 conventional files in one CA Culprit run.
- Produce automatically formatted or customized printed reports.
- Produce detailed or summary reports.
- Produce up to 100 reports in a single CA Culprit run.
- Produce reports on standard or special forms.
- Write reports to conventional files on cards, tape, or disk.
- Produce nonprint output in any format. (for example, packed decimal)
- Read, create, update, and store data tables in a CA-ICMS environment.
- Perform logical operations (computations, comparisons, sorting, file matching, branching, calls to external modules).
- Control processing order.

For more information, see the following topics:
What You Should Know Before You Begin

You do not need to be an expert to write CA Culprit reports. You do not need to be a computer programmer. A few basic data processing concepts are enough to get you started. From there, you can learn as you go.

What a File Description Tells You

When you obtain a description of the data file you are going to use for your report, you will notice that the file is described in these terms:

- **Record size**
  Indicates the length of a single record measured in characters or bytes.

- **Block size**
  Indicates the number of records stored next to each other before there is a spacing interval. The block size is also measured in bytes.

- **Record format**
  Indicates whether the records in the file equal (fixed) in length, vary in length, or have an undefined length.
• **File type**
  Indicates how the records are organized in the file. The possibilities are a sequential file, an ISAM file, a card file, a VSAM file, or a database.

The file description also gives size and location of each data field:

• **Start position**
  Indicates the beginning location of the field.

• **Length**
  Indicates the amount of space allocated for the particular field. The field length is expressed in bytes.

• **Data type**
  Identifies the data as: alphanumeric (a combination of alphabetic characters and numbers) or numeric (zoned decimal, packed decimal, unsigned packed decimal, binary, or bit).

---

### How Information Is Organized in a Computer

Information is organized in a computer in much the same way written information is organized in a file cabinet:

• **Collections of related information, called files, are analogous to file folders.** For example, most companies have personnel files.

• **Each file has subdivisions called records, which are analogous to the papers in a folder.** For example, the personnel file contains employee records.

• **Each record contains basic units of data called fields, which are analogous to related fragments of information contained on the papers in a folder.** For example, an employee record contains a name field.

The figure below illustrates the relationships between files, records, and fields:
How to Use Information in a File Description

You don’t have to understand the differences between the data types to use CA Culprit, but you must identify them correctly. You just have to read the information off of the file description and enter it correctly into your code.

The figure below shows the layout of a record within an employee data file. Sample data illustrates how an occurrence of the employee record might appear:

The sample record above has six data fields that can be used for a report. Note that the record is always 200 bytes long, even if all the space is not needed by a particular word or number. The data field lengths also remain constant, with blanks used to fill out the space.

For example, the record allocates a 10-byte space for FIRST-NAME; therefore, the five characters in ROGER are followed by five blanks so that WILCO begins in position 15, the starting position of LAST-NAME.
Building and Using Data Tables

CA Culprit is used in a CA IDMS/DB environment to create, retrieve, and manipulate data tables. CA Culprit:

- Creates tables from:
  - Conventional files
  - The database
  - Other tables
- Retrieves tables
- Updates and regenerates tables
- Consolidates tables
- Deletes tables

For more information, see the following topics:
- Creating Tables (see page 17)
- Retrieving Data Tables (see page 24)
- Modifying Data Tables (see page 25)
- Consolidating Tables (see page 31)

Creating Tables

You can:

- Create a table from data stored
  - On conventional files (sequential, ISAM, or VSAM)
  - On a database (CA IDMS/DB, IMS, RDMS, or TOTAL)
  - In an already existing table
- Store rows in ascending or descending order
- Find errors in your code by using the data table reports that are automatically generated in the Input Parameter and Run-Time Message Listings of the CA Culprit run
How to Do It

To create a table:

1. **Define the incoming data** using input definition parameters (shown in the table below).

2. **Define the table** using the OUTPUT parameter and the keywords listed in the table below.

3. **Define the columns of the table** using type 5 or type 6 parameters.

4. **Specify special options**, as needed, using the options listed in the table below.

You can find complete descriptions of the parameters and keywords in the Accessing Tables (https://docops.ca.com/display/IDMSCU/INPUT+-+Accessing+Tables).

### Required Parameters and Keywords for Creating a Table

<table>
<thead>
<tr>
<th>If the data source is...</th>
<th>the required parameters and keywords are...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A conventional file (sequential, ISAM, and VSAM)</td>
<td>An <strong>INPUT</strong> parameter, including the file type (PS, IS, CARD, or VS)</td>
</tr>
<tr>
<td></td>
<td>REC parameters (at least one)</td>
</tr>
<tr>
<td></td>
<td>An <strong>OUTPUT</strong> parameter with:</td>
</tr>
<tr>
<td></td>
<td><strong>TABLE</strong>=</td>
</tr>
<tr>
<td></td>
<td><strong>TYPE</strong>=CREATE</td>
</tr>
<tr>
<td></td>
<td><strong>USER</strong>=</td>
</tr>
<tr>
<td></td>
<td><strong>PW</strong>=</td>
</tr>
<tr>
<td></td>
<td><strong>CATALOG</strong>=</td>
</tr>
<tr>
<td></td>
<td>Type 5 parameters (at least one)</td>
</tr>
<tr>
<td></td>
<td>Type 6 parameters to create a totals-only table</td>
</tr>
<tr>
<td>A CA IDMS/DB database</td>
<td><strong>DATABASE PARAMETER</strong> with <strong>DICTNAME</strong>=</td>
</tr>
<tr>
<td></td>
<td>An <strong>INPUT DB SS</strong>= parameter</td>
</tr>
<tr>
<td></td>
<td><strong>PATH</strong> parameter</td>
</tr>
<tr>
<td></td>
<td>An <strong>OUTPUT</strong> parameter with:</td>
</tr>
<tr>
<td></td>
<td><strong>TABLE</strong>=</td>
</tr>
<tr>
<td></td>
<td><strong>TYPE</strong>=CREATE</td>
</tr>
<tr>
<td>If the data source is...</td>
<td>the required parameters and keywords are...</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>USER=</td>
</tr>
<tr>
<td></td>
<td>PW=</td>
</tr>
<tr>
<td></td>
<td>CATALOG=</td>
</tr>
<tr>
<td></td>
<td>Type 5 parameters (at least one)</td>
</tr>
<tr>
<td></td>
<td>Type 6 parameters to create a totals-only table</td>
</tr>
<tr>
<td>An existing table</td>
<td>An INPUT parameter with:</td>
</tr>
<tr>
<td></td>
<td>TABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=COPY</td>
</tr>
<tr>
<td></td>
<td>USER=</td>
</tr>
<tr>
<td></td>
<td>PW=</td>
</tr>
<tr>
<td></td>
<td>CATALOG=</td>
</tr>
<tr>
<td></td>
<td>An OUTPUT parameter with:</td>
</tr>
<tr>
<td></td>
<td>TABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=CREATE</td>
</tr>
<tr>
<td></td>
<td>USER=</td>
</tr>
<tr>
<td></td>
<td>PW=</td>
</tr>
<tr>
<td></td>
<td>CATALOG=</td>
</tr>
<tr>
<td></td>
<td>Type 5 parameters (at least one)</td>
</tr>
<tr>
<td></td>
<td>Type 6 parameters to generate a totals-only table</td>
</tr>
</tbody>
</table>

**Options Available when Creating a Table**

<table>
<thead>
<tr>
<th>The option ...</th>
<th>used on this parameter</th>
<th>specifies ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX=</td>
<td>type 5 or type 6</td>
<td>table row storage indexed in ascending (A) or descending (D) order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0151*001 CUST-NUM IX=A</td>
</tr>
<tr>
<td>COLUMN=</td>
<td>type 5 or type 6</td>
<td>absolute field placement for tables extending beyond column 9999</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0151 NAME COL=13001</td>
</tr>
</tbody>
</table>
The option ... used on this parameter specifies ...

<table>
<thead>
<tr>
<th>Option</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/D</td>
<td>OUTPUT</td>
<td>a totals-only (T) or a details-only (D) report</td>
</tr>
<tr>
<td>COMMENT=' '</td>
<td>OUTPUT</td>
<td>ASF comment field update</td>
</tr>
<tr>
<td>AREA=</td>
<td>OUTPUT</td>
<td>ASF area name field update</td>
</tr>
<tr>
<td>ONLINE=YES/NO</td>
<td>OUTPUT</td>
<td>Whether to create dialogs and maps for table viewing</td>
</tr>
<tr>
<td>DISPLAY=YES/NO</td>
<td>OUTPUT</td>
<td>The ability to retrieve data from the table</td>
</tr>
<tr>
<td>LOAD=YES/NO</td>
<td>OUTPUT</td>
<td>The ability to store data on the table</td>
</tr>
<tr>
<td>CHANGE=YES/NO</td>
<td>OUTPUT</td>
<td>The ability to update data from the table</td>
</tr>
<tr>
<td>ERASE=YES/NO</td>
<td>OUTPUT</td>
<td>The ability to erase rows of data</td>
</tr>
</tbody>
</table>

For more information about ASF, see the Using CA IDMS ASF (https://docops.ca.com/pages/viewpage.action?pagId=328580829).

Demonstration (1) Creating a Table from a Sequential File

Objective

This example creates a table from a sequential file containing customer information. The rows in the table are placed in ascending order based on the customer number (CUST-NUM). Generated data table reports that appear in the Input Parameter and Run Time Message Listing are shown.

Parameters

- **IN 80 F 4000 PS(TAPE)**
  - **IN 80 F 4000 PS(TAPE)** defines a sequential input file with fixed length 80-byte records, a block size of 4000 bytes, which is stored on tape.

  `REC CUST-NUM 1 SREC CUST-NAME 19 20`

  `REC` parameters define the fields of the sequential file.

<table>
<thead>
<tr>
<th>OUTPUT</th>
<th>TABLE=CUSTOMER-FILE TYPE=CREATE USER=DOC1 PW=DOC1 *CATALOG=ASFDICT ONLINE=YE S</th>
</tr>
</thead>
</table>

  `OUTPUT` identifies the OUTPUT parameter, which specifies options to define the output table.

- **TABLE=CUSTOMER-FILE** names the table. **TABLE=** must be the first keyword on the OUTPUT parameter.

  `TYPE=CREATE` defines a new table. **TYPE=** must be the second keyword on the OUTPUT parameter.

  `USER=DOC1 PW=DOC1` identifies the user and password for signing on to the catalog.

  `CATALOG=ASFDICT` identifies the catalog (dictionary) containing the table definition.
ONLINE=YES creates dialogs and maps to view the table online through ASF.

0151*001 CUST-NUM IX=A
0151*002 CUST-NAME

*001 and *002, specify the sequence of column placement.

CUST-NUM and CUST-NAME are the column names. The fields named on the type 5 parameters determine the data type and column size.

IX=A indexes the rows in ascending order on customer number.

Complete Code

col. 2

REC CUST-NAME 19 20
010OUTPUT TABLE=CUSTOMER-FILE TYPE=CREATE USER=DOC1 PW=DOC1*CATALOG=ASFDICT ONLINE=YES
0151*001 CUST-NUM IX=A 0151*002 CUST-NAME

Results

Demonstration (2) Creating a Table from the Database

Objective

This example creates a table from the employee database. Column headings different from the database field names are created by moving field names to work fields. The work field names are specified on the type 5 lines.

Parameters
Demonstration (3) Creating a Table from an Existing Table

Objective

This example selects all part-time workers from a table (ALL-EMPLOYEES) and creates a new table (PART-TIME) for part-time employees.

The required keywords for the INPUT parameter are similar to those shown earlier in this section for the OUTPUT parameter.

Parameters

INPUT TABLE=ALL-EMPLOYEES TYPE=COPY USER=DOC1 PW=DOC1 *CATALOG=ASFDICT
**INPUT** identifies the incoming data.

**TABLE=ALL-EMPLOYEES** names the table. **TABLE=** must be the first keyword on the **INPUT** parameter.

**TYPE=COPY** copies the EMPLOYEE table. **TYPE=** must be the second keyword on the **INPUT** parameter.

**USER=DOC1 PW=DOC1** identifies the user and password for signing on to the catalog (dictionary).

**CATALOG=ASFDICT** identifies the catalog containing the table definition.

**SELECT EMPLOYEES WHEN STATUS EQ 'PT'**

**SELECT** retrieves only part-time employees.

**02OUTPUT TABLE=PART-TIME TYPE=CREATE USER=DOC1 PW=DOC1 *CATALOG=ASFDICT ONLINE=YES**

**OUTPUT** identifies the **OUTPUT** parameter, which includes the **CREATE** instruction.

**TABLE=PART-TIME** names the new table. **TABLE=** must be the first keyword on the **OUTPUT** parameter.

**TYPE=CREATE** specifies a new table.

**USER=DOC1 PW=DOC1** identifies the authorized user and password for signing on to the catalog.

**CATALOG=ASFDICT** identifies the catalog (dictionary) containing the table definition.

**ONLINE=YES** creates dialogs and maps to view the table online through ASF.

- **02SORT NAME**
  - **SORT** alphabetizes the names.

**Complete Code**

```
DATABASE DICTNAME=TSTDICT
INPUT TABLE=ALL-EMPLOYEES TYPE=COPY USER=DOC1 PW=DOC1 *CATALOG=ASFDICT SELECT EMPLOYEE
S WHEN STATUS EQ 'PT'
02OUTPUT TABLE=PART-TIME TYPE=CREATE USER=DOC1 PW=DOC1 *CATALOG=ASFDICT ONLINE=YES 02S
ORT NAME 0251*001 ID 0251*002 NAME
```

**Demonstration (4) Creating a Totals-only Table**

**Objective**

In this example, the output table contains a summary of monthly receipts for each branch of a bank. Type 6 parameters define the name, size, and position of the table columns. One type 5 parameter accumulates daily receipts.

```
01OUT T...
* ONLINE=YES
* COMMENT='TOTAL MONTHLY RECEIPTS FOR EACH BRANCH'
```
T specifies a totals-only report.

**ONLINE=YES** directs ASF to create dialogs and maps for this data table.

**COMMENT=’... ’** updates the comment field on the ASF Table Definition screen.

```plaintext
0161*001 BRANCH FZ 0161*002 MONTH FZ 0161*003 YEAR FZ 0161*004 MONTHLY-
RECEIPTS SZ=5 FP DP=2
```

FZ and FP specify zoned and packed numeric fields respectively.

**Complete Code**

```plaintext
IN 80 F 80
REC DAILY-RECEIPTS 8 10 2 DP=2
REC DAY 18 2 2
REC MONTH 20 2 2
REC YEAR 22 2 2
REC BRANCH 24 2 2
01OUT T TABLE=RECEIPT-TOTALS TYPE=CREATE USER=DOC1 PW=DOC1
0161*...
```

Retrieving Data Tables

You can retrieve a copy of your data table and print a listing or a report based on the table contents. The table can also be viewed or modified with ASF.

**How to Do It**

A data table is retrieved by coding the following statements:

1. An **INPUT** parameter that specifies at least the following information:

    a. The name of the table being copied (**TABLE=**)  
    b. The copy function (**TYPE=COPY**)  
    c. A valid user ID (**USER=**)  
       If you are not the table owner, you must have authorization to access the table.  
    d. A valid password (**PW=**)  
    e. The catalog (dictionary) in which the table definition resides (**CATALOG=**)
Demonstration 12

Objective

This report retrieves and prints a listing of the part-time employees from the PART-TIME table, which was created earlier in this section.

Parameters

INPUT TABLE=PART-TIME
TYPE=COPY USER=DOC1 PW=DOC1
*CATALOG=ASFDICT

INPUT identifies the INPUT parameter, which includes the COPY instruction.

TABLE=PART-TIME specifies the table used for input. TABLE= must be the first keyword on the INPUT parameter.

TYPE=COPY specifies copying the PART-TIME table. TYPE= must be the second keyword on the INPUT parameter.

USER=DOC1 PW=DOC1 identifies the user and password for signing on to the catalog.

CATALOG=ASFDICT identifies the catalog that contains the table definition.

• 01OUTPUT 60
  OUTPUT 60 specifies printed output with 60 character lines.

Complete Code

col. 2

INPUT TABLE=PART-TIME TYPE=COPY USER=DOC1 PW=DOC1*CATALOG=ASFDICT 01OUTPUT 60 0151*010 ID HH 'EMPLOYEE ID'
0151*020 NAME HH 'EMPLOYEE NAME'

Result

<table>
<thead>
<tr>
<th>EMPLOYEE ID</th>
<th>EMPLOYEE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>476</td>
<td>BETSY ZEDI</td>
</tr>
<tr>
<td>51</td>
<td>CYNTHIA JOHNSON</td>
</tr>
<tr>
<td>49</td>
<td>DOUGLAS KAHALLY</td>
</tr>
<tr>
<td>457</td>
<td>HARRY ARM</td>
</tr>
</tbody>
</table>

Modifying Data Tables

You can:

• Add rows of data to an existing table
• Replace rows of data in an existing table
How to Do It

When modifying an existing table, specify:

1. **The source of the input data** on the INPUT parameter

2. **The modified table name** and the function to be performed (TYPE=ADD/REPLACE/DELETE/GENERATE) on the OUTPUT parameter

3. **The columns of the table** with type 5 or type 6 parameters

*Note: At least one type 5 parameter is required. A dummy type 5 parameter can be used when necessary.*

Demonstration (1) Adding Rows to an Existing Table

**Objective**

This example adds the contents of the NEW-CUSTOMER table to the CUSTOMER-LIST table. Note that the type 5 parameters must match the column definitions for the CUSTOMER-LIST table.

**Parameters**

```
INPUT TABLE=NEW-CUSTOMERS TYPE=COPY USER=DOC1 PW=DOC1 *
CATALOG=ASFDICT

OUTPUT TABLE=CUSTOMER-LIST TYPE=ADD USER=DOC1 PW=DOC1 *
CATALOG=ASFDICT
```

**INPUT** identifies the INPUT parameter, which includes the COPY instruction.

**TABLE=NEW-CUSTOMERS** specifies the table that will be added to CUSTOMER-LIST.

**TYPE=COPY** specifies copying rows from the input table NEW-CUSTOMERS.

**OUTPUT** identifies the OUTPUT parameter, which includes the ADD instruction.

**TABLE=CUSTOMER-LIST** specifies the table receiving the additional rows.

**TYPE=ADD** adds new rows to the CUSTOMER-LIST table.

```
051*001 CUST-NUMBER
051*002 CUST-NAME
051*003 CUST-ADDRESS
```
1, 2, and 3 define three table columns.

Complete Code

```
col. 2

INPUT TABLE=NEW-CUSTOMERS TYPE=COPY USER=DOC1 PW=DOC1
* CATALOG=ASFDICT
05OUTPUT TABLE=CUSTOMER-LIST TYPE=ADD USER=DOC1 PW=DOC1
* CATALOG=ASFDICT
0551*001 CUST-NUMBER FZ SZ=5
0551*002 CUST-NAME
0551*003 CUST-ADDRESS
```

Results

The contents of NEW-CUSTOMER:

<table>
<thead>
<tr>
<th>CUST-NUMBER</th>
<th>CUST-NAME</th>
<th>CUST-ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>25060</td>
<td>SUSAN</td>
<td>ARMITAGE 56725 OAK STREET PITTSFIELD MA02956</td>
</tr>
<tr>
<td>27056</td>
<td>ALLEN</td>
<td>KOHN 22651 POLK STREET SAN FRANCISCO CA09889</td>
</tr>
<tr>
<td>39557</td>
<td>HENRY</td>
<td>TRUMBLE 2102 WASHINGTON ST BROOKLINE MA02147</td>
</tr>
<tr>
<td>38415</td>
<td>SMITH</td>
<td>SMYTH 5999 SANDY LANE LONG BEACH CA09743</td>
</tr>
<tr>
<td>33480</td>
<td>VICKY</td>
<td>KNIGHT 5678 PINE ROAD PORTLAND ME06895</td>
</tr>
<tr>
<td>69879</td>
<td>BRUNO</td>
<td>THORPE 22002 PEACHS AVE FRESNO CA96543</td>
</tr>
<tr>
<td>99983</td>
<td>ELLEN</td>
<td>SANDS 1 APPLE ORCHARD BISCOE NC64321</td>
</tr>
</tbody>
</table>

The contents of CUSTOMER-LIST before the ADD modification:

<table>
<thead>
<tr>
<th>CUST-NUMBER</th>
<th>CUST-NAME</th>
<th>CUST-ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>15060</td>
<td>SHARON</td>
<td>ARMSTRONG 25 CEDAR STREET WAYLAND MA02756</td>
</tr>
<tr>
<td>21056</td>
<td>AMOS</td>
<td>JOHNSON 22651 MASS AVENUE SAN FRANCISCO CA09881</td>
</tr>
<tr>
<td>29557</td>
<td>IRWIN</td>
<td>TRIMBLE 102 COLBOURNE ST BROOKLINE MA02147</td>
</tr>
<tr>
<td>38115</td>
<td>IRMA</td>
<td>DOONES 5656 SANDHILL ROAD OCEAN CITY CA09743</td>
</tr>
<tr>
<td>33470</td>
<td>VICTORIA</td>
<td>DAY 1234 OAK HILL ROAD EVERGREEN MI07895</td>
</tr>
<tr>
<td>69876</td>
<td>BRUCE</td>
<td>THORPE 11002 PEACHTREE LA ATLANTA GA76543</td>
</tr>
<tr>
<td>99083</td>
<td>HELEN</td>
<td>SANTOVEC 3 APPLE ORCHARD CHARLOTTE NC64321</td>
</tr>
</tbody>
</table>

The contents of CUSTOMER-LIST after the ADD modification:

<table>
<thead>
<tr>
<th>CUST-NUMBER</th>
<th>CUST-NAME</th>
<th>CUST-ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>15060</td>
<td>SHARON</td>
<td>ARMSTRONG 25 CEDAR STREET WAYLAND MA02756</td>
</tr>
<tr>
<td>21056</td>
<td>AMOS</td>
<td>JOHNSON 22651 MASS AVENUE SAN FRANCISCO CA09881</td>
</tr>
<tr>
<td>29557</td>
<td>IRWIN</td>
<td>TRIMBLE 102 COLBOURNE ST BROOKLINE MA02147</td>
</tr>
<tr>
<td>38115</td>
<td>IRMA</td>
<td>DOONES 5656 SANDHILL ROAD OCEAN CITY CA09743</td>
</tr>
<tr>
<td>33470</td>
<td>VICTORIA</td>
<td>DAY 1234 OAK HILL ROAD EVERGREEN MI07895</td>
</tr>
<tr>
<td>69876</td>
<td>BRUCE</td>
<td>THORPE 11002 PEACHTREE LA ATLANTA GA76543</td>
</tr>
<tr>
<td>99083</td>
<td>HELEN</td>
<td>SANTOVEC 3 APPLE ORCHARD CHARLOTTE NC64321</td>
</tr>
</tbody>
</table>

Demonstration (2) Replacing Rows of an Existing Table

Objective

This example updates customer numbers by replacing the rows in the NEW-CUSTOMERS table with data read in from a sequential file. Note that the type 5 parameters must match the column definitions for the CUSTOMER-LIST table.

Parameters
**IN 80**

*IN 80* defines an 80-byte sequential file containing the new customer numbers.

```
05 OUTPUT TABLE=NEW-CUSTOMERS
   TYPE=REPLACE USER=DOC1 PW=DOC1
   * CATALOG=ASFDICT
```

**OUTPUT** identifies the OUTPUT parameter, which includes the REPLACE instruction.

**TABLE=NEW-CUSTOMERS** names the table in which data is replaced.

**TYPE=REPLACE** replaces existing table rows with new data.

0551*00
  CUST-NUMBER 1
0551*00
  CUST-NAME 2
0551*00
  CUST-ADDRESS 3

1, 2, and 3 define three table columns.

**Complete Code**

col. 2

```
IN 80 REC CUST-NUMBER 1 5 2
   REC CUST-NAME 6 20
   REC CUST-ADDRESS 26 42
05 OUTPUT TABLE=NEW-CUSTOMERS TYPE=REPLACE USER=DOC1 PW=DOC1
   * CATALOG=ASFDICT
0551*00
  CUST-NUMBER
0551*00
  CUST-NAME
0551*00
  CUST-ADDRESS
```

**Results**

*Previous contents of NEW-CUSTOMER:*

<table>
<thead>
<tr>
<th>CUST-NUMBER</th>
<th>CUST-NAME</th>
<th>ADDRESS</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>250606</td>
<td>SUSAN</td>
<td>ARMITAGE  56725 OAK STREET</td>
<td>PITTSFIELD</td>
<td>MA</td>
<td>02956</td>
</tr>
<tr>
<td>270566</td>
<td>ALLEN</td>
<td>KOHN  22651 POLK STREET</td>
<td>SAN FRANCISCO</td>
<td>CA</td>
<td>94105</td>
</tr>
<tr>
<td>395566</td>
<td>HENRY</td>
<td>TRUMBLE  2392 WASHINGTON ST</td>
<td>BROOKLINE</td>
<td>MA</td>
<td>02147</td>
</tr>
<tr>
<td>304666</td>
<td>MARY</td>
<td>SMYTH  5999 SANDY LANE</td>
<td>LONG BEACH</td>
<td>CA</td>
<td>90210</td>
</tr>
<tr>
<td>304666</td>
<td>VICKY</td>
<td>KNIGHT  5678 PINE ROAD</td>
<td>PORTLAND</td>
<td>ME</td>
<td>04020</td>
</tr>
<tr>
<td>699866</td>
<td>BRUNO</td>
<td>THOR  22002 PEACHES AVE</td>
<td>FRESNO</td>
<td>CA</td>
<td>93701</td>
</tr>
<tr>
<td>999866</td>
<td>ELLEN</td>
<td>SANDS  1 APPLE ORCHARD</td>
<td>BISCOE</td>
<td>NC</td>
<td>64321</td>
</tr>
</tbody>
</table>

*The contents of NEW-CUSTOMERS after REPLACE:*

<table>
<thead>
<tr>
<th>CUST-NUMBER</th>
<th>CUST-NAME</th>
<th>ADDRESS</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>SUSAN</td>
<td>ARMITAGE  56725 OAK STREET</td>
<td>PITTSFIELD</td>
<td>MA</td>
<td>02956</td>
</tr>
<tr>
<td>272</td>
<td>ALLEN</td>
<td>KOHN  22651 POLK STREET</td>
<td>SAN FRANCISCO</td>
<td>CA</td>
<td>94105</td>
</tr>
<tr>
<td>393</td>
<td>HENRY</td>
<td>TRUMBLE  2392 WASHINGTON ST</td>
<td>BROOKLINE</td>
<td>MA</td>
<td>02147</td>
</tr>
<tr>
<td>384</td>
<td>MARY</td>
<td>SMYTH  5999 SANDY LANE</td>
<td>LONG BEACH</td>
<td>CA</td>
<td>90210</td>
</tr>
<tr>
<td>335</td>
<td>VICKY</td>
<td>KNIGHT  5678 PINE ROAD</td>
<td>PORTLAND</td>
<td>ME</td>
<td>04020</td>
</tr>
<tr>
<td>696</td>
<td>BRUNO</td>
<td>THOR  22002 PEACHES AVE</td>
<td>FRESNO</td>
<td>CA</td>
<td>93701</td>
</tr>
<tr>
<td>997</td>
<td>ELLEN</td>
<td>SANDS  1 APPLE ORCHARD</td>
<td>BISCOE</td>
<td>NC</td>
<td>64321</td>
</tr>
</tbody>
</table>

**Demonstration (3) Deleting a Table**

**Objective**
This example deletes the definition and data of a table containing old customer information.

- **IN 80**
  
  **IN 80** is used to meet the CA Culprit INPUT parameter requirement.

```plaintext
05 OUTPUT TABLE=OLD-CUSTOMERS
  TYPE=DELETE USER=DOC1 PW=DOC1
  CATALOG=ASFDICT
```

**OUTPUT** identifies the OUTPUT parameter, which includes the DELETE instruction.

**TABLE=OLD-CUSTOMERS** names the table to be deleted.

**TYPE=DELETE** specifies deleting the table.

```plaintext
05 7001 DROP
```

`DROP` on the type 7 parameter stops unnecessary input processing.

**Complete Code**

```plaintext
col. 2

IN 80
REC INFIELD 1 80
05 OUTPUT TABLE=OLD-CUSTOMERS TYPE=DELETE USER=DOC1 PW=DOC1
  CATALOG=ASFDICT
05 7001 DROP
```

**Results**

<table>
<thead>
<tr>
<th>mm/dd/yy</th>
<th>RUN TIME</th>
<th>MESSAGE</th>
<th>volser</th>
<th>Vnn.n</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**********</td>
<td>END OF FILE</td>
<td>********************</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7 INPUT RECORDS READ
DATA TABLE UPDATE STATISTICS

<table>
<thead>
<tr>
<th>REPORT</th>
<th>FUNCTION</th>
<th>LR-NAME</th>
<th>DEFINITION</th>
<th>STORED</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>DEL</td>
<td>OLD-CUSTOMERS</td>
<td>DELETED</td>
<td>0</td>
</tr>
</tbody>
</table>

**Demonstration (4) Regenerating a Table**

**Objective**

This example modifies the existing NEW-CUSTOMERS table so that it can be viewed online through ASF. The ONLINE=YES and the TYPE=GENERATE instructions cause ASF to construct a dialog and regenerate the table.

**Parameters**

- **INPUT 80**
  
  **INPUT** is used to meet CA Culprit's INPUT parameter requirement.
OUTPUT identifies the OUTPUT parameter, which includes the GENERATE instruction.

TABLE=NEW-CUSTOMERS identifies the table to be generated.

TYPE=GENERATE regenerates the table.

ONLINE=YES creates a dialog to allow viewing of NEW-CUSTOMERS online with ASF.

COMMENT='REGENERATED WITH CA Culprit' specifies the ASF comment line text for the table.

5 specifies a type 5 parameter to meet the CA Culprit coding requirements.

- 057001 DROP
  DROP on the type 7 parameter stops unnecessary input processing.

Complete Code

```plaintext
05 OUTPUT TABLE=NEW-CUSTOMERS TYPE=GENERATE
USER=DOC1 PW=DOC1
* CATALOG=ASFDICT
ONLINE=YES
COMMENT='REGENERATED WITH CA Culprit'
```

Results: Viewing the Table Through ASF

Before regenerating the table:

The ASF ASEL Screen

Table Name:... User Name: DOC1

After regenerating the table:

The ASF TDEF Screen
Consolidating Tables

You can process data from more than one input table in a single CA Culprit run by reading the data tables as one logical file with the TYPE=CONSOLIDATION option. In addition, rows can be selected with WHERE or SELECT/BYPASS criteria.

Table data can be consolidated from:

- The same dictionary
- Different dictionaries
- Different DDS nodes
- Different central versions

How to Do It

To consolidate tables:

1. Check the column definitions of each table to be used. The definitions must be identical.
2. **Define each table on an INPUT parameter** using the keywords listed in the table below.

   - **TYPE=COPY** must appear on the first INPUT parameter to identify the primary data table. The primary data table is the source for:
     - The generated PATH parameter and all necessary REC parameters for the run
     - The specification of selection criteria (SELECT/BYPASS or the WHERE clause)
   
   - **TYPE=CONSOL** must appear on subsequent INPUT parameters to identify the secondary data tables. The order of appearance of the INPUT parameters determines the processing order of the tables.

3. **Specify the output** by using:

   - An OUTPUT parameter, if needed
   - At least one type 5 parameter
   - Type 6, type 7, and type 8 parameters as needed.

4. **Reference tables**, if needed, in a table consolidation run by using the reserved words shown in the table below. These reserved words are valid on SELECT/BYPASS, edit, SORT, and process parameters.

**Required Keywords:**

- **TABLE=**
  The table name

- **TYPE=COPY/CONSOL**
  The primary and secondary tables to be consolidated

- **USER=**
  The individual having authority to sign on to the catalog (dictionary)

- **PW=**
  The user's password

- **OWNER=**
  The owner of the table

- **CATALOG=**
  The catalog or dictionary containing the table definition

**Optional Keywords:**

- **LOCATION=**
  Overrides DATABASE DICTNODE=

- **SYSCTL= (z/OS and OS/390 Users)**
  Names the SYSCTL file that controls the system accessing the table.
• **VALIDATE**=FIRST/ALL
  Compares column definitions of secondary tables to definitions of the primary table.

• **WHERE**
  Applies WHERE clause criteria to all tables being consolidated.

**Reserved Words to Reference Tables**

<table>
<thead>
<tr>
<th>Reserved word</th>
<th>What it is</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE-ID</td>
<td>A 3-byte zoned decimal field</td>
<td>Indicates the data table currently being accessed</td>
</tr>
<tr>
<td>TABLE-NAME</td>
<td>A 56-byte alphanumeric field</td>
<td>Contains the name of the table currently being accessed</td>
</tr>
<tr>
<td>SUBSCHEMA-NAME</td>
<td>An 8-byte alphanumeric field</td>
<td>Contains the subschema name of the table being accessed</td>
</tr>
</tbody>
</table>

**Demonstration 13**

**Objective**

This example reads employee data from three tables that reside in different central versions and different dictionaries to produce a selected listing of employees in the data processing departments of three offices.

The SYSCTL= keyword values refer to ddnames that appear in an z/OS job control language stream:

```plaintext
//SYSTEM84 DD DSN=DBDC.SYSTEM84.SYSCTL,DISP=SHR
//SYSTEM85 DD DSN=DBDC.SYSTEM85.SYSCTL,DISP=SHR
//SYSTEM86 DD DSN=DBDC.SYSTEM86.SYSCTL,DISP=SHR
```

**Parameters**

**INPUT TABLE=BOSTON-EMPLOYEES TYPE=COPY**

`USER=DOC1 PW=DOC1
OWNER=DRH
CATALOG=ASFDICT
SYSCTL=SYSTEM84` * WHERE DEPT-ID EQ '1234'*

**TABLE=BOSTON-EMPLOYEES** is the first required keyword on the INPUT parameter and specifies BOSTON-EMPLOYEES as the first table to be retrieved.

**TYPE=COPY** copies the BOSTON-EMPLOYEES table.

**CATALOG=ASFDICT** identifies the catalog that contains the table definition.

**SYSCTL=SYSTEM84** specifies the central version z/OS that contains ASFDICT. (VM/ESA users should use CVMACH= option.)

**WHERE DEPT-ID EQ '1234'** specifies selection criteria applicable to all tables read.
INPUT TABLE=CHICAGO-EMPLOYEES TYPE=CONSOL
USER=DOC1 PW=DOC1 *
* OWNER=DDR
CATALOG=TSTDICT
SYSCTL=SYSTEM85

**TABLE=CHICAGO-EMPLOYEES** is the first required keyword on the INPUT parameter and specifies CHICAGO-EMPLOYEES as a secondary table.

**TYPE=CONSOL** specifies consolidation of CHICAGO-EMPLOYEES with BOSTON-EMPLOYEES.

**CATALOG=TSTDICT** identifies the catalog that contains the table definition.

**SYSCTL=SYSTEM85** specifies the central version that contains TSTDICT.

INPUT TABLE=DENVER-EMPLOYEES TYPE=CONSOL
USER=DOC1 PW=DOC1 *
* OWNER=ADR
CATALOG=PRODICT
SYSCTL=SYSTEM86

**TABLE=DENVER-EMPLOYEES** is the first required keyword on the INPUT parameter and specifies DENVER-EMPLOYEES as a secondary table.

**TYPE=CONSOL** specifies consolidation of DENVER-EMPLOYEES with CHICAGO-EMPLOYEES.

**CATALOG=PRODICT** identifies the catalog that contains the table definition.

**SYSCTL=SYSTEM86** specifies the central version that contains PRODICT.

**Complete Code**

```
15 janvier 1985

INPUT TABLE=CHICAGO-EMPLOYEES TYPE=CONSOL
USER=DOC1 PW=DOC1 *
* OWNER=DDR
CATALOG=TSTDICT
SYSCTL=SYSTEM85

INPUT TABLE=DENVER-EMPLOYEES TYPE=CONSOL
USER=DOC1 PW=DOC1 *
* OWNER=ADR
CATALOG=PRODICT
SYSCTL=SYSTEM86

output d

result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEES IN DATA PROCESSING</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
<th>EMPLOYEE</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOSTON-EMPLOYEES</td>
<td>John Smyth</td>
<td></td>
<td>Programming</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>BOSTON-EMPLOYEES</td>
<td>Mary Jones</td>
<td></td>
<td>DBA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>BOSTON-EMPLOYEES</td>
<td>Joe Green</td>
<td></td>
<td>Programming</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CHICAGO-EMPLOYEES</td>
<td>Joan White</td>
<td></td>
<td>Data Entry Clerk</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CHICAGO-EMPLOYEES</td>
<td>David Kelly</td>
<td></td>
<td>DBA</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DENVER-EMPLOYEES</td>
<td>Mel Smith</td>
<td></td>
<td>Data Entry Clerk</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DENVER-EMPLOYEES</td>
<td>Mario Jeni</td>
<td></td>
<td>DBA</td>
<td></td>
</tr>
</tbody>
</table>
```

**Table Extraction Statistics**
How Totals Processing Works

CA Culprit processes data in two input/output phases:

1. The *extract phase (CULL)*:
   
   a. Executes type 7 logic against the input data.
   
   b. Outputs a temporary work file, called the *extract file*.

2. The *output phase (CULE)*:
   
   a. Reads the extract file *after it has been sorted*.
   
   b. Outputs the report.

   When CA Culprit sends the detail report records to the printer, it looks for control breaks and executes type 8 logic.

**Code for Sample Totals-Processing Report**

This is a typical CA Culprit run where sorting and control breaks are processed by employee status within department. At each control break the average starting age for employees is calculated in type 8 logic.

**The Code**

```
IN 200
  REC LAST-NAME 15 10
  REC STATUS 82 2
  REC START-YEAR 97 2 2
  REC BIRTH-YEAR 109 2 2
  REC DEPARTMENT 115 25
01SORT DEPARTMENT - STATUS 0 $Fields are written to the extract file
013 AVERAGE START AGE BY DEPARTMENT AND STATUS
010 TOTAL-MESSAGE ' ' 
010 EMPLOYEE-COUNT 1
010 AGE
010 AVERAGE-AGE
0151*000 EMPLOYEE-COUNT $Type 5 lines build the
0151*010 DEPARTMENT HH 'DEPARTMENT' $extract file with the
0151*020 LAST-NAME HH 'EMPLOYEE NAME' $current values of each
0151*030 STATUS HH 'STATUS ' 'CODE' $field
0151*040 START-YEAR SZ=5 HH 'START' 'YEAR'
0151*050 BIRTH-YEAR SZ=5 HH 'BIRTH' 'YEAR' $Type 5 lines also
0151*055 AGE SZ=2 HH 'AGE' $format the print line
```
### COMPUTER OPERATIONS

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>CODE</th>
<th>YEAR</th>
<th>STATUS</th>
<th>START YEAR</th>
<th>BIRTH YEAR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TERNER</td>
<td>02</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIPSICH</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KRAAMER</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KLWELLEN</td>
<td>01</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GARDNER</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FONRAD</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FERNDALE</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRANE</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Average Age:** 27
- **Number of Employees by Status Code:** 7
- **Number of Employees by Department:** 8

### BRAINSTORMING

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>CODE</th>
<th>YEAR</th>
<th>STATUS</th>
<th>START YEAR</th>
<th>BIRTH YEAR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARM</td>
<td>05</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANDALE</td>
<td>03</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Average Age:** 32
- **Number of Employees by Status Code:** 6
- **Number of Employees by Department:** 3

### BLUE SKIES

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>CODE</th>
<th>YEAR</th>
<th>STATUS</th>
<th>START YEAR</th>
<th>BIRTH YEAR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOON</td>
<td>01</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DONOVAN</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLOUD</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Average Age:** 27
- **Number of Employees by Status Code:** 6
- **Number of Employees by Department:** 3

### ACCOUNTING AND PAYROLL

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>CODE</th>
<th>YEAR</th>
<th>STATUS</th>
<th>START YEAR</th>
<th>BIRTH YEAR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLOOMER</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Average Age:** 27
- **Number of Employees by Status Code:** 6
- **Number of Employees by Department:** 6

---

**RAI18 018400 COMPUTE AGE / EMPLOYEE-COUNT AVERAGE-AGE**

- Uses extract file data

**CA IDMS - 19.0**

15-Jan-2018 36/225
<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>CODE</th>
<th>YEAR</th>
<th>STATUS</th>
<th>START</th>
<th>BIRTH</th>
<th>YEAR</th>
<th>AGE</th>
<th>AVERAGE AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER OPERATIONS</td>
<td>KAHALLY</td>
<td>05</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTIVE ADMINISTRATION</td>
<td>WILDER</td>
<td>01</td>
<td>73</td>
<td></td>
<td></td>
<td></td>
<td>33</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>EXECUTIVE ADMINISTRATION</td>
<td>RUPEE</td>
<td>01</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td>33</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>EXECUTIVE ADMINISTRATION</td>
<td>WILDER</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td>55</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>DOUG</td>
<td>01</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td>51</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>GALLWAY</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>47</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>GARFIELD</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td>45</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>GRANGER</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>58</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>HEARONETZ</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>56</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>JACOB</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>JENSEN</td>
<td>01</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td>48</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>LITERATA</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>55</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>O‘HEARN</td>
<td>01</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>INTERNAL SOFTWARE</td>
<td>TYRO</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>55</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>FITZHUGH</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>56</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>ORGRATIZI</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>51</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>PEOPLES</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>49</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>JOHNSON</td>
<td>05</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td>45</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>ANGELO</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td>57</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>BANK</td>
<td>01</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td>50</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>BOWER</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td>39</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>JACKSON</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td>50</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>REPORT NO. 01</td>
<td>AVERAGE START AGE BY DEPARTMENT AND STATUS</td>
<td>mm/dd/yy</td>
<td>PAGE 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEPARTMENT</td>
<td>EMPLOYEE NAME</td>
<td>CODE</td>
<td>YEAR</td>
<td>STATUS</td>
<td>START</td>
<td>BIRTH</td>
<td>YEAR</td>
<td>AGE</td>
<td>AVERAGE AGE</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>MCDOUGALL</td>
<td>01</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>59</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>PENMAN</td>
<td>01</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>ZEDI</td>
<td>05</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>CLOTH</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td>45</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>FINN</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>TIME</td>
<td>01</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td>58</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>WILCO</td>
<td>01</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td>50</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>KASPAR</td>
<td>04</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
It is easier to understand how subtotaling and control break processing works if we look at the role each parameter plays in the sample report.

- The Extract Processing Phase (CULL) (see page 38)
- The Output Phase (CULE) (see page 38)
- The Processing Steps when an Extract File Is Read (see page 39)

### The Extract Processing Phase (CULL)

- A single type 7 logic statement and an automatic TAKE combine to calculate an employee's age. When the implied TAKE is executed in type 7 logic, a record consisting of the two fields on the SORT parameter and the seven fields on the type 5 parameters is written to the extract file.

- Type 5 lines perform two functions, in this order:
  1. Build an extract record, which consists of all the fields on the type 5 parameters and all of the fields on the SORT parameter.
  2. Establish the format of a detail line on the report from the size and auto-header code included on the line.

Because there is no DROP or SELECTION logic, the processing of every record from the input file results in each record being written to the extract file. No sorting or totaling is done by CA Culprit during this input processing; only the instructions coded in type 7 logic are executed.

When CA Culprit is through processing the input file, the sort specified on the SORT parameter is executed by the installation’s sort utility. The extract file is sorted in the order of the sort keys, which are part of each extract record. The extract file is then ready for output processing.

### The Output Phase (CULE)

- The sorted extract file records provide the input for control breaks and printing the report.

- The only fields from the extract record used in the type 8 logic or on the type 6 lines are those that are coded on the SORT parameter or on the type 5 lines. Work fields that are not part of the extract record are also available for use.

- Type 8 logic is executed whenever a control break is encountered on STATUS or DEPARTMENT.

- The TAKE after sequence 400 sends the type 6 line to be printed.

- One subtotal accumulator (bucket) is set up for each field at each break level for numeric fields named on type 5 lines and referenced in type 8 logic or on type 6 parameters (AGE and EMPLOYEE-COUNT). No other fields are automatically subtotaled.
The Processing Steps when an Extract File Is Read

CA Culprit performs the following steps when an extract file is read:

1. Sends each detail extract record to the printer. Grand total break processing is performed at the end-of-file.

2. Updates all subtotal accumulators.
   In our example, the detail values for AGE and EMPLOYEE-COUNT are added to all three accumulators. Since none of the other numeric fields from the type 5 line are referenced in type 6 or type 8 code, no other automatic totaling is performed. EMPLOYEE-COUNT serves as a counter because it always has a value of 1 on the type 5 line. It is not printed on the detail line of the report because position *000 (0000 will work too) is specified on the type 5 line, but it is still subtotaled.

3. Looks ahead to the next detail extract record, searching its sort keys for a control break. If there is no control break, then it processes that next detail extract record (step 1 above). If there is a control break:
   - CA Culprit holds that next detail extract record aside and executes type 8 logic for the appropriate control break level.
   - TAKE in type 8 logic causes all of the type 6 lines to be sent to the printer.

   If a control break occurs on more than one level at the same time, CA Culprit executes type 8 logic once for each level starting with the lowest level number first. In our example, when DEPARTMENT breaks, STATUS always has to break because it is a LEVL 1. Type 8 logic is executed for STATUS first.

4. Executes control break spacing after exiting type 8 logic because of a TAKE or a DROP. If multiple control breaks occur, such as when DEPARTMENT and STATUS break together, the control break spacing is only performed for the highest level.

5. Moves 0 to the accumulator for each field being totaled at the control break level being processed.

6. Continues with step 1 (above) for the next record read from the extract file.

7. Looks again to the next detail record from the extract file.
How to Produce a CA Culprit Report

Producing a CA Culprit report involves four basic steps:

1. Planning the report
2. Writing the code
3. Executing the code
4. Debugging the code

Step 1. Planning the Report

Define the report objectives:

1. What information do you want in the report?
2. Do you need each data occurrence?
3. Do you need totals? Subtotals?

Identify the data source:

1. What is the name of the file?
2. What data is in the file?
3. How is the file organized?

Define the report output:

1. Do you need a title? Subtitles?
2. What columns do you need?
3. What are the column headings?
4. Do you need total line labels?

Define the processing functions required to obtain the information needed in the report:

1. Are certain data items to be selected?
2. Should the data be sorted?
3. What calculations (if any) are needed?

4. Is testing (comparisons) needed between data items?

5. Will branching be used?

6. What logical sequence is required?

7. Other procedures?

Step 2. Writing the Code

You tell CA Culprit what to do and when to do it by coding instructions on CA Culprit parameters. You identify each CA Culprit parameter by a word or number you code in a specific column position.

A few general rules apply to all CA Culprit coding:

- Enter code in uppercase letters.

- Enter coding in the first ten columns in exactly the correct position

  - Mnemonic code that describes input files starts in column 2:
    
    IN 200 F 400
    
    REC EMP-NAME

  - Numeric code starts in column 2:
    
    0151*010 ' ' 

  - Mnemonic code that indicates copying and modifying stored code starts in column 1:
    
    USE
    
    =COPY
    
    =MACRO
    
    =MEND

- Continuation lines start in column 1:
  
  * . . .

- Entries following column 10 need not begin in a specific column, but must be separated by a comma or at least one space.

Step 3. Executing the Code

You execute CA Culprit code with Job Control Language (JCL). Consult your data-processing department for the JCL used to run CA Culprit reports at your site.
Step 4. Debugging the Code

No matter how careful you are, errors in your code will occur from time to time. At the end of each run, CA Culprit provides listings that flag errors:

- The Sequential Parameter Listing
- The Input Parameter Listing
- Run-Time Messages

The Debugging a CA Culprit Report section describes the CA Culprit listings and how to use them to debug CA Culprit code.

CA Culprit Reports

This topic describes how to perform the following tasks with CA Culprit Reports.

- Add Programming Logic to a Report (see page 42)
- Additional Standard Options (see page 49)
- Control Total Lines (see page 56)
- Enhance the Basic Report (see page 73)
- Generate a Basic Report from Standard Files (see page 83)
- Generate Reports From Database Files (see page 85)

Add Programming Logic to a Report

With CA Culprit you can produce more sophisticated reports than basic formatted file listings. Use CA Culprit to add the following programming logic to a report:

- Count specific items.
- Perform arithmetic operations.
- Test data against specified conditions.
- Direct the flow of processing.

Code Programming Logic

Use type 7 parameters to code programming logic for a report. Type 7 parameters instruct CA Culprit to process data while it reads from the input file.

Code type 7 parameters from left to right in the following order:

1. A 2-digit report number.
2. The parameter type (7).

3. A 3-digit sequence number indicating the order in which CA Culprit executes the parameter.

4. The instruction you want to execute.

Use Counters

You can use counters in CA Culprit to track the number of times an event occurs during processing. Specify a conditional counter to increment a work field, or use an automatic counter to keep a running total of a work field each time counting occurs. You can print the total count for a work field on a report.

Use Conditional Counters

You can code conditional counters to increment a count of a specific work field. The following example shows a computation that adds 1 to the value of the work field when counting occurs:

```
017100 COUNTER-A + 1 COUNTER-A
```

Use Automatic Counters

You can code automatic counters to keep a running total of a work field. You can either print or suppress the current count on a report. The following examples show how to code automatic counters:

- Code a value of 1 to add 1 to the value of the work field each time counting occurs:
  
  ```
  010 COUNTER-B 1
  ```

- Change the name of the work field on a type 5 parameter to print the value each time counting occurs:
  
  ```
  0151*010 COUNTER-B (Prints the value each time)
  ```

  ```
  0151*000 COUNTER-B (Does not print the value)
  ```

- Change the name of the work field on a type 6 parameter to print the total count on a report:
  
  ```
  0161*010 COUNTER-B
  ```

  Type 6 parameters are coded the same way as type 5 parameters.

For more information on parameter types, see Controlling Total Lines.

Example 1: Print the Count of each Occurrence

This example shows how to print a report that assigns a number to each employee in the public relations department. Use counters to add 1 to the value of each numeric work field. CA Culprit processes each new employee and prints the value of the work field.

Parameters
- **01OUT D**  
  Suppresses the automatic total of the employee number column.

- **010 COUNT**  
  Establishes the numeric work field COUNT.

- **0151*030 COUNT**  
  Prints the value of the work field COUNT.

- **017010 COUNT + 1 COUNT**  
  Adds 1 to the value contained in the work field COUNT.

**Code**

col. 2

```
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 20
REC DEPARTMENT 115 25
SEL DEPARTMENT EQ 'PUBLIC RELATIONS'
013 EMPLOYEE COUNT01OUT D 0010 COUNT0151*010 DEPARTMENT HH 'DEPARTMENT'
0151*020 EMPLOYEE HH 'EMPLOYEE NAME' 00151*030 COUNT SZ=3 HH 'EMPLOYEE NUMBER'
017010 COUNT + 1 COUNT
```

**Result**

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>DEPARTMENT</th>
<th>EMPLOYEE COUNT</th>
<th>mm/dd/yy PAGE</th>
<th>EMPLOYEE NAME</th>
<th>EMPLOYEE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLIC RELATIONS</td>
<td>MICHAEL ANGELO</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>DONTE BANK</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>JOCK JACKSON</td>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>CAROL MCDUGALL</td>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>LAURA PENMAN</td>
<td></td>
<td></td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>BETSY ZEDI</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2: Print a Total Count**

This example shows how to print a report that counts the number of employees that are being processed. Use counters to add 1 to the value of the numeric work field. CA Culprit uses a type 6 parameter to print the final value of the work field.

**Parameters**

- **010 COUNT**  
  Establishes the numeric work field.

- **0161*040 'NUMBER OF EMPLOYEES' 0161*050 COUNT**  
  Type 6 parameters used to label the print line and obtain the last value in the work field COUNT.

- **017010 COUNT + 1 COUNT**  
  Adds 1 to the value contained in the work field COUNT.

**Code**

col. 2

```
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 20
REC DEPARTMENT 115 25
```
Example 3: Use the Value of a Work Field

This example shows how to create a report that counts the number of employees that are being processed. Use a counter to increment the value of the numeric work field by 1. CA Culprit uses a type 6 parameter to print the value of the work field.

Parameters

0161*040 'NUMBER OF EMPLOYEES'
0161*050 COUNT

- **010 COUNT**
  Establishes a numeric work field that has a value of 1.

- **0151*000 COUNT**
  0151 keeps an automatic running total of the work field COUNT on a type 5 parameter. 000 suppresses the printing of the field value on the report. You can use more than one type 5 parameter.

- **0161*040 and 0161*050**
  Type 6 parameters that label the print line and obtain the last value in the work field COUNT.

Complete Code

```plaintext
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 20
REC DEPARTMENT 115 25
SEL DEPARTMENT EQ 'PUBLIC RELATIONS'
013 EMPLOYEE COUNT010 COUNT0151*010 DEPARTMENT HH 'DEPARTMENT'
0151*020 EMPLOYEE HH 'EMPLOYEE NAME'
0161*040 'NUMBER OF EMPLOYEES'
0161*050 COUNT017010 COUNT + 1 COUNT
```

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE COUNT</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLIC RELATIONS</td>
<td>MICHAEL ANGELO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>DONTE BANK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>JOCK JACKSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>LAURA PENMAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>Betsy Zedi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMBER OF EMPLOYEES</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Perform Arithmetic Operations

You can perform simple and complex arithmetic operations in your CA Culprit report.

Perform Arithmetic Operations

To perform one-step operations, code the following:

- A numeric field to use as the operand.
- An operator with a space on either side.
- The work field to contain the results.

017010 YR-SALARY / 52 WK-SALARY

To perform a complex statement with two or more operations, enter the word COMPUTE:

017010 COMPUTE (SALARY + 2000) X 0.10 BONUS

Perform Arithmetic When a Condition is Met

You can use an arithmetic operation alone, or as a result of a conditional statement when the condition is met.

The following example shows how to code an arithmetic operation as a result of a conditional statement:

017010 IF SALARY LE 19000 100
017020 ...
017100 SALARY / 52 WK-SALARY

Example: Use Arithmetic to Create a Report

This example shows how to print a report that compares employee annual earnings to a base salary of $19,000. Employees who earn more than the base amount are considered salaried. Employees who earn less than the base amount are considered hourly. The report calculates an hourly wage for all nonsalaried employees.

Testing and calculations for this report are on type 7 parameters.

Parameters

- 01OUT D
  Suppresses totals

- 010 AMOUNT DP=2
  A work field that holds computation results containing two decimal places.
• **017010 IF SALARY LE 19000 500**
  A conditional statement that tests for a range less than or equal to 19000. If the test is true, processing skips to line 500 of the type 7 parameters. If the test is not true, the type 7 statements are processed in sequence until TAKE is executed.

• **017 MOVE 'SALARIED' TO MESSAGE**
  Places SALARIED in the MESSAGE work field when SALARY is greater than 19000.

• **017 MOVE SALARY TO AMOUNT**
  Places the salary amount in the work field.

• **017 TAKE**
  Causes all the type 5 parameter fields to be written to a work file (the extract file) for use in later processing.

• **017500 MOVE 'HOURLY ' TO MESSAGE**
  Places the word HOURLY in the MESSAGE work field when the salary is less than 19000.

• **017 COMPUTE (SALARY / 52) / 40 AMOUNT**
  Specifies a two-step computation. The result is placed in the AMOUNT work field.

• **TAKE**
  An implied TAKE that causes all type 5 fields to be written to a work file (the extract file) for use in later processing. This TAKE is supplied by CA Culprit; it is not required in the code.

**Code**

```plaintext
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC LAST-NAME 15 15
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
01OUT DO1SORT LAST-NAME
013 EMPLOYEE COMPENSATION STATUS
010 MESSAGE ' ' ' ',
010 AMOUNT DP=2
0151*010 EMPLOYEE Sz=20 HH 'EMPLOYEE' 'NAME'
0151*020 MESSAGE HH 'STATUS'
0151*030 AMOUNT Sz=10 F$ HH 'SALARY'
017010 IF SALARY LE 19000 500 MOVE 'SALARIED' TO MESSAGE017 MOVE SALARY TO AMOUNT017 TAKE017500 MOVE 'HOURLY ' TO MESSAGE017 COMPUTE (SALARY / 52) / 40 AMOUNT
```

**Result**

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>EMPLOYEE COMPENSATION STATUS</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EMPLOYEE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROY</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANDALE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MICHAEL</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANGELO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HARRY</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ARM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MONTE</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BANK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JUNE</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLOOMER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARLES</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOXER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>BREEZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BREEZE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RALPH</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TYRO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RICHARD</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WAGNER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROGER</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WILCO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROBBY</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WILDER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BETSY</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ZEDI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$33,500.00</td>
</tr>
<tr>
<td>$8.65</td>
</tr>
<tr>
<td>$46,000.00</td>
</tr>
<tr>
<td>$80,000.00</td>
</tr>
<tr>
<td>$7.21</td>
</tr>
<tr>
<td>$38,500.00</td>
</tr>
<tr>
<td>$38,000.00</td>
</tr>
<tr>
<td>$38,000.00</td>
</tr>
<tr>
<td>$20,000.00</td>
</tr>
<tr>
<td>$47,000.00</td>
</tr>
<tr>
<td>$80,000.00</td>
</tr>
<tr>
<td>$90,000.00</td>
</tr>
<tr>
<td>$37,000.00</td>
</tr>
</tbody>
</table>
Direct Processing Flow

You can code a repeated procedure once and direct CA Culprit to repeat the procedure at any time.

To direct the processing flow, code the following:

- The word **PERFORM** on a processing parameter (type 7 or type 8). For more information about Type 8 parameters, see Controlling Total Lines.
- The parameter sequence number where CA Culprit starts to process the repetitive statements.
- The word **RETURN** at the end of the block of repetitive statements. Keep the related PERFORM /RETURN within the same parameter type.

Example: Code a Repeated Procedure

This example shows how to print a report that calculates employee bonuses based upon individual salary and longevity. You use PERFORM/RETURN statements to execute the repetitive computation based on longevity.

**Parameters**

- **100**
  Directs processing to sequence line 100 when SALARY is greater than $15,000.

- **PERFORM 500**
  Directs processing to sequence line 500 to calculate the longevity factor.

- **010**
  Directs processing to sequence line 010 if the calculated bonus is less than $500.

- **TAKE**
  Causes the type 5 line to be extracted.

- **200**
  Directs processing to sequence line 200 when SALARY is greater than $25,000.

- **110**
  Directs processing to sequence line 110 if the calculated bonus is less than $1200.

- **500**
  Tests START-YEAR. Employees who started before 1981 receive an additional $300 bonus. Those starting after 1981 receive an additional $100 bonus.

- **017520 RETURN**
  Directs processing back to the statement following the last PERFORM statement executed.

**Code**

```
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
```
Additional Standard Options

After you have created your basic CA Culprit report, you can use some of the CA Culprit optional capabilities to produce multiple reports in a single run, sequence report output, select data for processing, and specify multiple detail lines.

- Produce Multiple Reports (see page 49)
- Sequence Report Output (see page 51)
- Use Selective Processing (see page 52)
- Assign Information to Multiple Detail Lines (see page 55)

Produce Multiple Reports

In a single CA Culprit run, you can produce up to 100 reports that use the same data set. Assign a 2-digit report number to each report you want to create.

Code the report number on report-specific parameters.
Example: Produce Multiple Reports in a Single Run

**Objective**

We are going to produce three different reports in one run from the Commonweather personnel file.

**Parameters**

1. **01OUT 60 013 EMPLOYEE SALARIES 0151*010 EMPLOYEE SZ=20 HR 0151*020 SALARY SZ=11 F$ HR**
   
   All parameters having 01 coded in columns 2 and 3 belong to report 01.
   
   **OUT 60** specifies 60-character lines.
   
   EMPLOYEE SALARIES is the title.
   
   EMPLOYEE and SALARY are the two data fields printed for this report. Both columns have a specified width. SALARY is formatted with dollar signs.

2. **02OUT 80 023 DEPARTMENT PERSONNEL LIST 0251*010 DEPARTMENT SZ=25 HR 0251*020 TITLE HR 0251*030 EMPLOYEE SZ=20 HR**
   
   All parameters having 02 coded in columns 2 and 3 belong to report 02.
   
   **OUT 80** specifies 80-character lines.
   
   DEPARTMENT PERSONNEL LISTING is the title for report 02.
   
   Report 02 uses information from the DEPARTMENT, TITLE, and EMPLOYEE input fields.

3. **03OUT 80 033 EMPLOYEE DEPARTMENTS 0351*010 EMPLOYEE SZ=20 HR 0351*020 DEPARTMENT SZ=25 HR**
   
   All parameters having 03 coded in columns 2 and 3 belong to report 03.
   
   **OUT 80** specifies 80-character lines.
   
   EMPLOYEE DEPARTMENTS is the title for report 03.
   
   Report 03 uses information from the EMPLOYEE and DEPARTMENT input fields.

**Complete Code**

```
col. 2
IN  200 F 400 PS(TAPE)
REC EMPLOYEE 5 25 'EMPLOYEE NAME'
REC DEPARTMENT 115 45 'DEPARTMENT'
REC SALARY 160 5 3 DP=2 'ANNUAL SALARY'
REC TITLE 171 20 'TITLE'
01OUT 60013 EMPLOYEE SALARIES 0151*010 EMPLOYEE SZ=20 HR 0151*020 SALARY SZ=11 F$ HR
02OUT 80023 DEPARTMENT PERSONNEL LIST 0251*010 DEPARTMENT SZ=25 HR 0251*030 EMPLOYEE SZ=20 HR
03OUT 80033 EMPLOYEE DEPARTMENTS 0351*010 EMPLOYEE SZ=20 HR
```

**Results**

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>EMPLOYEE</th>
<th>SALARIES</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE NAME</td>
<td>ANNUAL SALARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>$15,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDWARD</td>
<td>HUTTON</td>
<td>$44,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUPERT</td>
<td>JENSON</td>
<td>$82,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIANNE</td>
<td>KIMBALL</td>
<td>$45,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td>$14,500.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRIAN</td>
<td>NICEMAN</td>
<td>$14,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HERBERT</td>
<td>CRANE</td>
<td>$75,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANE</td>
<td>FERDIALE</td>
<td>$22,500.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEORGE</td>
<td>FORRAD</td>
<td>$14,750.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROBIN</td>
<td>GARDNER</td>
<td>$14,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>$20,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERENCE</td>
<td>KLWELLEN</td>
<td>$43,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sequence Report Output

You can sequence alphanumeric and numeric data in your report in ascending or descending order.

How to Do It

Code the **SORT** parameter line with the following information, from left to right:

1. The 2-digit report number.
2. The word **SORT**. There is no space after the report number.
3. The name of the field to be sorted.
4. The sort order:
   - **Blank** or **A** for ascending order
   - **D** for descending order
5. Repeat steps 3 and 4 for any additional sort fields, separating entries by at least one space.

Example: List Employees in Alphabetical Order

**Objective**

The next report lists employees in alphabetical order by performing a sort on employee last name.
Parameters

- **01SORT LAST-NAME**
  01 is the report number. The **SORT** parameter is coded in columns 4 to 8, immediately after the report number. **LAST-NAME** is the sort field. This field is also defined to the report by appearing on the REC parameter.

Complete Code

```plaintext
IN 200 F 400 PS(TAPE)
REC EMPLOYEE      5   25
REC LAST-NAME     15  15
REC SALARY        160  5   3   DP=2
013 EMPLOYEE SALARY LISTING01SORT LAST-NAME0151*010 EMPLOYEE SZ=20   HH 'EMPLOYEE'
0151*020 SALARY     HH 'SALARY'
```

Result

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>EMPLOYEE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>MICHAEL ANGELO</td>
<td>18,000.00</td>
</tr>
<tr>
<td></td>
<td>DONTE BANK</td>
<td>80,000.00</td>
</tr>
<tr>
<td></td>
<td>ALBERT BREEZE</td>
<td>38,000.00</td>
</tr>
<tr>
<td></td>
<td>BETH M. CLOUD</td>
<td>52,750.00</td>
</tr>
<tr>
<td></td>
<td>ALAN DONOVAN</td>
<td>33,500.00</td>
</tr>
<tr>
<td></td>
<td>PERCY EINSTEIN</td>
<td>34,500.00</td>
</tr>
<tr>
<td></td>
<td>JANE FERNDALE</td>
<td>22,500.00</td>
</tr>
<tr>
<td></td>
<td>TOM FITZHUGH</td>
<td>13,000.00</td>
</tr>
<tr>
<td></td>
<td>ROBIN GARDNER</td>
<td>14,000.00</td>
</tr>
<tr>
<td></td>
<td>JENNIFER GARFIELD</td>
<td>65,000.00</td>
</tr>
<tr>
<td></td>
<td>VLADIMIR HEAROWITZ</td>
<td>33,000.00</td>
</tr>
<tr>
<td></td>
<td>EDWARD HUTTON</td>
<td>44,000.00</td>
</tr>
<tr>
<td></td>
<td>JOCK JACKSON</td>
<td>34,000.00</td>
</tr>
<tr>
<td></td>
<td>JAMES JACOBI</td>
<td>55,000.00</td>
</tr>
<tr>
<td></td>
<td>JULIE JANSSEN</td>
<td>37,000.00</td>
</tr>
<tr>
<td></td>
<td>RUPERT JENSON</td>
<td>82,000.00</td>
</tr>
<tr>
<td></td>
<td>MARYLOU JOHNSON</td>
<td>12.00</td>
</tr>
<tr>
<td></td>
<td>CYNTHIA JOHNSON</td>
<td>13,500.00</td>
</tr>
<tr>
<td></td>
<td>DOUGLAS KAALLY</td>
<td>20,000.00</td>
</tr>
</tbody>
</table>

Use Selective Processing

You can refine the scope of your processing by choosing only those records that meet certain criteria.

How to Do It

You can use either a **SELECT** or **BYPASS** parameter. The choice depends on which option allows for the clearest statement of the condition controlling the choice. You can specify more than one selection criterion on a single **SELECT** or **BYPASS** parameter. However, the two parameters cannot be applied together.

To use **SELECT** or **BYPASS**:

1. Code the **SELECT** or **BYPASS** parameter immediately after the REC parameters that describe the file.
2. Enter the data item and test condition that the data must meet; such as, **EQ** (equal to), **GT** (greater than), **LT** (less than).

3. Join multiple criteria with logical operators:
   - **AND**, to indicate that all conditions must be true
   - **OR**, to indicate that only one condition need be true

**Example 1: Select Records with Single Criteria**

**Objective**

We are going to modify our report by listing only employees earning $50,000.00 and over.

The following code selects the appropriate records:

**Parameters**

- **SEL SALARY GE 50000.00**
  - SEL SALARY GE 50000.00 specifies selection of salaries equal to or greater than 50,000.00.

  or

- **BYP SALARY LT 50000.00**
  - BYP SALARY LT 50000.00 specifies omission of salaries less than 50,000.00.

**Complete Code**

```
col. 2
IN  200 F 400 PS(TAPE)
REC EMPLOYEE  5  25
REC LAST-NAME  15  15
REC SALARY 160   5  3 DP=2
013 EMPLOYEE SALARY LISTING
01SORT LAST-NAME
0151*010 EMPLOYEE HH 'EMPLOYEE'
0151*020 SALARY SZ=10 HH 'SALARY'
```

**Result**

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>SALARY (mm/dd/yy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONTE</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>BETH</td>
<td>$52,750.00</td>
</tr>
<tr>
<td>HERBERT</td>
<td>$75,000.00</td>
</tr>
<tr>
<td>JENNIFER</td>
<td>$65,000.00</td>
</tr>
<tr>
<td>HEMIETTA</td>
<td>$240,000.00</td>
</tr>
<tr>
<td>JAMES</td>
<td>$55,000.00</td>
</tr>
<tr>
<td>RUPERT</td>
<td>$82,000.00</td>
</tr>
<tr>
<td>BURT</td>
<td>$54,500.00</td>
</tr>
<tr>
<td>RENE</td>
<td>$85,000.00</td>
</tr>
<tr>
<td>DANIEL</td>
<td>$72,000.00</td>
</tr>
<tr>
<td>THEMIS</td>
<td>$108,000.00</td>
</tr>
<tr>
<td>ELEANOR</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>JOHN</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>ROGER</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>ROBBY</td>
<td>$90,000.00</td>
</tr>
<tr>
<td></td>
<td>1,291,250.00</td>
</tr>
</tbody>
</table>
Example 2: Select Records with Multiple Criteria

Objective

Our next report selects employees earning $15,000 or less and those earning between $35,000 and $50,000.

Parameters

- SEL SALARY EQ (35000 TO 50000) OR SALARY LE 15000
- SEL SALARY EQ (35000 TO 50000)

SEL SALARY EQ (35000 TO 50000) specifies the first selection criteria.

OR relates the first test to the to the second by allowing selection of a record that meets only one of the specified conditions.

SALARY LE 15000 specifies the alternative record selection criteria.

The following BYPASS parameter achieves the same results:

Parameters

- BYP SALARY EQ (15000.01 TO 34999.99) OR SALARY GT 50000
- BYP SALARY EQ (15000.01 TO 34999.99)

BYP SALARY EQ (15000.01 TO 34999.99) specifies the first record bypass criteria.

OR relates the first test to the second by allowing bypass of a record that meets only one of the specified conditions.

SALARY GT 50000 specifies the alternative record bypass criteria.

Complete Code

col. 2

IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC LAST-NAME 15 15
REC SALARY 160 5 3 DP=2
3 EMPLOYEE SALARY LISTING
01SORT LAST-NAME
0151*010 EMPLOYEE HH 'EMPLOYEE'
0151*020 SALARY SZ=10 HH 'SALARY'

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE</th>
<th>SALARY LISTING</th>
<th>mm/dd/yy PAGE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARRY</td>
<td>ARM</td>
<td></td>
<td></td>
<td>46,000.00</td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td></td>
<td></td>
<td>15,000.00</td>
</tr>
<tr>
<td>CHARLES</td>
<td>BOWER</td>
<td></td>
<td></td>
<td>38,500.00</td>
</tr>
<tr>
<td>C</td>
<td>BREEZE</td>
<td></td>
<td></td>
<td>38,000.00</td>
</tr>
<tr>
<td>TERRY</td>
<td>CLOTH</td>
<td></td>
<td></td>
<td>38,000.00</td>
</tr>
<tr>
<td>CAROLYN</td>
<td>CROW</td>
<td></td>
<td></td>
<td>37,500.00</td>
</tr>
<tr>
<td>PHINEAS</td>
<td>FINN</td>
<td></td>
<td></td>
<td>45,000.00</td>
</tr>
<tr>
<td>TOM</td>
<td>FITZMAH</td>
<td></td>
<td></td>
<td>12,000.00</td>
</tr>
<tr>
<td>GEORGE</td>
<td>FORNAD</td>
<td></td>
<td></td>
<td>14,750.00</td>
</tr>
<tr>
<td>ROB</td>
<td>GARDNER</td>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>EDWARD</td>
<td>HUTTON</td>
<td></td>
<td></td>
<td>44,000.00</td>
</tr>
<tr>
<td>JULIE</td>
<td>JENSEN</td>
<td></td>
<td></td>
<td>37,000.00</td>
</tr>
<tr>
<td>CYNTHIA</td>
<td>JOHNSON</td>
<td></td>
<td></td>
<td>13,500.00</td>
</tr>
<tr>
<td>MARIANNE</td>
<td>KIMBALL</td>
<td></td>
<td></td>
<td>45,000.00</td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td></td>
<td></td>
<td>14,500.00</td>
</tr>
<tr>
<td>TERENCE</td>
<td>KLWELLEN</td>
<td></td>
<td></td>
<td>43,000.00</td>
</tr>
<tr>
<td>SANDY</td>
<td>KRAEMER</td>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>LARRY</td>
<td>LITERATA</td>
<td></td>
<td></td>
<td>37,500.00</td>
</tr>
<tr>
<td>RICHARD</td>
<td>MUNYON</td>
<td></td>
<td></td>
<td>36,000.00</td>
</tr>
<tr>
<td>BRIAN</td>
<td>NICEMAN</td>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>KATHERINE</td>
<td>O'HEARN</td>
<td></td>
<td></td>
<td>42,500.00</td>
</tr>
</tbody>
</table>
Assign Information to Multiple Detail Lines

Up to this point we have used the type 5 parameter to assign several pieces of information to a single print line. In CA Culprit this is called a **detail line**. For example,

```
MAX DELANO 12 ORCHARD LANE LIPTON NJ 07080
```

You can also assign information to multiple detail lines that print as a single block of information. For example,

```
MAX DELANO first detail line 12 ORCHARD LANE second detail line LIPTON N J 07080 third detail line
```

**How to Do It**

To specify multiple detail lines on a type 5 parameter, change the number in column 5 to indicate the line on which the information should appear:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0151*010</td>
<td>NAME first detail line</td>
</tr>
<tr>
<td>0152*010</td>
<td>STREET-ADDRESS second detail line</td>
</tr>
<tr>
<td>0153*010</td>
<td>TOWN third detail line</td>
</tr>
<tr>
<td>0153*020</td>
<td>ZIPCODE</td>
</tr>
</tbody>
</table>

**Example: Use Multiple Detail Lines**

**Objective**

Our next report uses multiple detail lines to produce an employee address list. The employee name, street number, city, state, and zip code appear on three separate lines.

**Parameters**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0151*010</td>
<td>LAST-NAME</td>
</tr>
<tr>
<td>0151*012</td>
<td>FIRST-NAME</td>
</tr>
<tr>
<td>0152*020</td>
<td>STREET</td>
</tr>
<tr>
<td>0153*020</td>
<td>CITY</td>
</tr>
<tr>
<td>0153*025</td>
<td>STATE</td>
</tr>
<tr>
<td>0153*030</td>
<td>ZIP-CODE</td>
</tr>
</tbody>
</table>

1 specifies placement of the last name and first name on the first detail line.

2 specifies placement of the street on the second detail line.

3 specifies placement of the city, state, and zip code on the third detail line.

**Complete Code**

col. 2
### Control Total Lines

Advantage CA-Culprit generates automatic total amounts for numeric fields that appear on type 5 lines. You can label and control the printing of the total lines by using parameters that act upon data before it is printed on the report (the output phase).

When working with total amounts, use the following parameters:

- Type 6 parameters to control the print lines.
- An OUT parameter with the T option to print total amounts only. (Use the D option to print detail lines only).
- A SORT parameter to obtain subtotals.
- Type 8 parameters to process the data.

### Selectively Print Total Amounts

You can print only the totals you want by using type 6 parameters, and entering the following information from left to right:

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>EMPLOYEE ADDRESS LIST</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGELO</td>
<td>MICHAEL</td>
<td>507 CISTINE DR WELLESLEY</td>
<td>MA</td>
</tr>
<tr>
<td>BABBIT</td>
<td>HERBIE</td>
<td>30 HERON AVE KINGSTON</td>
<td>NJ</td>
</tr>
<tr>
<td>BANK</td>
<td>DONTE</td>
<td>45 EAST GROVE DR HANIBAL</td>
<td>MA</td>
</tr>
<tr>
<td>BLOOMER</td>
<td>DUDY</td>
<td>14 ZITHER TERR LEXINGTON</td>
<td>MA</td>
</tr>
<tr>
<td>BREEZE</td>
<td>ALBERT</td>
<td>100 BOARDWALK OCEAN CITY</td>
<td>NJ</td>
</tr>
</tbody>
</table>
• A 2-digit report number

• The type 6 parameter

• The number of the total line

• The column placement consisting of either an asterisk followed by a 3-digit number for relative placement, or a 4-digit number for specific placement.

• The spacing specification

• The name of the field you want to print on the total line.

• Additional options, such as size, number of decimal places, and special formatting.

Example

The coding example below uses the following parameters to print only the Salary total:

• 01
  The number of the report.

• 6
  Specifies a type 6 parameter.

• 1
  Specifies the first total line.

• *055
  Specifies automatic page formatting with the salary total printed under the list of individual salaries.

• 0
  Specifies a blank line before the total is printed.

• SALARY
  Specifies the field to total.

• SZ=11 F$
  Specifies the size of the total field to avoid truncation. The total is formatted with a dollar sign.

Coding Example

The following code prints only the Salary total:

```
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE      5   25
REC FIRST-NAME    5   10
REC LAST-NAME     15  15
REC START-YEAR    97  2   2
REC BIRTH-YEAR    109 2   2
REC DEPARTMENT    115 25
REC SALARY        160 5   3  DP=2
013 EMPLOYEE SALARY LISTING BY DEPARTMENT
01SORT DEPARTMENT 0 SALARY D EMPLOYEE
```
### Coding Example Result

The following example displays the output from the above coding example, for printing only the Salary total:

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE SALARY LISTING BY DEPARTMENT</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENT</td>
<td>EMPLOYEE NAME</td>
<td>BIRTH</td>
<td>START</td>
<td>ANNUAL SALARY</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>JENSON</td>
<td>RUPERT</td>
<td>48</td>
<td>29</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>KIMBALL</td>
<td>MARIANNE</td>
<td>49</td>
<td>19</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>HUTTON</td>
<td>EDWARD</td>
<td>41</td>
<td>7</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>BLOOMER</td>
<td>JUNE</td>
<td>60</td>
<td>5</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>KING</td>
<td>DORIS</td>
<td>60</td>
<td>16</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>NICEMAN</td>
<td>BRIAN</td>
<td>55</td>
<td>6</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>CLOTH</td>
<td>TERRY</td>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>TIME</td>
<td>MARK</td>
<td>58</td>
<td>5</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>KASPAR</td>
<td>JOE</td>
<td>40</td>
<td>5</td>
</tr>
</tbody>
</table>

$214,500.00

### Label Total Lines

You can label each printed total by using type 6 parameters, and entering the following information:

- A type 6 parameter
- The text of the total line label, not a field name, enclosed in single quotation marks.

### Example

The coding example below uses the following parameters to place a Salary label on a total line:

- 01
  - Specifies report 01.
- 6
  - Specifies a type 6 parameter.
- 1
  - Specifies the first total line.
- *040
  - Specifies automatic page formatting with the label starting to the left of the grand total.
- 0
  - Specifies a blank line before the total line.
• 'SALARY TOTAL=' Specifies the text for the total line label.

Coding Example

The following example displays the code for placing a Salary label on a total line:

```
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC FIRST-NAME 5 10
REC LAST-NAME 15 15
REC START-YEAR 97 2 2
REC BIRTH-YEAR 19 2 2
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
013 EMPLOYEE SALARY LISTING
0151*010 DEPARTMENT HH 'DEPARTMENT'
0151*020 LAST-NAME SZ=10 HH 'EMPLOYEE NAME'
0151*030 FIRST-NAME
0151*040 BIRTH-YEAR HH 'BIRTH' 'YEAR'
0151*050 START-YEAR HH 'START' 'YEAR'
0151*055 SALARY SZ=9 F$ HH 'ANNUAL SALARY'
0161*055
'SALARY TOTAL=' SALARY SZ=9 F$
```

Coding Example Result

The following example displays the output of the above code to place a label on a total line:

```
REPORT NO. 01 EMPLOYEE SALARY LISTING mm/dd/yy PAGE 1

DEPARTMENT EMPLOYEE NAME BIRTH START ANNUAL SALARY
ACCOUNTING AND PAYROLL BLOOMER JUNE 69 5 $35,000.00
ACCOUNTING AND PAYROLL HUTTON EDWARD 41 7 $44,000.00
ACCOUNTING AND PAYROLL JENSON RUPERT 48 29 $82,000.00
INTERNAL SOFTWARE LITERATA LARRY 55 9 $37,500.00
INTERNAL SOFTWARE O’HEARN KATHERINE 54 4 $42,500.00
INTERNAL SOFTWARE TYRO RALPH 55 21 $20,000.00
EXECUTIVE ADMINISTRATION HENDON HENRIETTA 33 21 $240,000.00
EXECUTIVE ADMINISTRATION PAPAIZEUS THEMIS 35 7 $100,000.00
EXECUTIVE ADMINISTRATION RUPEE JOHN 33 23 $80,000.00

SALARY TOTAL= $2,522,500.00
```

Generate Only Total Amounts

You can produce a report that contains only totals of numeric fields that appear on type 5 lines. You specify a totals-only report using the following information:

- The OUTPUT parameter with the totals-only option (OUT T)
- A type 5 parameter for each numeric field to be totaled
- A type 6 parameter for totals to be printed

Example

The coding example below uses the following parameters, to generate the total of all the salaries in a company:
• T
  Specifies totals only.

• 0151*020 SALARY
  Specifies the type 5 parameter that retrieves and causes the individual salaries to be totaled.

• 6
  Identifies the parameter type.

• 'ALL DEPARTMENTS'
  Specifies the label for the total line.

• SALARY
  Specifies the field whose total is to be printed.

Coding Example

The following example displays the code for generating the total of all the salaries in a company:

```plaintext
col. 2
IN 2 F 4 PS(TAPE)  
REC SALARY 16 5 3 DP=2 'SALARY' 'TOTAL'   
1OUT T  
151*020 SALARY  
1611 'ALL DEPARTMENTS' HH 'DEPARTMENT'   
1612 SALARY SZ=1 HH 'TOTAL SALARIES'
```

Coding Example Result

The following example displays the output of the above coding example, for generating the total of all the salaries:

```
DEPARTMENT TOTAL SALARIES
ALL DEPARTMENTS 2,522,500.00
```

Suppress All Totals

You can suppress all totals by specifying a details-only report on the OUT parameter, using the following information:

• The 2-digit report number

• The word OUT

• The letter D, after skipping one or more spaces

Example

The coding example below uses the following parameters, to produce a listing of employee salaries and suppress totals:

• 01
  The report number.
• OUT
  Specifies an OUTPUT parameter.

• D
  Specifies printing of only detail lines.

Coding Example

The following coding example displays the code for producing a listing of employee salaries and suppress totals:

```plaintext
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE   5  25      'EMPLOYEE NAME'
REC SALARY    160  5  3 DP=2  'ANNUAL SALARY' 01OUT D13 EMPLOYEE SALARY LISTING
151*010 EMPLOYEE SZ=20 HR
151*020 SALARY SZ=11 F$ HR
```

Coding Example Results

The following example displays the output of the above code, to produce a listing of employee salaries and suppress totals:

<table>
<thead>
<tr>
<th>EMPLOYEE NAME</th>
<th>ANNUAL SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERRY JANENS</td>
<td>$38,000.00</td>
</tr>
<tr>
<td>JOE NGUYA</td>
<td>$31,000.00</td>
</tr>
<tr>
<td>MARK TIME</td>
<td>$33,000.00</td>
</tr>
<tr>
<td>ROGER WILCO</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>ALBERT BREEZE</td>
<td>$38,000.00</td>
</tr>
<tr>
<td>CAROLYN CROW</td>
<td>$37,500.00</td>
</tr>
<tr>
<td>BURT LANCHESTER</td>
<td>$54,500.00</td>
</tr>
<tr>
<td>RENE MAKER</td>
<td>$85,000.00</td>
</tr>
<tr>
<td>DUDY B LOOMER</td>
<td>$15,000.00</td>
</tr>
<tr>
<td>EDWARD HUTTON</td>
<td>$44,000.00</td>
</tr>
</tbody>
</table>

Generate Subtotals

You can retrieve subtotals on a sorted field by creating a pause with a control break, using the following information:

- The 2-digit report number
- The word SORT
- The name of the field to be sorted
- The sort order (ascending or descending)
- The control break codes to manage what to do after the subtotal is printed
Example

The coding example below uses the following parameters to create a subtotal by department, insert two lines after each subtotal, and list the salaries in descending order. Duplicate salaries sort in alphabetical order by employee last name.

- **01**
  The report number.

- **SORT**
  Specifies the name of the field to sort.

- **DEPARTMENT**
  Specifies a sort on department.
  - A dash ( - ) specifies a control break, which generates the subtotal and spacing for each department.

- **SALARY**
  Specifies a sort on the salary within each department.

- **D**
  Specifies to sort the salary in descending order, from largest to smallest. If the descending sort order is not specified, by default the sort is in ascending order.

- **LAST-NAME**
  Specifies a sort on employees within the same salary and department.

In addition to the above parameters that define the sorted field, the following control break codes are used to manage what to do after the subtotal is printed:

- **1**
  Start a new page.

- **0**
  Print one blank line.

- **Dash symbol ( - )**
  Print two blank lines.

- **Plus sign ( + )**
  Begin the next sorted field section immediately after the subtotal line.

Repeat specifying the field to sort, the sort order, and the control break code for each sorted field you define. To easily distinguish a subtotal, configure at least one line between sorted fields.

**Coding Example**

The following example displays the code used to create a subtotal by department, insert two lines after each subtotal, and list the salaries in descending order. Duplicate salaries sort in alphabetical order by employee last name.
### Coding Example Result

The following example displays the output of the above code that creates a subtotal by department:

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>DEPARTMENT NAME</th>
<th>EMPLOYEE NAME</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>JENSON</td>
<td>MGR ACCTNG</td>
</tr>
<tr>
<td>/PAYROLL</td>
<td>82,000.00</td>
<td>RUPERT</td>
<td></td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>KIMBALL</td>
<td>MARIANNE</td>
<td>ACCOUNTANT</td>
</tr>
<tr>
<td>ANALYST</td>
<td>44,000.00</td>
<td>HUTTON</td>
<td>FINANCIAL</td>
</tr>
<tr>
<td>CLERK</td>
<td>15,000.00</td>
<td>BLOOMER</td>
<td>PAYROLL</td>
</tr>
<tr>
<td></td>
<td>14,500.00</td>
<td>KING</td>
<td>AR</td>
</tr>
<tr>
<td></td>
<td>14,000.00</td>
<td>NICEMAN</td>
<td>AP</td>
</tr>
<tr>
<td></td>
<td>214,500.00</td>
<td>MOON</td>
<td>MGR BLUE SKIES</td>
</tr>
<tr>
<td>BLUE SKIES</td>
<td>72,800.00</td>
<td>DANIEL</td>
<td></td>
</tr>
<tr>
<td>CARETAKER</td>
<td>52,750.00</td>
<td>CLOUD</td>
<td>CUMULUS</td>
</tr>
<tr>
<td>BLUE SKIES</td>
<td>33,500.00</td>
<td>DONOVAN</td>
<td>SUNSHINE</td>
</tr>
<tr>
<td>SUPERVISOR</td>
<td>158,250.00</td>
<td>ALAN</td>
<td></td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>8,000.00</td>
<td>WILCO</td>
<td>MGR</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>45,000.00</td>
<td>PHINEAS</td>
<td>KEEPER OF</td>
</tr>
<tr>
<td>BALLOON</td>
<td>38,800.00</td>
<td>CLOTH</td>
<td>HUMIDITY CONTROL</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>33,000.00</td>
<td>TIME</td>
<td>WINTERIZER</td>
</tr>
<tr>
<td>31,000.00</td>
<td>31,000.00</td>
<td>KASPAR</td>
<td>JOE</td>
</tr>
<tr>
<td>227,000.00</td>
<td>227,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2,522,500.00</td>
<td>2,522,500.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Select Subtotal and Total Line Placement

You can define the placement of totals and labels in specific report column, using the following information:
The 2-digit report number

The type 6 parameter

The field to appear on the first total print line

The placement of the subtotal

The field to be totaled.

⚠️ **Note:** When the same numeric data item appears on both type 5 and type 6 parameters, both parameters must contain identical information, such as size (SZ=) and decimal point placement (DP=). Heading information can appear on either parameter, but not both.

### Example

The coding example below uses the following parameters to print salary subtotal and grand total fields in a separate column:

- **01**
  - The report number.

- **6**
  - The parameter type.

- **1**
  - Specifies this field to appear on the first totals print line.

- ***055**
  - Specifies automatic placement of the subtotal to the right of the individually listed fields to total.

- **SALARY**
  - Specifies the field to be totaled.

### Coding Example

The following example displays the code for printing the salary subtotal and grand total fields in a separate column:

```plaintext
col. 2

IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC FIRST-NAME 5 01
REC LAST-NAME 15 15
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
REC TITLE 171 20
013 EMPLOYEE SALARY LISTING BY DEPARTMENT
0150RT DEPARTMENT SALARY D LAST-NAME
0151*010 DEPARTMENT HH 'DEPARTMENT'
0151*020 LAST-NAME SZ=1 HH ' EMPLOYEE NAME'
```
Coding Example Result

The following example displays the output of the above code to print subtotal and grand total fields in a separate column:

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>JOB TITLE</th>
<th>ANNUAL SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>ACCOUNTING AND PAYROLL</td>
<td>JENSON</td>
<td>MGR</td>
<td>82,000.00</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>KIMBALL</td>
<td>ACCOUNTANT</td>
<td>45,000.00</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>HUTTON</td>
<td>FINANCIAL ANALYST</td>
<td>44,000.00</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>BLOOMER</td>
<td>PAYROLL CLERK</td>
<td>15,000.00</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>KING</td>
<td>AR CLERK</td>
<td>14,500.00</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>NICEMAN</td>
<td>AP CLERK</td>
<td>14,000.00</td>
</tr>
<tr>
<td>08</td>
<td>BLUE SKIES</td>
<td>MOON</td>
<td>MGR BLUE SKIES</td>
<td>72,000.00</td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>CLOUD</td>
<td>CUMULUS CARETAKER</td>
<td>52,750.00</td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>DONOVAN</td>
<td>SUNSHINE SUPERVISOR</td>
<td>33,500.00</td>
</tr>
<tr>
<td>08</td>
<td>THERMOREGULATION</td>
<td>WILCO</td>
<td>MGR THERMOREGULATION</td>
<td>8,000.00</td>
</tr>
<tr>
<td></td>
<td>THERMOREGULATION</td>
<td>FINN</td>
<td>KEEPER OF BALLOONS</td>
<td>45,000.00</td>
</tr>
<tr>
<td></td>
<td>THERMOREGULATION</td>
<td>CLOTH</td>
<td>HUMIDITY CONTROL CLK</td>
<td>38,000.00</td>
</tr>
<tr>
<td></td>
<td>THERMOREGULATION</td>
<td>TIME</td>
<td>WINTERIZER</td>
<td>33,000.00</td>
</tr>
<tr>
<td></td>
<td>THERMOREGULATION</td>
<td>KASPAR</td>
<td>WINTERIZER</td>
<td>31,000.00</td>
</tr>
<tr>
<td>08</td>
<td>THERMOREGULATION</td>
<td>JOE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Selectively Print Output Lines

When you work with multiple levels of output (one or more control breaks), you can control when the output of each level is printed, using the following information:

- The 2-digit report number
- The word SORT
- A type 6 parameter, to print the line
- A type 8 parameter, to act on the data
- A **TAKE** instruction on a Type 8 parameter, to specify which type 6 line to print and stop further processing for that control break
- A **DROP** instruction on a type 8 parameter, to stop processing for that control break.
- A 3-digit sequence number

 Optionally, you can print output lines with a type 6 parameter alphanumeric work field. See "Print Output Lines Using a Work Field," below.
Print Output Lines Using Type 8 Parameters

This section describes how to use type 8 parameters to print different salary totals.

Example

The coding example below uses the following type 8 parameters to print salary totals by employee job status, department, and grand total:

- Specifies the report number.
- **SORT**
  Precedes the name of the field to sort.
- **DEPARTMENT**
  Specifies the field to sort.
- **STATUS**
  Specifies to total salaries by employee job status
- **0**
  Specifies control breaks for subtotals.
- **61, 62, 63**
  Specifies groups of print lines used to label and print subtotal, total, and grand totals
- **8**
  Specifies a type 8 parameter that is used when testing control break levels.
- **IF LEVL EQ 1 100**
  Specifies that when the value of STATUS changes, go to line 100 of the type 8 parameter code.
- **IF LEVL EQ 2 200**
  Specifies that when the value of DEPARTMENT changes, go to line 200 of the type 8 parameter code.
- **IF LEVL EQ 3 300**
  Specifies that when all the records are processed, go to line 300 of the type 8 parameter code.
- **TAKE 1**
  Specifies to print line 1 of the type 6 parameter code.
- **TAKE 2**
  Specifies to print line 2 of the type 6 parameter code.
- **TAKE 3**
  Specifies to print line 2 of the type 6 parameter code.

Coding Example

The following example displays the code for printing salary totals by employee job status, department, and grand total:
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC FIRST-NAME 5 10
REC LAST-NAME 15 15
REC STATUS 82 2
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
REC TITLE 171 20
013 EMPLOYEE SALARY LISTING
01SORT DEPARTMENT - STATUS 0 SALARY D
0151*010 DEPARTMENT HH 'DEPARTMENT'
0151*020 LAST-NAME SZ=10 HH 'EMPLOYEE NAME'
0151*025 FIRST-NAME
0151*030 TITLE HH 'TITLE'
0151*035 STATUS HH 'STATUS CODE'
0151*040 SALARY SZ=10 HH 'ANNUAL SALARY'
0161*0300 'STATUS CODE TOTAL'
0161*040 SALARY SZ=10
0162*0300 'DEPARTMENT TOTAL'
0162*040 SALARY SZ=10
018010 IF LEVL EQ 1 100
018020 IF LEVL EQ 2 200
018030 IF LEVL EQ 3 300
018100 TAKE 1
018200 TAKE 2
018300 TAKE 3

Code Example Result:

The following example displays the output of the above code, to print three different totals:

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE SALARY LISTING</th>
<th>PAYROLL</th>
<th>PAGE</th>
<th>EMPLOYEE NAME</th>
<th>TITLE</th>
<th>STATUS CODE</th>
<th>ANNUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>PEOPLE</td>
<td>ELEANOR</td>
<td>MGR</td>
<td>PERSONNEL</td>
<td>01</td>
<td>80,000.00</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>ORGRATZI</td>
<td>MADELINE</td>
<td>RECR</td>
<td>INTERVWR</td>
<td>01</td>
<td>39,000.00</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>FITZHUGH</td>
<td>TOM</td>
<td>CLERK</td>
<td>PERSONNEL</td>
<td>01</td>
<td>13,000.00</td>
<td></td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>JOHNSON</td>
<td>CYNTHIA</td>
<td>MGR</td>
<td>PERSONNEL</td>
<td>05</td>
<td>13,500.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>BANK</td>
<td>Donte</td>
<td>MGR</td>
<td>PUBLIC RELATIONS</td>
<td>01</td>
<td>80,000.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>PENMAN</td>
<td>LAURA</td>
<td>WR</td>
<td>PR WRITER</td>
<td>01</td>
<td>39,000.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>JACKSON</td>
<td>JOCK</td>
<td>CONS</td>
<td>SPORTS CONSULTANT</td>
<td>01</td>
<td>34,000.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>ANGELO</td>
<td>MICHAEL</td>
<td>ILLU</td>
<td>ILLUSTRATOR</td>
<td>01</td>
<td>18,000.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>MCDUGALL</td>
<td>CAROL</td>
<td>PASTE</td>
<td>PASTE-UP ARTIST</td>
<td>01</td>
<td>18,000.00</td>
<td></td>
</tr>
<tr>
<td>PUBLIC RELATIONS</td>
<td>ZEDI</td>
<td>BETSY</td>
<td></td>
<td>PR WRITER</td>
<td>05</td>
<td>37,000.00</td>
<td></td>
</tr>
</tbody>
</table>

Print Output Lines Using a Work Field

In addition to using the type 8 parameters to print different total lines, described above, you can set up and place the total line labels using type 6 parameters.

Example

The coding example below adds the following type 6 parameters to identify total line labels, for printing salary totals by employee job status, department, and grand total:
• 0
Identifies the work field

• TOTAL-MESSAGE
Creates a space holder for the labels of the total line fields.
You use a single quotation mark at the beginning and end of the field. Create enough space to accommodate the maximum number of letters needed for the total line label. For example, if the longest total line label is STATUS CODE TOTAL, you create a space with 17 characters between the single quotation marks.

• MOVE
Places the total line label in the work field.

Coding Example

The follow example displays the type 8 and type 6 parameter codes for printing salary totals by employee job status, department, and grand total. (Only the type 6 parameters are in bold print.):

col. 2
RECO  EMPLOYEE 5 25
REC  FIRST-NAME 5 10
REC  LAST-NAME 15 15
REC  STATUS 82 2
REC  DEPARTMENT 115 25
REC  SALARY 160 5 3 DP=2
REC  TITLE 171 20
013  EMPLOYEE  SALARY  LISTING
015  SORH  DEPARTMENT 0 STATUS 0 SALARY D,
010  TOTAL-MESSAGB
0151*010  DEPARTMENT  HH 'DEPARTMENT' HH 'EMPLOYEE NAME'
0151*020  LAST-NAME  SZ=10 HH '
0151*030  FIRST-NAME
0151*040  TITLE  HH 'TITLE'
0151*050  STATUS  HH 'STATUS' 'CODE'
0151*060  SALARY  SZ=10 HH 'ANNUAL SALARY'
0161*0400  TOTAL-MESSAGE
0161*060  SALARY  SZ=10
018010  IF  LEVL EQ 1 100
018020  IF  LEVL EQ 2 200
018030  IF  LEVL EQ 3 300
018100  MOVE  'STATUS CODE TOTAL'  TO  TOTAL-MESSAGE
018110  TAKE
018200  MOVE  'DEPARTMENT TOTAL'  TO  TOTAL-MESSAGE
018210  TAKE
018300  MOVE  'GRAND TOTAL'  TO  TOTAL-MESSAGE
018310  TAKE

Coding Example Result

The following example displays the output of the above code, to print salary totals by employee job status, department, and grand total:

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>EMPLOYEE SALARY LISTING</th>
<th>m/d/y</th>
<th>PAGE</th>
<th>DEPARTMENT</th>
<th>EMPLOYEE NAME</th>
<th>TITLE</th>
<th>STATUS CODE</th>
<th>ANNUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td></td>
<td></td>
<td>1</td>
<td>PERSONNEL</td>
<td>PEOPLES</td>
<td>ELEANOR MGR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td>PERSONNEL</td>
<td>ORGRATZI</td>
<td>MADELINE RECRUITER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/INTERVNR</td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>80,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/INTERVNR</td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>39,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Obtain Sort Field Values

You can obtain the value of a numeric sort key when control breaks occur and when the last detail line for a control break is processed, if the following conditions are true:

- The field appears on a SORT parameter
- The field does not appear on a type 5 edit line.

Example

The code example below uses the following parameters to create a report that lists hourly employees who make less than $19,000 annually, sorts the salaries in ascending order, and prints the highest hourly wage:

- **0 AMOUNT**
  Defines a numeric work field to hold the hourly wage computation.

- **SORTAMOUNT**
  Specifies sorting of AMOUNT in ascending order, for the last amount to represent the highest hourly wage.

- **62**
  Specifies that the second type 6 line labels and prints the last value for AMOUNT.
Code Example

The following example displays the code for creating a report that lists hourly employees who make less than $19,000 annually, sorts the salaries in ascending order, and prints the highest hourly wage:

```
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC LAST-NAME 15 15
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
SEL SALARY LE 19000
01OUT 80
01SORT AMOUNT
013 UNSALARIED EMPLOYEE COMPENSATION
010 AMOUNT DP=2
0151*010 EMPLOYEE HH 'EMPLOYEE' 'NAME'
0151*020 SALARY SZ=9 F$ HH 'ANNUAL SALARY'
0161*020 SALARY SZ=9 F$
0162*0150'HIGHEST HOURLY WAGE'
0162*030 AMOUNT SZ=9 F$
017010 COMPUTE (SALARY / 52) / 40 AMOUNT
```

Code Example Result

The following example displays the output of the code example above, to create a report that lists hourly employees who make less than $19,000 annually, sorts the salaries in ascending order, and prints the highest hourly wage:

```
REPORT NO. 01 UNSALARIED EMPLOYEE COMPENSATION mm/dd/yy PAGE 1

EMPLOYEE NAME ANNUAL SALARY
NANCY TERNER $13,000.00
TOM FITZHUGH $13,000.00
CYNTHIA JOHNSON $13,500.00
BRIAN NICEMAN $14,000.00
ROBIN GARDNER $14,000.00
SANDY KRAINER $14,080.00
DORIS KING $14,500.00
GEORGE FONRAD $14,750.00
JUNE BLOOPER $15,000.00
MICHAEL ANGELO $18,000.00
CAROL MCDougALL $18,000.00
HERBERT LIPSICH $18,500.00
$180,250.00
HIGHEST HOURLY WAGE $8.89
```

Obtain Sorted Work Field Values

You can obtain the value of an alphanumeric work field when a control break occurs or when the last detail line for a control break is processed.

Use the following parameters to enter the work field value:

- **SORT**
  Specifies to sort by the work field.

- **MESSAGE**
  Specifies the alphanumeric work field.
Example

The coding example below uses the following parameters to sort on the work field to print a report that lists employees by executive, salaried, and hourly categories.

- **SORT MESSAGE - SALARY**
  Specifies to sort on the salary work field.

- **AMOUNT**
  The work field that holds the hourly wage computation.

- **0152*030**
  Specifies a second print line for the hourly wage.

  - **+**
    Specifies to print the amount on the first line (overprint).

  - **AMOUNT**
    Specifies printing the hourly wages.

- **SALARY**
  Specifies printing the total salary at a control break.

- **IF SALARY LE 19 500**
  Monitors for salaries less than or equal to $19,900. If true, line 500 is executed.

- **IF SALARY GT 19 50000 600**
  Monitors for salaries greater than $50,000. If true, line 600 is executed.

- **MOVE 'SALARIED' TO MESSAGE**
  Places the salary wage in the work field.

- **TAKE 1**
  Specifies printing the first detail line.

- **MOVE 'HOURLY' TO MESSAGE**
  Places the hourly wage in the work field.

- **COMPUTE (SALARY / 52) / 40 AMOUNT**
  Calculates the hourly salary.

- **TAKE (1 2)**
  Specifies extracting the first and second detail lines.

- **MOVE 'EXECUTIVE' TO MESSAGE**
  Places the executive job status in the work field, if the salary is greater than $50,000.

- **TAKE 1**
  Specifies printing the first detail line.
Coding Example

The following example displays the code for sorting on the work field to print a report that lists employees by executive, salaried, and hourly categories:

```
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC LAST-NAME 15 10
REC DEPARTMENT 115 25
REC SALARY 160 5 3 DP=2
01SORT MESSAGE - SALARY
013 EMPLOYEE COMPENSATION STATUS
010 MESSAGE ' ',
010 AMOUNT DP=2
0151*010 EMPLOYEE HH 'EMPLOYEE NAME'
0151*020 MESSAGE HH 'STATUS'
0151*030 AMOUNT SZ=9 F$ HH 'HOURLY RATE'
0151*040 SALARY SZ=9 F$ HH 'ANNUAL SALARY'
0161*0400 SALARY SZ=9 F$
017010 IF SALARY LE 19000 500
017020 IF SALARY GT 50000 600
017 MOVE 'SALARIED ' TO MESSAGE
017 TAKE 1
017500 MOVE 'HOURLY ' TO MESSAGE
017 COMPUTE (SALARY / 52) / 40 AMOUNT
017 TAKE (1 2)
017600 MOVE 'EXECUTIVE' TO MESSAGE
017 TAKE 1
```

Coding Example Result

The following example displays the output of the above code for sorting on the work field to print a report that lists employees by executive, salaried, and hourly categories:

```
<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE COMPENSATION STATUS</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETH</td>
<td>CLOUD</td>
<td>EXECUTIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BURT</td>
<td>LANCHESTER</td>
<td>EXECUTIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROBBY</td>
<td>WILDER</td>
<td>EXECUTIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THEMIS</td>
<td>PAPAZEUS</td>
<td>EXECUTIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HENRIETTA</td>
<td>HENDON</td>
<td>EXECUTIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEORGE</td>
<td>FORNAD</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICHAEL</td>
<td>ANGELD</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAROL</td>
<td>MCDUGALL</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HERBERT</td>
<td>LIPSICH</td>
<td>HOURLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERENCE</td>
<td>KLWELLEN</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDWARD</td>
<td>HUTTON</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIANNE</td>
<td>KIMBALL</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHINEAS</td>
<td>FINN</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HARRY</td>
<td>ARM</td>
<td>SALARIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RICHARD</td>
<td>WAGNER</td>
<td>SALARIED</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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

ANNUAL SALARY

| BETH          | $52,750.00        |
| BURT          | $54,500.00        |
| ROBBY         | $90,000.00        |
| THEMIS        | $100,000.00       |
| HENRIETTA     | $240,000.00       |
| DORIS         | $1,291,250.00     |
| GEORGE        | $14,750.00        |
| JUNE          | $15,800.00        |
| MICHAEL       | $18,000.00        |
| CAROL         | $18,000.00        |
| HERBERT       | $18,500.00        |
| TERENCE       | $180,250.00       |
| EDWARD        | $44,000.00        |
| MARIANNE      | $45,000.00        |
| PHINEAS       | $45,000.00        |
| HARRY         | $46,000.00        |
| RICHARD       | $47,000.00        |

|               | $1,051,000.00     |
|               | $2,522,500.00     |
Enhance the Basic Report

CA Culprit provides optional parameters and codes you can use to make a report easier to read. In this chapter, some the CA Culprit most commonly used enhancement and numeric formatting options are introduced.

- Add a Title to a Report (see page 73)
- Add Column Headings (see page 74)
- Modify Column Widths (see page 78)
- Format Numeric Data (see page 79)
- Adjust Line Length and Page Depth (see page 81)

Add a Title to a Report

You can supply a title for your report. CA Culprit automatically prints this title at the top of each page, along with the report number, current date, and page number.

Add a Title

To add a title, code a type 3 parameter in the following order, from left to right:

1. A 2-digit report number
2. The parameter type (3)
3. A title (not to exceed 50 characters, including spaces)

Example: Add a Title to a Report

Objective

This example adds a title EMPLOYEE SALARY LISTING to the report.

Parameters

- 013 EMPLOYEE SALARY LISTING
  - 01 is the report number.
  - 3 specifies a type 3 parameter.
  - EMPLOYEE SALARY LISTING is the title for the printed report.

Complete Code

col. 2

IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC SALARY 160 5 3 DP=2013 EMPLOYEE SALARY LISTING.
0151*010 EMPLOYEE
0151*020 SALARY
Add Column Headings

You can direct CA Culprit to obtain column headings from:

- Field names supplied on REC parameters
- Text added to REC parameter lines
- Text added to type 5 parameter lines
- Text added to type 4 parameter lines

How to Do It

You can use type 5 or type 4 parameters to create heading lines.

Method 1:

Use any of the following codes on type 5 parameters:

- **HF**, which directs CA Culprit to the associated field name:

  ```
  REC EMP-NAME 1 25
  0151*020 EMP-NAME HF
  ```

  or

  ```
  010 EMP-NAME0151*020 EMP-NAME HF
  ```

  Column heading is EMP-NAME.

- **HR**, which directs CA Culprit to heading text that is enclosed in single quotation marks on the record description (REC) line:

  ```
  REC EMP-NAME 1 25 'EMPLOYEE'
  ```

  Column heading is EMPLOYEE.

- **HH**, which directs CA Culprit to the text enclosed in single quotation marks following the HH option on the type 5 line:
A multiple-line heading is generated by enclosing the text for each line within a separate set of quotation marks:

```
0151*020 EMP-NAME HH 'EMPLOYEE' 'NAME'
```

or

```
REC EMP-NAME 'EMPLOYEE' 'NAME' 0151*020 EMP-NAME HR
```

Column heading is: EMPLOYEE NAME

**Method 2:**

Use type 4 parameters to create the headings. The type 4 parameter, typically used for special-purpose heading lines, is coded in the following order, from left to right:

1. A 2-digit report number
2. The parameter type (4)
3. A numeral to identify the level of the line
4. A column placement indicator consisting of either an asterisk (*) followed by a 3-digit number for relative placement or a 4-digit number for specific placement
5. A spacing indicator, such as a blank for single spacing, a zero (0) for double spacing, or a dash (-) for triple spacing
6. The text of the heading enclosed in single quotation marks or a field name

**Example 1: Format Multiple Line Column Headings**

**Objective**

This report prints a 2 line heading for the employee column and a 3 line heading for the salary column.

**Parameters**

- **REC EMPLOYEE 5 25 'EMPLOYEE' 'NAME'**
  - 'EMPLOYEE' is the first line of the EMPLOYEE column heading.
  - 'NAME' is the second line.

- **0151*020 SALARY HH 'ANNUAL' 'SALARY' 'AMOUNT'**
  - 'ANNUAL' is the first line of the salary column heading.
  - 'SALARY' is the second line.
  - 'AMOUNT' is the third line.

**Complete Code**

```
col. 2
IN 200 F 400 PS(TAPE)
```
Example 2: Add Subtitles

Objective

This example adds heading lines that function as subtitles, not column headings.

Parameters

- **01410054 'FROM mm/dd/yy TO mm/dd/yy'**
  01 is the report number.
  4 specifies a type 4 parameter.
  1 specifies that 'FROM ...' is printed on the first row of the subtitle.
  0054 indicates that this row of text begins exactly 54 spaces from the left margin. *Your parameter would contain dates in mm/dd/yy format.

- **0142*001 'ALL DEPARTMENTS'**
  014 specifies the report number and the parameter type, respectively.
  2 specifies that 'ALL DEPARTMENTS' is printed on the second row of the subtitle.
  *001 specifies that this row of text is left justified.

- **0143*010-'''**
  014 specifies the report number and the parameter type, respectively.
  3 specifies the third line of the subtitle.
  *010 specifies a left-hand column position.
  - tells CA Culprit to skip two lines before printing the next line.
  '' prints a blank space, which in this case creates a blank line.

Complete Code

```
col. 2
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25
REC SALARY 160 5 3 DF=2
013 EMPLOYEE SALARY LISTING 0151*010 EMPLOYEE
0151*020 SALARY
01410054 'FROM mm/dd/yy TO mm/dd/yy'
0142*001 'ALL DEPARTMENTS'
0143*010-''
0151*000 PAGE 1
```
### Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE SALARY LISTING</th>
<th>mm/dd/yy PAGE 1 FROM mm/dd/yy TO mm/dd/yy ALL DEPARTMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUNE BLOOMER</td>
<td>15,000.00</td>
<td></td>
</tr>
<tr>
<td>EDWARD HUTTON</td>
<td>44,000.00</td>
<td></td>
</tr>
<tr>
<td>RUPERT JENSON</td>
<td>82,000.00</td>
<td></td>
</tr>
<tr>
<td>MARIANNE KIMBALL</td>
<td>45,000.00</td>
<td></td>
</tr>
<tr>
<td>DORIS KING</td>
<td>14,500.00</td>
<td></td>
</tr>
<tr>
<td>BRIAN NIEFMAN</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>HERBERT CRANE</td>
<td>75,000.00</td>
<td></td>
</tr>
<tr>
<td>JANE FERNDALE</td>
<td>22,500.00</td>
<td></td>
</tr>
<tr>
<td>GEORGE FONRAD</td>
<td>14,750.00</td>
<td></td>
</tr>
<tr>
<td>ROBIN GARDNER</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>DOUGLAS KAHLALLY</td>
<td>20,000.00</td>
<td></td>
</tr>
<tr>
<td>TERENCE KLEWELLEN</td>
<td>43,000.00</td>
<td></td>
</tr>
<tr>
<td>SANDY KRAMER</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>HERBERT LIPSICH</td>
<td>18,500.00</td>
<td></td>
</tr>
<tr>
<td>NANCY TERNER</td>
<td>13,000.00</td>
<td></td>
</tr>
<tr>
<td>BETH CLOUD</td>
<td>52,750.00</td>
<td></td>
</tr>
<tr>
<td>ALAN DONOVAN</td>
<td>33,500.00</td>
<td></td>
</tr>
<tr>
<td>DANIEL MOON</td>
<td>72,800.00</td>
<td></td>
</tr>
<tr>
<td>ROY ANDALE</td>
<td>33,500.00</td>
<td></td>
</tr>
<tr>
<td>HARRY ARM</td>
<td>46,000.00</td>
<td></td>
</tr>
<tr>
<td>C. BREEZE</td>
<td>38,000.00</td>
<td></td>
</tr>
<tr>
<td>CAROLYN CRON</td>
<td>37,500.00</td>
<td></td>
</tr>
<tr>
<td>BURT LANCHESTER</td>
<td>54,500.00</td>
<td></td>
</tr>
<tr>
<td>RENE MAKER</td>
<td>91,000.00</td>
<td></td>
</tr>
<tr>
<td>RICHARD MARYON</td>
<td>36,000.00</td>
<td></td>
</tr>
<tr>
<td>RICHARD WAGNER</td>
<td>47,000.00</td>
<td></td>
</tr>
<tr>
<td>TOM FITZHUGH</td>
<td>13,000.00</td>
<td></td>
</tr>
<tr>
<td>CYNTHIA JOHNSON</td>
<td>13,500.00</td>
<td></td>
</tr>
<tr>
<td>MADELINE ORGRATZI</td>
<td>39,000.00</td>
<td></td>
</tr>
<tr>
<td>ELEANOR PEOPLES</td>
<td>89,000.00</td>
<td></td>
</tr>
<tr>
<td>MICHAEL ANGELD</td>
<td>18,000.00</td>
<td></td>
</tr>
<tr>
<td>MONTE BANK</td>
<td>80,000.00</td>
<td></td>
</tr>
<tr>
<td>CHARLES BOWER</td>
<td>38,500.00</td>
<td></td>
</tr>
<tr>
<td>JOCK JACKSON</td>
<td>34,000.00</td>
<td></td>
</tr>
<tr>
<td>CAROL MCDougALL</td>
<td>18,000.00</td>
<td></td>
</tr>
<tr>
<td>LAURA PENMAN</td>
<td>39,000.00</td>
<td></td>
</tr>
<tr>
<td>BETSY ZEID</td>
<td>37,000.00</td>
<td></td>
</tr>
<tr>
<td>TERRY CLOTH</td>
<td>38,000.00</td>
<td></td>
</tr>
<tr>
<td>PHINEAS FINN</td>
<td>45,000.00</td>
<td></td>
</tr>
<tr>
<td>JOE KASPAR</td>
<td>31,000.00</td>
<td></td>
</tr>
<tr>
<td>MARK TIME</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>ROGER WILCO</td>
<td>80,000.00</td>
<td></td>
</tr>
<tr>
<td>JANE DOUGH</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>JAMES GALLWAY</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>JENNIFER GARIFIELD</td>
<td>65,000.00</td>
<td></td>
</tr>
<tr>
<td>PERCY GRANGER</td>
<td>34,500.00</td>
<td></td>
</tr>
<tr>
<td>VLADIMIR HEARON</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>JAMES JACOB</td>
<td>55,000.00</td>
<td></td>
</tr>
<tr>
<td>JULIE JENSEN</td>
<td>37,000.00</td>
<td></td>
</tr>
<tr>
<td>LARRY LITERATA</td>
<td>37,500.00</td>
<td></td>
</tr>
<tr>
<td>KATHERINE O’HEARN</td>
<td>42,500.00</td>
<td></td>
</tr>
<tr>
<td>RALPH TYRO</td>
<td>20,000.00</td>
<td></td>
</tr>
<tr>
<td>HENRIETTA HENDON</td>
<td>240,000.00</td>
<td></td>
</tr>
<tr>
<td>THEMIS PAPAZEUSS</td>
<td>180,000.00</td>
<td></td>
</tr>
<tr>
<td>JOHN RUPEE</td>
<td>80,000.00</td>
<td></td>
</tr>
<tr>
<td>ROBBY WILDER</td>
<td>90,000.00</td>
<td></td>
</tr>
</tbody>
</table>

2,522,500.00
Modify Column Widths

You can control the number of characters or digits printed in a column. To place a specified number of characters or digits in a column, code:

1. A type 5 line for the field that is to be printed.

2. A $SZ=$ expression indicating the number of characters or digits you want to print on the report. This expression follows the field name and precedes any column headings.

Objective

We are going to modify our report by:

- Printing the employee last name
- Printing the first initial of the employee's first name
- Decreasing the size of the SALARY column

Parameters

- REC FIRST-NAME 5 10
  FIRST-NAME 5 10 defines the first-name field separately from the last-name field to allow separate processing.

- REC LAST-NAME 15 15
  LAST-NAME 15 15 defines the last-name field separately from the first-name field to allow separate processing.

- 0251*010 LAST-NAME $SZ=$10 HH 'EMPLOYEE'
  $SZ=$10 sets the column width to ten characters.

- 0251*020 FIRST-NAME $SZ=$1
  $SZ=$1 sets the column width to accommodate one character, which will truncate all but the first letter of the employee's first name.

- 0251*030 SALARY $SZ=$9 HH 'SALARY'
  $SZ=$9 sets the column width for 9 numbers; without sizing, the column saves space for 13 digits plus commas and decimal points. When reducing the size, be sure to allow enough room for totals to print without truncation.

Complete Code

```
col. 2
IN 200 F 400 PS('TAPE')
REC FIRST-NAME 5 10 REC LAST-NAME 15 15 REC SALARY 160 5 3 DP=2
023 EMPLOYEE SALARY LISTING
0251*010 LAST-NAME $SZ=$10 HH 'EMPLOYEE'
0251*030 SALARY $SZ=$9 HH 'SALARY'
```

Result
### Format Numeric Data

You can format numeric data automatically by using format codes supplied by CA Culprit or by using your own patterns.

Use a format code on the type 5 line. Useful format codes for printed output are:

- **FN** -- Prints leading zeros and omits commas in numbers such as those used for employee identification.
- **FS** -- Inserts dashes in social security numbers.
- **FD** -- Inserts slashes in dates.
- **F$** -- Inserts dollar signs in monetary information.
- **FM** -- Formats fields according to a specified pattern. The numeric field is symbolically represented as follows:
  - 9s and Zs represent digits. The Z suppresses leading zeroes.
  - Periods (.) represent decimal points.
  - Special characters (for example, slashes or asterisks) are preceded by a printable digit.

### Example 1: Use Formatting Codes

**Objective**

This example uses codes for automatically formatting the employee identification, social security numbers, salaries, and employment dates.

**Parameters**

- **0251*020 ID FN HH 'ID'**
  - FN prints leading zeros and omits commas.
Example 2: Use Formatting Patterns

Objective

The next sample report uses user-supplied patterns to format employee identification numbers with asterisks (*) and consecutive data fields with dashes.

Parameters

- **0251*020 ID FM '9*999' HH 'ID'**
  
  **FM** specifies that a user-supplied pattern formats the data. **'9*999'** is the format pattern.
• 0251*040 DATES FM '99/99-99/99/99/99' HH 'EMPLOYMENT DATES'
  FM specifies that a user-supplied pattern formats the data.

Complete Code

col. 2

IN 200 F 400 PS(TMPE)
REC ID 1 4 2
REC EMPLOYEE 5 25
REC SOC-SEC 84 9 2
REC DATES 93 12 2
REC SALARY 160 7 3 DP=2
013 PERSONNEL LISTING REPORT
0151*010 EMPLOYEE SZ=20 HH 'EMPLOYEE'
0151*020 ID HH 'ID'
0151*030 SOC-SEC FS HH 'SOCIAL SECURITY' 'NUMBER'
0151*040 DATES HH 'EMPLOYMENT DATES'
0151*050 SALARY FS HH 'SALARY'

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>PERSONNEL LISTING REPORT</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>SOCIAL SECURITY</th>
<th>EMPLOYMENT DATES</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUNE BLOOMER</td>
<td>0*069</td>
<td>039-55-7818</td>
<td>80/05/05-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDWARD HUTTON</td>
<td>0*100</td>
<td>011-22-3333</td>
<td>77/09/87-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUPERT JENSON</td>
<td>0*011</td>
<td>022-34-7891</td>
<td>80/09/29-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIANNE KINBALL</td>
<td>0*067</td>
<td>022-77-8878</td>
<td>78/09/19-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORIS KING</td>
<td>0*186</td>
<td>067-84-5516</td>
<td>80/08/16-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRIAN NICEMAN</td>
<td>0*101</td>
<td>033-45-6110</td>
<td>80/05/05-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HERBERT CRANE</td>
<td>0*004</td>
<td>016-77-7451</td>
<td>77/05/14-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANE FENDALE</td>
<td>0*032</td>
<td>034-56-7891</td>
<td>79/09/89-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEORGE FONRAD</td>
<td>0*045</td>
<td>092-34-8763</td>
<td>80/04/14-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROBIN GARDNER</td>
<td>0*053</td>
<td>022-33-4444</td>
<td>81/06/15-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUGLAS KANALLY</td>
<td>0*049</td>
<td>029-66-1234</td>
<td>79/09/29-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERENCE KUWELLEN</td>
<td>0*016</td>
<td>010-28-1239</td>
<td>78/01/86-00/00/00</td>
<td>$.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adjust Line Length and Page Depth

You can specify line length and page depth by using OUTPUT (OUT) parameter options. When the options are not used, CA Culprit automatically prints 132 characters to a line and 55 lines to a page, not including title lines and headings.

OUTPUT parameter options that adjust report size are coded by:

- A numeral to specify the number of characters on a line
- LP= immediately followed by a numeral to specify the number of lines on a page

Example 1: Specify Output Parameters for a Report

Objective

We are going to print a report with a 60-character line length and a 19-line page size by using OUTPUT parameter options.
The report is shown printed without the OUTPUT parameter specifications and again with the specifications.

Parameters

- **01OUT 60 LP=19**
  - 01 is the report number.
  - OUT specifies the OUTPUT parameter.
  - 60 specifies the number of characters to be printed on a report line.
  - LP=19 specifies a page length of 19 lines.

Complete Code

```
IN 200 F 400 PS(TAPE)
REC EMPLOYEE 5 25 'EMPLOYEE NAME'
REC SALARY 160 5 3 DP=2
013 EMPLOYEE SALARY LIST
0151*010 EMPLOYEE HR
01OUT 60 LP=19
0151*020 SALARY HF
```

Result BEFORE

```
REPORT NO. 01 EMPLOYEE SALARY LIST mm/dd/yy PAGE 1
EMPLOYEE SALARY
MICHAEL ANGELO 18,000.00
DONTE BANK 80,000.00
ALBERT BREEZE 38,000.00
BETH M. CLOUD 52,750.00
ALAN DONOVAN 33,500.00
PERCY EINSTEIN 34,500.00
JANE FERNDALE 22,500.00
TOM FITZHUGH 13,000.00
ROBIN GARDNER 14,000.00
JENNIFER GARFIELD 65,000.00
VLADIMIR HEAROWITZ 33,000.00
EDWARD HUTTON 44,000.00
JOCK JACKSON 34,000.00
JAMES JACOBI 55,000.00
JULIE JANSSEN 37,000.00
RUPERT JENSON 82,000.00
MARYLOU JOHNSON 12.00
CYNTHIA JOHNSON 13,500.00
DOUGLAS KAHALLY 20,000.00
MICHAEL ANGELO 18,000.00
DONTE BANK 80,000.00
ALBERT BREEZE 38,000.00
BETH M. CLOUD 52,750.00
ALAN DONOVAN 33,500.00
PERCY EINSTEIN 34,500.00
JANE FERNDALE 22,500.00
TOM FITZHUGH 13,000.00
ROBIN GARDNER 14,000.00
JENNIFER GARFIELD 65,000.00
EDWARD HUTTON 44,000.00
JOCK JACKSON 34,000.00
JAMES JACOBI 55,000.00
JULIE JANSSEN 37,000.00
RUPERT JENSON 82,000.00
MARYLOU JOHNSON 12.00
CYNTHIA JOHNSON 13,500.00
DOUGLAS KAHALLY 20,000.00
```

Result AFTER

```
REPORT NO. 01 EMPLOYEE SALARY LIST mm/dd/yy PAGE 1
EMPLOYEE SALARY
MICHAEL ANGELO 18,000.00
DONTE BANK 80,000.00
ALBERT BREEZE 38,000.00
BETH M. CLOUD 52,750.00
ALAN DONOVAN 33,500.00
PERCY EINSTEIN 34,500.00
JANE FERNDALE 22,500.00
TOM FITZHUGH 13,000.00
ROBIN GARDNER 14,000.00
JENNIFER GARFIELD 65,000.00
EDWARD HUTTON 44,000.00
JOCK JACKSON 34,000.00
JAMES JACOBI 55,000.00
JULIE JANSSEN 37,000.00
RUPERT JENSON 82,000.00
MARYLOU JOHNSON 12.00
CYNTHIA JOHNSON 13,500.00
DOUGLAS KAHALLY 20,000.00
```
Generate a Basic Report from Standard Files

Locating the Data

The data for our report is stored in the personnel file of the Commonweather Corporation. The figure below describes part of the file. The complete description is in the appendix "The Personnel File Description".

Information about the data you will use to create your CA Culprit report is contained in a file description similar to the one above. The description is usually provided by your data-processing department.

<table>
<thead>
<tr>
<th>RECORD SIZE</th>
<th>BLOCK SIZE</th>
<th>RECORD FORMAT</th>
<th>FILE TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 bytes</td>
<td>400 bytes</td>
<td>FIXED (F)</td>
<td>SEQUENTIAL (PS)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Start Position</th>
<th>Length (bytes)</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE-NAME</td>
<td>5</td>
<td>25</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>FIRST-NAME</td>
<td>5</td>
<td>10</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>LAST-NAME</td>
<td>15</td>
<td>15</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>DEPT-NAME</td>
<td>115</td>
<td>45</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>SALARY-AMOUNT</td>
<td>160</td>
<td>7</td>
<td>Packed Decimal</td>
</tr>
<tr>
<td>TITLE</td>
<td>171</td>
<td>20</td>
<td>Alphanumeric</td>
</tr>
</tbody>
</table>

Information about the data you will use to create your CA Culprit report is contained in a file description similar to the one above. The description is usually provided by your data-processing department.

Coding the Report

A simple CA Culprit report is produced with three instructions (parameters):

- The INPUT (IN) parameter, which tells CA Culprit how the file containing the data is organized
- REC parameters, which tell CA Culprit what data to use
• **Type 5** parameters, which tell CA Culprit what and where to print the data

It is important to remember that one INPUT parameter with at least one REC parameter and one type 5 parameter are minimal requirements for any CA Culprit report produced from standard files.

Our first report lists the names of employees and their salaries from the data stored in the personnel file. We will create the report from the three required parameters.

**How to Define the Input**

To define the input data, code:

1. An INPUT (IN) parameter to describe the file
2. A REC parameter for each data field used in the report

**Sample Parameters**

- **IN 200 F 400 PS(TAPE)**  
  200 tells CA Culprit that each record in the personnel file contains 200 characters (bytes).  
  F indicates fixed-length records.  
  400 indicates that the file is blocked in 400 byte segments.  
  PS(TAPE) specifies a physical sequential file on a tape device.

- **REC EMPLOYEE-NAME 5 25**  
  EMPLOYEE-NAME 5 25 directs CA Culprit to use the data stored in positions 5 through 25 of each record.

- **REC SALARY-AMOUNT 160 7 3 DP=2**  
  SALARY-AMOUNT 160 7 directs CA Culprit to use the data from positions 160 through 166 for salary information.  
  3 tells CA Culprit that the salary data is stored as packed decimal.  
  DP=2 tells CA Culprit that SALARY-AMOUNT has two decimal places.

**How to Define the Output**

To define the report output, code a **type 5** parameter for each data item. The information on a type 5 parameter is coded in the following order, from left to right:

1. A 2-digit report number
2. The parameter type (5)
3. A 1-digit print-row identifier
4. The column placement, consisting of either an asterisk (*) followed by a 3-digit number for relative placement or a 4-digit number for specific placement.
5. A spacing indicator consisting of a blank for a single space, a zero (0) for a double space, or a dash (-) for a triple space
6. The name of the data item to appear in the column

Sample Parameters

- **0151*010 EMPLOYEE**
  - 01 is the assigned report number.
  - 5 specifies a type 5 parameter.
  - 1 specifies that the employee name appears on the first print row.
  - *010 specifies that CA Culprit determines the placement of the employee name on the page.
  - Low-number columns print on the left-hand side of the page; high-number columns print progressively to the right of the first column. The **blank** in column 10 specifies single spacing.
  - **EMPLOYEE** directs CA Culprit to print the employee's name in this column.

- **0151*020 SALARY**
  - 0151 specifies the report number, parameter type, and first print row.
  - *020 directs CA Culprit to print **SALARY** to the right of the employee name.
  - **SALARY** directs CA Culprit to print the salary amount in this column.

Generate Reports From Database Files

CA Culprit reports can be written from data stored in a database as well as from data stored in standard files.

You need to obtain the following information about your CA IDMS/DB database before you code your report:

- The **data dictionary name**, if you are not using the primary dictionary of your installation

- The name of the **subschema** that describes the database record types and fields that CA Culprit accesses to build the report

- A **data structure diagram**, similar to that shown in the following figure
Introduction

List Data

You can list related occurrences of records in the database.

How to Do It

To access the database, code:

1. A DATABASE parameter with the DICTNAME= option if you are using an alternate dictionary

2. An INPUT parameter with the following information:
   a. DB to specify that the report is accessing a database
   b. SS= with the name of the subschema you are using
   c. The path name (PATH-ID)
   d. The name of each record as it is encountered on the path

3. Type 5 parameters with appropriate field names (as listed in the subschema)
Note: CA Culprit automatically generates REC parameters from the definitions stored in the data dictionary. REC parameters do not have to be coded unless you want field names or data types that are different from those given in the subschema.

Example: List Employee Data

Objective

This report lists all employee name, ID, start date, and job title information stored in the Personnel database. Any employee having more than one job title in the course of employment is listed each time the job title changes. Numeric fields are not edited.

Parameters

- DATABASE DICTNAME=DOCUDICT
  DATABASE specifies the database parameter. If used, it must always be the first parameter.
  DICTNAME=DOCUDICT specifies DOCUDICT as the dictionary used for the report.

- IN DB SS=EMPSS01
  DB specifies a database as the data source.
  SS=EMPSS01 specifies EMPSS01 as the subschema used.

- PATHAA EMPLOYEE EMPOSITION JOB
  PATH specifies the PATH parameter.
  AA names the path.
  EMPLOYEE EMPOSITION JOB specifies the related database records used as a path through the database.

The field names are the same as those used in the database.

Complete Code

<table>
<thead>
<tr>
<th>col. 2</th>
<th>DATABASE DICTNAME=DOCUDICTIN DB SS=EMPSS01PATHAA EMPLOYEE EMPOSITION JOB010UT 80 D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0151*010 EMP-NAME-0415</td>
<td>HH 'EMPLOYEE' 'NAME'</td>
</tr>
<tr>
<td>0151*020 EMP-ID-0415</td>
<td>HH 'EMPLOYEE' 'ID'</td>
</tr>
<tr>
<td>0151*030 START-DATE-0420</td>
<td>HH 'START' 'DATE'</td>
</tr>
<tr>
<td>0151*040 TITLE-0440</td>
<td>HH 'JOB' 'TITLE'</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE LISTING</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE NAME</td>
<td>EMPLOYEE ID</td>
<td>START DATE</td>
<td>JOB TITLE</td>
</tr>
<tr>
<td>KATHERINE O'HEARN</td>
<td>23</td>
<td>790505</td>
<td>PROGRAMMER/ANALYST</td>
</tr>
<tr>
<td>KATHERINE O'HEARN</td>
<td>23</td>
<td>780504</td>
<td>PROGRAMMER TRAINEE</td>
</tr>
<tr>
<td>ROBBY WILDER</td>
<td>472</td>
<td>790716</td>
<td>DIR CORP CONFUSION</td>
</tr>
<tr>
<td>BURT LANCHESTER</td>
<td>381</td>
<td>808203</td>
<td>RAINMAKER</td>
</tr>
</tbody>
</table>
Use Logical Records

Logical records allow reports to be created from data stored in the database with no concern for where the information is located. All you need to know is the name of the logical record and the fields contained in the record.

How to Do It

When using logical records, code:

1. A DATABASE parameter with the DICTNAME= option if an alternate dictionary is being used
2. An INPUT DB parameter with the SS= keyword and the subschema name
3. Type 5 parameters with appropriate field names

Example: List Employee Data from a Logical Record

Objective

This report lists employee information from data contained in the logical record EMP-JOB-LR.

Parameters

- REC START-DATE 79 6 2
  REC START-DATE redefines the date field as numeric to allow automatic formatting.

- PATHAA EMP-JOB-LR
  EMP-JOB-LR is the logical record containing data from the database.

- 01OUT D
  D suppresses totaling of the EMP-ID-0415 field.

Complete Code

```
REC START-DATE 79 6 2
PATHAA EMP-JOB-LR
01OUT D
```

Result

```
REPORT NO. 01 EMPLOYEE LISTING mm/dd/yy PAGE 1
```

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>ID</th>
<th>Phone</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burt</td>
<td>Lanchester</td>
<td>301</td>
<td>770203</td>
<td>Rainmaker</td>
</tr>
<tr>
<td>Vlad</td>
<td>Hearowitz</td>
<td>27</td>
<td>810909</td>
<td>Programmer/Analyst</td>
</tr>
<tr>
<td>Themis</td>
<td>Papazeus</td>
<td>471</td>
<td>820101</td>
<td>Dir Weather</td>
</tr>
<tr>
<td>Monte</td>
<td>Bank</td>
<td>7</td>
<td>780907</td>
<td>Mgr Brainstorming</td>
</tr>
<tr>
<td>Carolyn</td>
<td>Crow</td>
<td>334</td>
<td>790617</td>
<td>Raindance Consultant</td>
</tr>
<tr>
<td>Julie</td>
<td>Jensen</td>
<td>19</td>
<td>820929</td>
<td>Programmer/Analyst</td>
</tr>
</tbody>
</table>
Select Data for a Report

Data that meets specified criteria can be selected for a report by applying selection criteria to logical records.

How to Do It

Data can be selected:

1. With conditional statements set up by a \textbf{WHERE} clause. This retrieves only those records with a job ID equal to 3051.

2. By searching for defined values with \textbf{CONTAINS} or \textbf{MATCHES} patterns:

   - Retrieves employee records where the character set CAS appears somewhere in the name.
   - Retrieves employee records where the first three characters of the employee name are CAS.
   - Retrieves employee records where names start with CA followed by any alphabetic character.
   - Retrieves employee records where ids start with 02 followed by any two digits.

Example: Specify Data Selection Criteria for a Report

\textbf{Objective}

This report lists information about those employees with both the word ENTRY in their job title and ids in the ranges 0001 to 0048 and 0999 to 9999.

\textbf{Parameters}

\texttt{PATHAA \textit{EMPLOYEE-TABLE} WHERE}

\texttt{(EMP-ID-0415 LE '0048' OR * EMP-ID-0415 GT '0999')}\texttt{* AND DEPT-ID-0410 GT '1000'}

\texttt{EMPLOYEE-TABLE} names the logical record containing database data.
WHERE forms a compound conditional statement to specify the criteria for record retrieval.

* specifies a continuation line.

**Complete Code**

col. 2

```sql
DATABASE DICTNAME=DOCU_DICT
IN DB SS=EMPSS09
PATH AA
(EMP-ID-0415 LE '0048' OR
* EMP-ID-0415 GT '0999')
* AND DEPT-ID-0410 GT '1000'
01OUT D
013 EMPLOYEE LISTING
0151*005 EMP-ID-0415 HH 'ID'
0151*010 EMP-NAME-0415 HH 'EMPLOYEE NAME'
0151*020 DEPT-ID-0410 HH 'DEPT ID'
```  

**Result**

<table>
<thead>
<tr>
<th>EMPLOYEE ID</th>
<th>EMPLOYEE NAME</th>
<th>DEPT ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MONTE BANK</td>
<td>4000</td>
</tr>
<tr>
<td>4</td>
<td>HERBERT CRANE</td>
<td>3200</td>
</tr>
<tr>
<td>24</td>
<td>JANE DOUGH</td>
<td>3100</td>
</tr>
<tr>
<td>32</td>
<td>JANE FERNDALE</td>
<td>3200</td>
</tr>
<tr>
<td>45</td>
<td>GEORGE FONRAD</td>
<td>3200</td>
</tr>
<tr>
<td>29</td>
<td>JAMES GALLWAY</td>
<td>3100</td>
</tr>
<tr>
<td>3</td>
<td>JENNIFER GARFIELD</td>
<td>3100</td>
</tr>
<tr>
<td>28</td>
<td>PERCY GRANGER</td>
<td>3100</td>
</tr>
<tr>
<td>27</td>
<td>VLADIMIR HEAROWITZ</td>
<td>3100</td>
</tr>
<tr>
<td>20</td>
<td>JAMES JACOBIE</td>
<td>3100</td>
</tr>
<tr>
<td>19</td>
<td>JULIE JENSEN</td>
<td>3100</td>
</tr>
<tr>
<td>11</td>
<td>RUPERT JENSON</td>
<td>2600</td>
</tr>
<tr>
<td>16</td>
<td>TERENCE KLWELLEN</td>
<td>3200</td>
</tr>
<tr>
<td>31</td>
<td>HERBERT LIPSICH</td>
<td>3200</td>
</tr>
<tr>
<td>35</td>
<td>LARRY LITERATA</td>
<td>3100</td>
</tr>
<tr>
<td>15</td>
<td>RENE MAKER</td>
<td>5100</td>
</tr>
<tr>
<td>23</td>
<td>KATHERINE O'REAR</td>
<td>3100</td>
</tr>
<tr>
<td>48</td>
<td>NANCY TERNER</td>
<td>3200</td>
</tr>
<tr>
<td>21</td>
<td>RALPH TYRO</td>
<td>3100</td>
</tr>
</tbody>
</table>
5. Print information beyond that required for the report.


This section presents a summary of CA Culprit capabilities that can be useful in debugging.
- Reviewing CA Culprit Listings and Messages (see page 91)
- Using CULLUS48 (see page 98)
- Printing Additional Report Information (see page 99)

### Reviewing CA Culprit Listings and Messages

CA Culprit generates a sequential parameter listing, an input parameter listing, and run-time messages in addition to printed reports. Through these listings, errors can be flagged from all of CA Culprit’s processing phases.

### Sequential Parameter Listing

**Preliminary parameter syntax diagnostics** (precompile phase) are printed in the sequential parameter listing. Diagnostics are printed immediately below an invalid parameter or at the end of the listing. Asterisks (*) may underline the field in error:

```
REC  SALARY   160
***
E INCOMPLETE PARAMETER CARD
```

### Input Parameter Listing

**Warning and error messages** produced during parameter validation (compile phase) are printed on the input parameter listing. Error messages relating to a particular input parameter appear below the invalid parameter. Errors relating to the entire report are listed at the end. The final message indicates if the extract phase was performed:

```plaintext
<table>
<thead>
<tr>
<th>mm/dd/yy</th>
<th>INPUT PARAMETER LISTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01 OUT</td>
</tr>
<tr>
<td></td>
<td>EDIT LINE CC COLUMN</td>
</tr>
<tr>
<td></td>
<td>VALUE OR FIELD-NAME AND</td>
</tr>
<tr>
<td></td>
<td>EDIT OPTIONS</td>
</tr>
<tr>
<td></td>
<td>01 5</td>
</tr>
<tr>
<td></td>
<td>*020  SALARY</td>
</tr>
<tr>
<td></td>
<td>01 6</td>
</tr>
<tr>
<td></td>
<td>*010  'ALL DEPARTMENTS' HH 'DEPARTMENT'</td>
</tr>
</tbody>
</table>
```
Runtime Messages

Processing errors resulting from input-file processing, buffer dumps, and end-of-file statistics are listed in run-time messages (extract phase):

- Processing errors include database access and file-matching operations.
- Buffer dumps show the contents of the input buffer in character and hexadecimal representation when an error occurs in input processing.

If an error condition does not create a buffer dump, the dump can be forced by creating a zero-divide condition in the CA Culprit program:

```
010 ZERO
```

The dump looks like this:

```
mm/dd/yy RUN TIME MESSAGES valser Vnn.n PAGE 1
01/06/2001 01000016 SALARY S2=0010 HH 'TOTAL SALARIES'
C300106 EXTRACT WILL BE PERFORMED
C300119 PROFILE OPTION IN EFFECT: RELEASE = 6

Runtime Messages

Processing errors resulting from input-file processing, buffer dumps, and end-of-file statistics are listed in run-time messages (extract phase):

- Processing errors include database access and file-matching operations.
- Buffer dumps show the contents of the input buffer in character and hexadecimal representation when an error occurs in input processing.

If an error condition does not create a buffer dump, the dump can be forced by creating a zero-divide condition in the CA Culprit program:

```
010 ZERO
```

The dump looks like this:

```
mm/dd/yy RUN TIME MESSAGES valser Vnn.n PAGE 1
01/06/2001 01000016 SALARY S2=0010 HH 'TOTAL SALARIES'
C300106 EXTRACT WILL BE PERFORMED
C300119 PROFILE OPTION IN EFFECT: RELEASE = 6

Runtime Messages

Processing errors resulting from input-file processing, buffer dumps, and end-of-file statistics are listed in run-time messages (extract phase):

- Processing errors include database access and file-matching operations.
- Buffer dumps show the contents of the input buffer in character and hexadecimal representation when an error occurs in input processing.

If an error condition does not create a buffer dump, the dump can be forced by creating a zero-divide condition in the CA Culprit program:

```
010 ZERO
```

The dump looks like this:

```
mm/dd/yy RUN TIME MESSAGES valser Vnn.n PAGE 1
01/06/2001 01000016 SALARY S2=0010 HH 'TOTAL SALARIES'
C300106 EXTRACT WILL BE PERFORMED
C300119 PROFILE OPTION IN EFFECT: RELEASE = 6

Runtime Messages

Processing errors resulting from input-file processing, buffer dumps, and end-of-file statistics are listed in run-time messages (extract phase):

- Processing errors include database access and file-matching operations.
- Buffer dumps show the contents of the input buffer in character and hexadecimal representation when an error occurs in input processing.

If an error condition does not create a buffer dump, the dump can be forced by creating a zero-divide condition in the CA Culprit program:

```
010 ZERO
```

The dump looks like this:

```
mm/dd/yy RUN TIME MESSAGES valser Vnn.n PAGE 1
01/06/2001 01000016 SALARY S2=0010 HH 'TOTAL SALARIES'
C300106 EXTRACT WILL BE PERFORMED
C300119 PROFILE OPTION IN EFFECT: RELEASE = 6

Runtime Messages

Processing errors resulting from input-file processing, buffer dumps, and end-of-file statistics are listed in run-time messages (extract phase):

- Processing errors include database access and file-matching operations.
- Buffer dumps show the contents of the input buffer in character and hexadecimal representation when an error occurs in input processing.

If an error condition does not create a buffer dump, the dump can be forced by creating a zero-divide condition in the CA Culprit program:

```
010 ZERO
```

The dump looks like this:
End-of-file statistics are generated at the conclusion of extract file processing. The amount of information depends upon the type of run:

<table>
<thead>
<tr>
<th>mm/dd/yy</th>
<th>RUN TIME MESSAGES</th>
<th>volser Vnm.n</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C350023</td>
<td>INVALID NUMERIC DATA ENCONTERED IN REPORT 01</td>
<td>CAUSE OF DUMP</td>
<td></td>
</tr>
<tr>
<td>(CONT) ERROR OCCURRED DURING PROCESSING DETAILED LINE 1 FIELD 1</td>
<td>CHARACTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CONT) MOST RECENT SUBSCRIPT VALUE 0</td>
<td>WHERE DUMP OCCURRED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CONT) INPUT BUFFER NUMBER 1</td>
<td>OF INPUT BUFFER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CONT) RECORD BUFFER DUMP CHAR</td>
<td>06/09/01 JUNE</td>
<td>14 BLOOMER</td>
<td></td>
</tr>
<tr>
<td>ZONE</td>
<td>00691453000000266459000000001499303593590000003579573500000041016750000000000010359571080050500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIGIT</td>
<td>01...5...10....5...20....5...30....5...40....5...50....5...60....5...70....5...80....5...90....5...00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total DML Syntax Error Codes**

When the transparency runtime interface receives a request from a Total application program, it checks for DML syntax errors by performing a Total error-checking routine before calling CA IDMS/DB. The runtime interface returns the results of this error-checking routine in the form of standard Total error-status codes. Refer to the appropriate Total documentation for an explanation of these codes.

**Error Conditions Detected by the CA IDMS/DB**

When the transparency runtime interface issues a call to CA IDMS/DB to perform a database service requested by the Total application program, CA IDMS/DB processes (or attempts to process) the request and returns a CA IDMS/DB status code to the runtime interface. The runtime interface translates this status code to an equivalent Total error-status code and returns the Total code to the requesting Total application.

**Corresponding Total and CA IDMS/DB Status Codes**

The following table lists Total status codes and the corresponding CA IDMS/DB status codes. Some Total status codes are issued due to situations other than the return of a CA IDMS/DB status code resulting from a CA IDMS DML call. In cases where there is no reasonable Total status code to match a CA IDMS/DB status code, the actual CA IDMS/DB status code is used as the Total status. A CA IDMS/DB status code used instead of a Total status indicates an unusual situation.

If a status code is returned that is not noted in this table, see the appropriate Total error codes section or the **CA IDMS Messages and Codes Section**.
<table>
<thead>
<tr>
<th>Total status</th>
<th>CA IDMS/DB status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0508</td>
<td></td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td>0708</td>
<td></td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td>1108</td>
<td></td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td>1208</td>
<td></td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td>0210</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0310</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0510</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0710</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0810</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0910</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>1210</td>
<td></td>
<td>Function not allowed</td>
</tr>
<tr>
<td>0323</td>
<td></td>
<td>Invalid area or parameter list</td>
</tr>
<tr>
<td>IRLC</td>
<td>0302</td>
<td>DBGKEY out of page range</td>
</tr>
<tr>
<td></td>
<td>1202</td>
<td>DBGKEY out of page range</td>
</tr>
<tr>
<td></td>
<td>0371</td>
<td>Page range not found in DMCL</td>
</tr>
<tr>
<td></td>
<td>0971</td>
<td>The AREA being readied could not be found in the current DBNAME</td>
</tr>
<tr>
<td>DUPM</td>
<td>0705</td>
<td>Duplicates not allowed</td>
</tr>
<tr>
<td></td>
<td>0805</td>
<td>Duplicates not allowed</td>
</tr>
<tr>
<td></td>
<td>1205</td>
<td>Duplicates not allowed</td>
</tr>
<tr>
<td>NHLD</td>
<td>0306</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>0706</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>0806</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>1106</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>1506</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>1606</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>0313</td>
<td>No current of run unit</td>
</tr>
<tr>
<td></td>
<td>0513</td>
<td>No current of run unit</td>
</tr>
<tr>
<td></td>
<td>0813</td>
<td>No current of run unit</td>
</tr>
<tr>
<td></td>
<td>1613</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>0220</td>
<td>Current record not the same type as the named record</td>
</tr>
<tr>
<td></td>
<td>0520</td>
<td>Current record not the same type as the named record</td>
</tr>
<tr>
<td></td>
<td>0820</td>
<td>Current record not the same type as the named record</td>
</tr>
<tr>
<td></td>
<td>0225</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>0825</td>
<td>Currency not established</td>
</tr>
<tr>
<td></td>
<td>1225</td>
<td>Currency not established</td>
</tr>
<tr>
<td>Total status</td>
<td>CA IDMS/DB status</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>LOCK</td>
<td>0209</td>
<td>Named area readied in incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0709</td>
<td>Named area readied in incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0809</td>
<td>Named area readied in incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>1109</td>
<td>Named area readied in incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>1209</td>
<td>Named area readied in incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0221</td>
<td>Area other than the named area was readied in the incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0721</td>
<td>Area other than the named area was readied in the incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0821</td>
<td>Area other than the named area was readied in the incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>1121</td>
<td>Area other than the named area was readied in the incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>1221</td>
<td>Area other than the named area was readied in the incorrect usage mode</td>
</tr>
<tr>
<td></td>
<td>0966</td>
<td>Area not available for update</td>
</tr>
<tr>
<td>FULL</td>
<td>1211</td>
<td>Area full</td>
</tr>
<tr>
<td>NOIO</td>
<td>0318</td>
<td>Record not bound</td>
</tr>
<tr>
<td></td>
<td>0518</td>
<td>Record not bound</td>
</tr>
<tr>
<td></td>
<td>1218</td>
<td>Record not bound</td>
</tr>
<tr>
<td>RSRV</td>
<td>0928</td>
<td>Named areas previously readied</td>
</tr>
<tr>
<td>IMDL</td>
<td>0230</td>
<td>Record is owner of a nonempty set</td>
</tr>
<tr>
<td>FTYP</td>
<td>0331</td>
<td>Storage mode conflict</td>
</tr>
<tr>
<td>IOER</td>
<td>0970</td>
<td>Database or journal file will not open properly</td>
</tr>
<tr>
<td>NACT</td>
<td>1400</td>
<td>Run unit not bound</td>
</tr>
<tr>
<td></td>
<td>1469</td>
<td>Run unit not bound</td>
</tr>
<tr>
<td>FNAV</td>
<td>1408</td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td></td>
<td>1508</td>
<td>Invalid record or set name</td>
</tr>
<tr>
<td>ICOR</td>
<td>1472</td>
<td>Insufficient for LOAD or STORE</td>
</tr>
<tr>
<td>TFUL</td>
<td>1473</td>
<td>Run unit maximum reached</td>
</tr>
<tr>
<td>ACTV</td>
<td>1477</td>
<td>Run unit bound twice</td>
</tr>
</tbody>
</table>

Transparency Errors

When the transparency runtime interface receives a Total application program request that it cannot process because all the transparency requirements for handling the request are not met, or when the runtime interface receives a status code from CA IDMS/DB for which there is no Total equivalent, the runtime interface issues one of the following codes:

- IDMS status code
• CICS

• USxx

IDMS Status Code

The runtime interface receives a CA IDMS/DB status code in response to a request for database services; a Total equivalent to this request does not exist.

⚠️ **Note:** For more information on CA IDMS/DB status codes, see the *CA IDMS Messages and Codes Section.*

### CICS

The CICS code indicates that the interface between the central version and CICS has not been started. The transparency runtime interface has issued a call to CA IDMS/DB to process a request from a Total CICS application, and CA IDMS/DB has returned a status code of 68.

⚠️ **Note:** Minor code 68 can be issued with any major code.

### US

A US status code indicates that the transparency runtime interface cannot process a DML command issued by the Total application. US status codes can result from any of the following errors in a Total DML command:

- The key needed to perform the CA IDMS/DB equivalent of a Total store command for a variable file is not provided in the data-list parameter.

- A Total store command has been issued for a coded variable file, but the CA IDMS/DB equivalent of the record code in the data area is not defined in the subschema.

- The linkpath named in a Total linkpath parameter is not valid for the record code in the data area.

- The data-list parameter in the application call to ETOTBINT or ETOTCINT is not correctly punctuated.

- ETOTTBL was not found. During SINON processing, the runtime control table (ETOTTBL), designated by the DBMOD parameter or the SINON call, could not be loaded or located.

- ETOTTRAN was not found (batch only). If ETOTTRAN was not linked during the link edit of ETOTBINT, a copy of ETOTTRAN could not be loaded or located.

- Runtime work area storage was not available. At the time the application issued a SINON call, enough runtime work area storage for processing was not available.
• RQLOC function was invoked under central version. This function is available in local mode only.

• RQLOC-FILE (record name) was not found in subschema.

• RQLOC-FILE (record name) was not found in ETOTTBL.

US Status Codes

The following table lists US status codes.

<table>
<thead>
<tr>
<th>Total status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>US01</td>
<td>Internal hashing error -- REGEN ETOTTBL</td>
</tr>
<tr>
<td>US02</td>
<td>Record buffer too small</td>
</tr>
<tr>
<td>US03</td>
<td>Set key missing</td>
</tr>
<tr>
<td>US04</td>
<td>Missing set relation</td>
</tr>
<tr>
<td>US05</td>
<td>Improper key connect</td>
</tr>
<tr>
<td>US06</td>
<td>Invalid record code</td>
</tr>
<tr>
<td>US07</td>
<td>Insufficient storage (batch)</td>
</tr>
<tr>
<td>US08</td>
<td>Error in load of ETOTTRAN or ETOTTBL</td>
</tr>
<tr>
<td>US09</td>
<td>RQLOC function invoked under central version</td>
</tr>
<tr>
<td>US10</td>
<td>RQLOC record or file name not found in the subschema</td>
</tr>
<tr>
<td>US99</td>
<td>Element table not correct for <strong>REST</strong>; table must be regenerated</td>
</tr>
</tbody>
</table>

The Report

Logic and format errors, generated in the output phase, can frequently be detected by reviewing the printed report. Error messages are printed where the processing problem occurs to the limit set in the OE= specification on the PROFILE parameter:

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>AVERAGE START AGE BY DEPARTMENT AND STATUS CODE</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
<th>BIRTH YEAR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>BLOOMER</td>
<td>01</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>HUTTON</td>
<td>01</td>
<td>77</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>JENSON</td>
<td>01</td>
<td>80</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>KIMBALL</td>
<td>01</td>
<td>78</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>KING</td>
<td>01</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>NICEMAN</td>
<td>01</td>
<td>80</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>CLOUD</td>
<td>01</td>
<td>77</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>DONOVAN</td>
<td>01</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>MOON</td>
<td>01</td>
<td>78</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>BRAINSTORMING</td>
<td>BREEZE</td>
<td>01</td>
<td>79</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>BRAINSTORMING</td>
<td>CROW</td>
<td>01</td>
<td>79</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>BRAINSTORMING</td>
<td>LANCHESTER</td>
<td>01</td>
<td>75</td>
<td>32</td>
</tr>
</tbody>
</table>

NUMBER OF EMPLOYEES BY STATUS CODE: 6
NUMBER OF EMPLOYEES BY DEPARTMENT: 6
AVERAGE AGE: 27
AVERAGE AGE: 27

NUMBER OF EMPLOYEES BY STATUS CODE: 3
NUMBER OF EMPLOYEES BY DEPARTMENT: 3
AVERAGE AGE: 32
AVERAGE AGE: 32
### Using CULLUS48

Status or diagnostic messages can be added to the run-time message section of a CA Culprit job by using the CULLUS48 user module. A message containing up to 132 characters can be executed in type 7 logic, as follows:

```
010 MESSAGE '0 THIS IS A MESSAGE %'

927 CALL US48 (MESSAGE)
```

**Note:** The message, an alphanumeric string placed within single quotation marks, must start with a carriage control character and end with two percent signs.

More information about CULLUS48 can be found in [Creating a Run-time Message](https://docops.ca.com/pages/viewpage.action?pageId=309428331).
Printing Additional Report Information

Debugging complex reports can be made easier by printing additional information in your report. You can:

- Print out sort key values in order to understand CA Culprit's sort sequence
- Print out detail lines for totals-only reports in order to determine the source of totals generated for the report
- Release values between complex calculations in order to track the processing sequence

Example: Printing Additional Report Information

Objective

This example lists values obtained while computing the volume of a sphere, in cubic centimeters, from diameters measured in inches. The program releases a detail line after every second arithmetic operation.

Procedure

The program issues several arithmetic instructions to compute the volume of the sphere. CA Culprit automatically rounds the result of a calculation. As the results below indicate, the code should specify truncation for the arithmetic operations in this example.

Code

```
col. 2
IN 80
REC TANK-DIA-IN 1 3
010 RADIUS-IN
010 RADIUS-CM
010 WORK-FLD
010 CUBIC-CM
013 CALCULATING THE CC VOLUME OF A SPHERE FROM A DIAMETER IN INCHES
0152*010 TANK-DIA-IN HF
0152*020 RADIUS-IN HF
0152*030 RADIUS-CM HF
0152*040 WORK-FLD HF
0152*050 CUBIC-CM HF
.
.
017 TANK-DIA / 2 RADIUS-IN
017 RADIUS-IN X 2.54 RADIUS-CM
017 RELS 2
017 COMPUTE (RADIUS-CM X RADIUS-CM X RADIUS-CM) WORK-FLD
017 WORK-FLD X 3.1416 WORK-FLD
017 RELS 2
017 WORK-FLD X 4 WORK-FLD
017 WORK-FLD / 3 CUBIC-CM
017 RELS 2
.
.
```
Result

The values released between calculations are:

<table>
<thead>
<tr>
<th>TANK-DIA-IN</th>
<th>RADIUS-IN</th>
<th>RADIUS-CM</th>
<th>WORK-FLD</th>
<th>CUBIC-CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>8</td>
<td>20</td>
<td>25,133</td>
<td>33,511</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>20</td>
<td>180,532</td>
<td>33,511</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>20</td>
<td>125,665</td>
<td>33,511</td>
</tr>
<tr>
<td>45</td>
<td>24</td>
<td>60</td>
<td>125,665</td>
<td>33,511</td>
</tr>
</tbody>
</table>

The result should have been:

<table>
<thead>
<tr>
<th>TANK-DIA-IN</th>
<th>RADIUS-IN</th>
<th>RADIUS-CM</th>
<th>WORK-FLD</th>
<th>CUBIC-CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>7.5</td>
<td>19.05</td>
<td>21,718.8</td>
<td>28,958.4</td>
</tr>
<tr>
<td>15</td>
<td>7.5</td>
<td>19.05</td>
<td>86,875.2</td>
<td>28,958.4</td>
</tr>
<tr>
<td>45</td>
<td>22.5</td>
<td>57.15</td>
<td>100,532</td>
<td>28,958.4</td>
</tr>
</tbody>
</table>

The released values show that rounding occurs in all calculations. Truncation should be indicated in the code for correct results.

Totals Processing Techniques

CA Culprit processes data in two input/output phases. The first phase executes type 7 logic against the input data and outputs a temporary work file (the extract file). The second phase reads the extract file after it has been sorted and performs final calculations on accumulated totals, executes control breaks, and generates report output.

Within the CA Culprit final processing phase you can perform calculations on accumulated totals, obtain multiple level subtotals, and access current values of work fields that do not appear on SORT or type 5 parameters.

- Calculations on Accumulated Totals (see page 100)
- Multiple-level Subtotals (see page 102)
- Creating a Sparse Listing (see page 105)
- Obtaining Work Field Values (see page 106)

Calculations on Accumulated Totals

You can obtain subtotals, perform calculations on totals, and perform logical operations on totals.

How to Do It

Use:

- **Control breaks** on the SORT parameter to obtain subtotals

- **Type 8** logic to:
  - Test control break levels
Demonstration 14

Objective

This report prints account balances for several branch offices and computes an average branch balance.

Procedure

- The branch number is the sort key value obtained when a control break occurs on BRANCH (LEVL=1).
- Branch account totals are automatically obtained by the control break.
- An average balance is computed in type 8 logic from the automatic grand total amount and a count of the branch control breaks.

Complete Code

col. 2
REC CURRENT-BAL 4 8 2 DP=2
01OUT T01SORT BRANCH 0 $Control break for branch number
010 PRINT1 'BRANCH' 010 COUNT010 AVERAGE DP=2
013 SUMMARY REPORT OF AVERAGE TRANSACTIONS
0141*001 ' ' 0151*010 CURRENT-BAL
01610020 PRINT1
01610027 BRANCH $Prints BRANCH value
01610031 'TOTAL BALANCE' 01610045 CURRENT-BAL SZ=11
016200310 'TOTAL BALANCE' 01620045 CURRENT-BAL SZ=11 $Prints branch totals
016300310 'AVERAGE PER BRANCH' 01630050 AVERAGE
016300510 'AVERAGE' 016300511 $Prints the average
018 IF LEVL EQ 2 200 $Test for grand-total time
018 COUNT + 1 COUNT
018 TAKE 1 $Prints branch information
018200 CURRENT-BAL / COUNT AVERAGE $Uses the grand total of CURRENT-BAL TAKE (2 3) $Prints type 6 lines 2 and 3

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>SUMMARY REPORT OF AVERAGE TRANSACTIONS</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCH 081 TOTAL BALANCE</td>
<td>999,999.99</td>
<td>1,358,349.11</td>
<td></td>
</tr>
<tr>
<td>BRANCH 010 TOTAL BALANCE</td>
<td>345,678.90</td>
<td>345,575.18</td>
<td></td>
</tr>
</tbody>
</table>
Multiple-level Subtotals

You can obtain automatically calculated subtotals for each control-break level. After the extract file is sorted, records are read into the output (CULE) phase of CA Culprit. When a control break occurs, the type 8 logic is executed and the subtotal accumulators for the level with the control break are reinitialized after the break. If no control break occurs, the values of the numeric fields appearing on type 5 lines are added to the accumulators.

How to Do It

1. Code the control break on the SORT parameter.
2. Specify the type 5 parameter field name on a type 6 parameter.
3. Test control break levels in type 8 logic.
4. Specify the type 6 line to be printed with the TAKE command in type 8 logic.

Demonstration 15

Objective

This report lists the sales for each plant and prints subtotals by plant and division.

Procedure

- **Control break levels** are printed on the report to show where accumulator initialization occurs. (The contents of the accumulators during processing are shown in the table below.)

- **COUNT**, a numeric work field on a type 5 line, produces a subtotal of the number of records processed:
  - The subtotal for each plant is obtained on the first type 6 line.
  - The subtotal for each division is obtained on the second type 6 line.
  - The total number of records for all divisions is obtained on the third type 6 line.

- **DIVISION** and **PLANT**, are alphanumeric sort keys:
  - The sort key value (division and plant number) is obtained at the appropriate control break levels.
- Omission of DIVISION on the third type 6 line results in the division number on the grand total line being left out.

- **AMOUNT** is a numeric input item:
  - On a type 5 line, AMOUNT prints on detail lines.
  - On the first type 6 line, AMOUNT prints the subtotal for each plant.
  - On the second type 6 line, AMOUNT prints the subtotal for each division.
  - On the third type 6 line, AMOUNT prints the grand total.

**Accumulated Total Values**

The figure below shows the total values that occur during the processing of the code for the sales report:

**Accumulated Total Values for Sales Report**

<table>
<thead>
<tr>
<th>SORT KEY VALUES</th>
<th>TYPE 5 FIELDS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LEVEL</strong> 2 DIVISION</td>
<td><strong>LEVEL</strong> 1 PLANT</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>02</td>
</tr>
<tr>
<td>00</td>
<td>02</td>
</tr>
<tr>
<td>00</td>
<td>02</td>
</tr>
<tr>
<td>00</td>
<td>03</td>
</tr>
</tbody>
</table>

* denotes initialization of the bucket to zero after the control break

The values of the accumulators during the output processing of the Sales by Division and Plant code are incremented until a control break occurs. A control break causes initialization of the accumulators for the level having the control break.

**Complete Code**

col. 2

REC PLANT 3 2
REC AMOUNT 5 4 2 DP=2
REC TYPE 9 1
REPORT NO. 01 SALES BY DIVISION AND PLANT mm/dd/yy PAGE 1

DIVISION NUMBER: 01

<table>
<thead>
<tr>
<th>PLANT</th>
<th>AMOUNT</th>
<th>COUNT TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>50.00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>20.00</td>
<td></td>
</tr>
</tbody>
</table>

01 01 90.00 3

01 02 40.00
01 02 50.00

01 02 90.00 2

01 180.00 5

REPORT NO. 01 SALES BY DIVISION AND PLANT mm/dd/yy PAGE 2

DIVISION NUMBER: 02

<table>
<thead>
<tr>
<th>PLANT</th>
<th>AMOUNT</th>
<th>COUNT TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>.00</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>40.00</td>
<td></td>
</tr>
</tbody>
</table>

02 02 60.00 3

02 60.00 3

REPORT NO. 01 SALES BY DIVISION AND PLANT mm/dd/yy PAGE 3

DIVISION NUMBER: 06

<table>
<thead>
<tr>
<th>PLANT</th>
<th>AMOUNT</th>
<th>COUNT TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>40.00</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>40.00</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>40.00</td>
<td>1</td>
</tr>
</tbody>
</table>

GRAND TOTAL: 280.00 9
Creating a Sparse Listing

You can eliminate repeated occurrences of selected fields on printed output by forcing a control break on a work field that is not tied to input data.

How to Do It

1. Define a numeric work field.
2. Code the work field on a SORT parameter as a low-order sort key with a control break.
3. Code a type 7 statement that increments the control-break work field by 1. (A control break will then occur on each extract record when the value of the work field changes.)
4. Code type 8 instructions for processing the control break.

Demonstration 16

Objective

This report uses a forced control break to make individual plant and division codes available for labeling print lines.

Procedure

- A numeric work field (WORK-X) is placed on a SORT parameter to force a control break each time the value of the work field changes.
- The value of WORK-X is incremented in type 7 logic.
- The control break on WORK-X causes:
  - Type 8 statements to be processed for every extract record. Type 8 logic checks for a control break at LEVL 4, 3, or 2.
  - If one of the type 8 conditions is true, processing branches to the appropriate type 6 subtotal or total print line.

Complete Code

col. 2

REC PLANT 3 2
REC AMOUNT 5 4 2 DP=2
01 SORT DIVISION 0 PLANT 0 WORK-X + $Forced control break
013 SALES BY PLANT AND DIVISION
010 COUNT 1
010 PLANT-SAVE ' ' 010 DIV-SAVE ' ' 010 WORK-X01OUT T
0151*0000 COUNT
Obtaining Work Field Values

Work fields that do not appear on SORT or type 5 parameters are not part of the extract record. Any references made to such work fields during output processing return only the most current value of the work field.
To obtain the value of a subscripted work field, see the section "Getting Started with CA Culprit".

**Demonstration 17**

**Objective**

The Summary Report of Account Totals is repeated here to show the use of the work field AVERAGE.

**Procedure**

- **AVERAGE** is computed in type 8 logic.
- The computation results are stored in the work field AVERAGE.
- The current value of AVERAGE is printed by using a type 6 parameter reference.

**Complete Code**

```
col. 2
IN 80 F 320 PS(TAPE)
REC BRANCH 1 3
REC CURRENT-BAL 4 8 2 DP=2
01OUT T
01SORT BRANCH,0
01 PRINT1 'BRANCH'
010 PRINT10 AVERAGE DP=2 $Work field holding computation results
013 SUMMARY REPORT OF AVERAGE TRANSACTIONS
0141*001 ' ' 0151*010 CURRENT-BAL
01610020 PRINT1
01610027 BRANCH
01610031 'TOTAL BALANCE'
016200310 'TOTAL BALANCE'
01620045 CURRENT-BAL SZ=11
016300260 'AVERAGE PER BRANCH'01630045 AVERAGE SZ=11 $Print the value of $Compute the work field value
018 IF LEVL EQ 2 200
018 COUNT + 1 COUNT
018 TAKE 1
018200 CURRENT-BAL / COUNT AVERAGE
```

**Result**

```
REPORT NO. 01 SUMMARY REPORT OF AVERAGE TRANSACTIONS mm/dd/yy PAGE 1
BRANCH 081 TOTAL BALANCE 999,999.99
BRANCH 010 TOTAL BALANCE 1,358,349.11
BRANCH 011 TOTAL BALANCE 345,678.90
BRANCH 020 TOTAL BALANCE 345,575.18
TOTAL BALANCE 3,049,603.18
AVERAGE PER BRANCH 762,400.80
```

**Using Subscripts**

15-Jan-2018 107/225
Subscripts define and provide access to particular occurrences of repeating data fields. References can be made to work field occurrences, segments of a field, and fixed or variable repeating input fields.

- **Explicit Subscripts** (see page 108)
- **Zero Subscripts** (see page 111)
- **Fixed Repeating Fields** (see page 113)
- **Variable Repeating Groups** (see page 115)
- **Floating Fields** (see page 116)
- **Obtaining Accumulated Totals** (see page 118)
- **Obtaining Specific Field Values** (see page 120)
- **Obtaining Sort-key Values** (see page 122)

### Explicit Subscripts

You can reference a field occurrence directly by:

- Defining an explicit number of occurrences in a given work field:
  
  ```
  010 CHARACTERS.3 'A' 'B' 'C'
  ```

- Referencing a specific work field occurrence by a number or integer name:
  
  ```
  0151*010 CHARACTERS.2 (By a number)
  0151*020 CUSTOMER-NAME.INDEX (By an integer)
  ```

### How to Do It

Code a work field parameter with:

1. The name of the repeating field
2. A period (.) immediately followed by a numeric literal that specifies the number of repeating elements
3. Optional initialization of each occurrence of the field

### Demonstration (1) Using a Literal Subscript

**Objective**

This report lists the interest due on loans. Based on the size of the loan, there are three possible interest rates. Literal subscripts are used to reference each specific interest rate occurrence.
Procedure

- Interest rates are stored in a work field **INTEREST.3**.

- **Type 7** logic references each occurrence of the work field.

- **INTEREST.3** has a **DP=4** specification, which translates each occurrence into a 16-byte packed decimal.

- The size of the printed interest rate is controlled by using **SZ=4** on line **0151*030**.

Complete Code

col. 2

```plaintext
IN 200 F 400 PS(TAPE)
REC NAME 5 20
REC LOAN 160 7 3 DP=2
01OUT 010 INTEREST.3 DP=4 0.0525 0.0550 0.0575 $The subscripted work field
010 AMOUNT DP=2
010 INT DP=4
0151*010 NAME HH 'CUSTOMER'
0151*020 LOAN SZ=8 HH 'LOAN AMOUNT'0151*030 INT SZ=4 DP=4 HH 'INTEREST' 'RA TE'
0151*040 AMOUNT SZ=7 DP=2 HH 'INTEREST' 'AMOUNT'
017 IF LOAN GT 50000.00 100
017 IF LOAN EQ (20000.00 TO 49999.99) 200
017 IF LOAN LT 20000.00 300
017100 COMPUTE ROUND (LOAN X INTEREST.1) AMOUNT $Use the first occurrence
017 MOVE INTEREST.1 TO INT
017 TAKE
017200 COMPUTE ROUND (LOAN X INTEREST.2) AMOUNT $Use the second occurrence
017 MOVE INTEREST.2 TO INT
017 TAKE
017300 COMPUTE ROUND (LOAN X INTEREST.3) AMOUNT $Use the third occurrence
017 MOVE INTEREST.3 TO INT
017 TAKE

Result

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>LOAN AMOUNT</th>
<th>RATE</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUNE BLOMBER</td>
<td>15,000.00</td>
<td>.0575</td>
<td>862.50</td>
</tr>
<tr>
<td>EDWARD HUTTON</td>
<td>44,000.00</td>
<td>.0550</td>
<td>2,420.00</td>
</tr>
<tr>
<td>RUPERT JENSON</td>
<td>82,000.00</td>
<td>.0525</td>
<td>4,305.00</td>
</tr>
<tr>
<td>MARIANO KIMBALL</td>
<td>45,000.00</td>
<td>.0550</td>
<td>2,475.00</td>
</tr>
<tr>
<td>DORIS KING</td>
<td>14,500.00</td>
<td>.0575</td>
<td>833.75</td>
</tr>
<tr>
<td>BRIAN NICEMAN</td>
<td>14,000.00</td>
<td>.0575</td>
<td>805.00</td>
</tr>
<tr>
<td>HERBERT CRANE</td>
<td>75,000.00</td>
<td>.0525</td>
<td>3,937.50</td>
</tr>
<tr>
<td>JANE FERDALE</td>
<td>22,500.00</td>
<td>.0550</td>
<td>1,237.50</td>
</tr>
<tr>
<td>GEORGE FONRAD</td>
<td>14,750.00</td>
<td>.0575</td>
<td>848.13</td>
</tr>
<tr>
<td>ROBIN GARDNER</td>
<td>14,000.00</td>
<td>.0575</td>
<td>805.00</td>
</tr>
<tr>
<td>DOUGLAS KAHALLY</td>
<td>20,000.00</td>
<td>.0550</td>
<td>1,100.00</td>
</tr>
<tr>
<td>TERENCE KLEWELLEN</td>
<td>43,000.00</td>
<td>.0550</td>
<td>2,365.00</td>
</tr>
<tr>
<td>SANDY KRAEMER</td>
<td>14,000.00</td>
<td>.0575</td>
<td>805.00</td>
</tr>
<tr>
<td>HERBERT LIPSICH</td>
<td>18,500.00</td>
<td>.0575</td>
<td>1,063.75</td>
</tr>
<tr>
<td>NANCY TERRER</td>
<td>13,000.00</td>
<td>.0575</td>
<td>747.50</td>
</tr>
<tr>
<td>BETH CLOUD</td>
<td>52,750.00</td>
<td>.0525</td>
<td>2,769.38</td>
</tr>
<tr>
<td>ALAN DONOVAN</td>
<td>33,500.00</td>
<td>.0550</td>
<td>1,842.50</td>
</tr>
<tr>
<td>DANIEL MOON</td>
<td>72,000.00</td>
<td>.0525</td>
<td>3,780.00</td>
</tr>
<tr>
<td>ROY ANDALE</td>
<td>33,500.00</td>
<td>.0550</td>
<td>1,842.50</td>
</tr>
<tr>
<td>HARRY ARM</td>
<td>46,000.00</td>
<td>.0550</td>
<td>2,530.00</td>
</tr>
<tr>
<td>C. BREEZE</td>
<td>38,000.00</td>
<td>.0550</td>
<td>2,090.00</td>
</tr>
<tr>
<td>CAROLYN CROW</td>
<td>37,500.00</td>
<td>.0550</td>
<td>2,062.50</td>
</tr>
<tr>
<td>BURT LANCHESTER</td>
<td>54,500.00</td>
<td>.0525</td>
<td>2,861.25</td>
</tr>
<tr>
<td>RENÉ MAKER</td>
<td>85,500.00</td>
<td>.0525</td>
<td>4,462.50</td>
</tr>
<tr>
<td>RICHARD MUNYON</td>
<td>36,000.00</td>
<td>.0550</td>
<td>1,980.00</td>
</tr>
<tr>
<td>RICHARD WAGNER</td>
<td>47,000.00</td>
<td>.0550</td>
<td>2,585.00</td>
</tr>
<tr>
<td>TOM FITZHUGH</td>
<td>13,000.00</td>
<td>.0575</td>
<td>747.50</td>
</tr>
</tbody>
</table>
Demonstration (2) Using a Counter as a Subscript

Objective

This report lists data occurrences that are stored in a work field.

Procedure

The value of the counter INDEX is used as a subscript on a type 5 parameter.

Complete Code

```
col. 2

IN 80
REC CODE 1 2 2
020 FOREIGN-MAKES.30  'ROLLS ROYCE'  'ALPHA ROMEO'  'FIAT'
*  'VOLKSWAGEN'  'PORCHE'  'LANCIA'  'BMW'  'AUDI'  'DATSUN'  'TOYOTA'
** 'MERCEDES'  'MASERATI'  'LADA'  'SAAB'  'HONDA'  'MAZDA'  'RENAULT'
*  'VOLVO'  'LAMBORGHINI'  'TRIUMPH'  'LOTUS'  'FERRARI'

020 INDEX  $Counter work field

02OUT 80
025*010 FOREIGN-MAKES.INDEX  027010  INDEX + 1 INDEX  $Increment counter
027020 IF INDEX GT 30  STOP-RPT  $Check for subscript value
027 RELS
027 B 10
```

Result

```
ROLLS ROYCE
ALPHA ROMEO
FIAT
VOLKSWAGEN
PORCHE
LANCIA
BMW
DATSUN
MERCEDES
```
Zero Subscripts

You can define an alphanumeric work field, redefine adjacent storage to contain segments of the work field, and reference the segments with explicit subscripts.

How to Do It

Define the subscripted work field by coding a work field parameter (0) with:

1. The name of the primary work field that redefines the adjacent fields
2. A period (.) followed immediately by zero (0)
3. Spaces (enclosed in single quotation marks) to indicate the length of the primary work field

```
010 NAME-0.0 '   '
```

Redefine the storage area that immediately follows the zero-subscripted work field by coding on one or more work field parameters (in alphabetical or numerical order):

1. The prefix of the field name being defined
2. Spaces (enclosed in single quotation marks) to indicate the length of the redefined field:

```
010 NAME-0.0 '   '
010 NAME-1 '   '
```

Reference a zero-subscripted work field by using:

1. The name of the work field
2. An explicit subscript

```
010 FIELD-0.0 '   '
010 FIELD-1 '   '
010 FIELD-2 '   '
010 FIELD-3 '   '
0151*010 FIELD-0.1 $References the first occurrence of FIELD-0.0
```

Demonstration 18

Objective
This report lists the name, password, and address of employees. The passwords are created from the first, third, and fifth letters of the employees' first names.

Procedure

- The NAME-1 work field is redefined into 1-byte segments.
- WRKFLD-1, -2, and -3 are combined into a single field (WRKFLD-0.0).
- The five address segments are combined into one field (ADDRESS-0.0).

Complete Code

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0170</td>
<td>MOV</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
<tr>
<td>0170</td>
<td>MOVE</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>NAME</th>
<th>PASSWORD</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROY</td>
<td>ANDALE</td>
<td>44 TRIGGER RD FRAMINGHAM MA 03461</td>
</tr>
<tr>
<td>MICHAEL</td>
<td>ANGELO</td>
<td>587 CISTINE DR WELLESLEY MA 01568</td>
</tr>
<tr>
<td>HARRY</td>
<td>ARM</td>
<td>77 SUNSET STRIP NATICK MA 02178</td>
</tr>
<tr>
<td>MONTE</td>
<td>BANK</td>
<td>14 JETER TERR LEXINGTON MA 01675</td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>356 PINKY LN NITACIK MA 02718</td>
</tr>
<tr>
<td>CHARLES</td>
<td>BOWER</td>
<td>450 THRUWAY ST MANSFIELD MA 02301</td>
</tr>
<tr>
<td>C.</td>
<td>BREEZE</td>
<td>99 VERDE ST STOUGHTON MA 02070</td>
</tr>
<tr>
<td>TERRY</td>
<td>CLOTH</td>
<td>15 DAWSON ST MEDFORD MA 02332</td>
</tr>
<tr>
<td>ROONEY</td>
<td>GARDNER</td>
<td>68 75TH ST LOWELL MA 02945</td>
</tr>
</tbody>
</table>

15-Jan-2018 112/225
### Fixed Repeating Fields

You can retrieve input data items that occur a fixed number of times.

**How to Do It**

Define the related items as a group of similar elements before processing takes place.

1. Define the **group** of repeating fields by coding one REC parameter with:
   a. A group name for the repeating fields
   b. The start position of the group
   c. The keyword **GROUP**
   d. A 2-character group identifier
   e. The total length (in bytes) of the group, followed by a period (.)
   f. The number of occurrences in the group
2. Define the **elements** of the group by coding a REC parameter for each element of the group with:

   a. The name of the element
   
   b. The start position *within the group*
   
   c. The length of the element
   
   d. The keyword **ELMNT**
   
   e. The same 2-character identifier as the group to which the element belongs

3. Reference the elements by using explicit subscripts.

**Demonstration 19**

**Objective**

This report prints interest rates in three columns.

**Procedure**

The input records contain three repeating rates:

- **INTEREST-RATE-DATA** defines the group.

- **INTEREST-RATE** defines the individual elements.

- Explicit subscripts reference the elements on type 5 parameters.

**Complete Code**

```
col. 2

IN 80
REC INTEREST-RATE-DATA 1 GROUP AA 4.3REC INTEREST-RATE 1 4 2 DP=4 ELMNT AA00OUT 80 D
003 INTEREST RATE TABLE
0051*010 INTEREST-RATE.1 HH ' ' 'RATE CODE 1' '(ACCOUNT TYPE 1)'
0051*020 INTEREST-RATE.2 HH 'RATE CODE 2' '(ACCOUNT TYPE 2)'
0051*030 INTEREST-RATE.3 HH 'RATE CODE 3' '(ACCOUNT TYPE 3)'
```

**Result**

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>INTEREST RATE TABLE</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RATE CODE 1 (ACCOUNT TYPE 1)</td>
<td>RATE CODE 2 (ACCOUNT TYPE 2)</td>
<td>RATE CODE 3 (ACCOUNT TYPE 3)</td>
</tr>
<tr>
<td></td>
<td>.0500</td>
<td>.0525</td>
<td>.0550</td>
</tr>
<tr>
<td></td>
<td>.0525</td>
<td>.0550</td>
<td>.0575</td>
</tr>
<tr>
<td></td>
<td>.0550</td>
<td>.0575</td>
<td>.0600</td>
</tr>
<tr>
<td></td>
<td>.0550</td>
<td>.0575</td>
<td>.0625</td>
</tr>
</tbody>
</table>
Variable Repeating Groups

You can retrieve occurrences of input fields that repeat a variable number of times.

How to Do It

Define the group by coding one REC parameter with:

1. An identifying name for the group of fields
2. The start position
3. The keyword GROUP
4. A group identifier
5. The total length of one occurrence of the group (in bytes), followed by a period (.)
6. The name of the field in the fixed portion of the record that contains the number of occurrences of the group

Define the elements of the group by coding a REC parameter for each element of the group with:

1. The name of the element
2. The start position within the group
3. The length of the element
4. The keyword ELMNT
5. The identifier of the group

Reference the elements by using explicit subscripts.

Demonstration 20

Objective

This report lists insurance policy numbers and coverage dates for several store locations. The number of policies varies for each store.

Procedure

- Three fields (LOCATION-ID, LOC-EFF-DATE, and LOC-EXP-DATE) in a variable repeating data group are defined on REC parameters:
  - LOCATION-INFO is the assigned group name.
- **LOCATION-NUMBER** is the input record field containing the number of occurrences.

- **INDEX**, which is used as an explicit subscript, is a work field that contains the current occurrence count.

**Complete Code**

```
col. 2
IN 80 F 320 PS(TAPE)
REC POLICY-NUMBER 5 10 2
REC COVERAGE-CODE 15 3
REC EFFECTIVE-DATE 18 6 2
REC LOCATION-NUMBER 24 4
REC LOCATION-INFO 28 GROUP AA 14.LOCATION-NUMBER
AA010UT D
010 INDEX 1 $Counter
0141*010 'POLICY NUMBER'
0141*020 'COVERAGE'
0141*030 'EFFECTIVE DATE'
0141*040 'LOCATION ID'
0141*050 'LOC-EFF-DATE'
0141*060 'LOC-EXP-DATE'
01420001 ' '
0151*010 POLICY-NUMBER F0
0151*020 COVERAGE-CODE
0151*030 EFFECTIVE-DATE FD
0152*040 LOCATION-ID_INDEX0152*050 LOC-EFF-DATE_INDEX FD
0152*060 LOC-EXP-DATE_INDEX FD
017 RELS 1 $Print first detail line; continue processing
017100 RELS 2 $Print second detail line; continue processing
017 INDEX + 1 INDEX017 IF INDEX LE LOCATION-NUMBER 100 $Test subscript value
017 MOVE 1 TO INDEX $Reset subscript value
017 DROP
```

**Result**

<table>
<thead>
<tr>
<th>POLICY NUMBER</th>
<th>COVERAGE</th>
<th>EFFECTIVE DATE</th>
<th>LOCATION ID</th>
<th>LOC-EFF-DATE</th>
<th>LOC-EXP-DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111</td>
<td>010</td>
<td>mm/dd/yy</td>
<td>1</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>222222</td>
<td>011</td>
<td>mm/dd/yy</td>
<td>5</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>333333</td>
<td>005</td>
<td>mm/dd/yy</td>
<td>10</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>4444444</td>
<td>092</td>
<td>mm/dd/yy</td>
<td>4</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>

**Note:** The report would show actual dates in the format shown.

**Floating Fields**

You can retrieve a field that occurs immediately after a variable repeating group (floating field) by creating code that is relative to the end of the variable group.
How to Do It

Define the floating group by coding one REC parameter with:

1. The assigned group name
2. A start position of 1
3. The keyword GROUP
4. A unique group identifier
5. The group identifier of the last variable repeating group
6. The length of the floating field in bytes

Define the field by coding a REC parameter with:

1. The name of the floating field
2. A start position of 1
3. The length of the field
4. The keyword ELMNT
5. The identifier of the group

Reference the field directly.

Demonstration 21

Objective

This report lists insurance coverage and risk level for a series of stores from an input file containing the floating field RISK-LEVEL.

Procedure

- OTHER-DATA defines the floating group.
- RISK-LEVEL defines the floating field.
- RISK-LEVEL is referenced directly on a type 5 parameter.

Complete Code

col. 2

IN 80
REC POLICY-NUMBER 5 10 2
REC COVERAGE-CODE 15 3
REC EFFECTIVE-DATE 18 6 2
CA IDMS - 19.0

```plaintext
REC LOCATION-NUMBER 24 4 2
REC LOCATION-INFO 28 GROUP AA 14.LOCATION-NUMBER
REC LOCATION-ID 1 2 2 ELMNT AA
REC LOC-EFF-DATE 3 6 2 ELMNT AA
REC LOC-EXP-DATE 9 6 2 ELMNT AAREC OTHER-DATA 1 GROUP BB AA 1REC RISK-LEVEL 1 1 ELMNT BB01OUT D
010 INDEX 1
0141*010 'POLICY NUMBER'
0141*020 'COVERAGE'
0141*030 'EFFECTIVE DATE'
0141*035 'RISK LEVEL'
0141*040 'LOCATION ID'
0141*050 'LOC-EFF-DATE'
0141*060 'LOC-EXP-DATE'
01420001 '
0151*010 POLICY-NUMBER F0
0151*020 COVERAGE-CODE
0151*030 EFFECTIVE-DATE FD0151*035 RISK-LEVEL $The floating field
0152*040 LOCATION-ID.INDEX
0152*050 LOC-EFF-DATE.INDEX FD
0152*060 LOC-EXP-DATE.INDEX FD
017 RELS 1
017100 RELS 2
017 INDEX + 1 INDEX
017 IF INDEX LE LOCATION-NUMBER 100
017 MOVE 1 TO INDEX
017 DROP

Result

<table>
<thead>
<tr>
<th>POLICY NUMBER</th>
<th>COVERAGE</th>
<th>EFFECTIVE DATE</th>
<th>RISK LEVEL</th>
<th>LOCATION ID</th>
<th>LOC-EFF-DATE</th>
<th>LOC-EXP-DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111</td>
<td>010</td>
<td>mm/dd/yy</td>
<td>1</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>222222</td>
<td>011</td>
<td>mm/dd/yy</td>
<td>5</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>333333</td>
<td>005</td>
<td>mm/dd/yy</td>
<td>10</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>4444444</td>
<td>002</td>
<td>mm/dd/yy</td>
<td>1</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>

Note: The report would show actual dates in the format shown.

Obtaining Accumulated Totals

You can obtain the accumulated totals value for each occurrence of a subscripted field.

How to Do It

1. Name the occurrences of the subscripted field on type 5 parameters.

2. Print the specific totals values with type 6 parameters.

15-Jan-2018  118/225
In the example below, AMT.5 is defined as a work field consisting of five repeating fields. The type 5 parameters extract the value of each occurrence of AMT.1 and AMT.2. The type 6 parameters return the sum of the occurrences of AMT.1 and the sum of the occurrences of AMT.2:

010 AMT.5
0151*005 AMT.1
0151*010 AMT.2
0161*005 AMT.1
0161*010 AMT.2

Demonstration 22

Objective

This report lists the quarterly gain or loss for a series of stores by retrieving the accumulated totals value for each occurrence of the subscripted work field GAIN-OR-LOSS.4.

Procedure

- Two repeating input fields are defined:
  - INCOME and EXPEND are the group names.
  - QTR and AMT are the field names.
- Gain or loss statistics for each fiscal quarter are computed from the elements (QTR - AMT), which are placed in the work field GAIN-OR-LOSS.4.
- Individual occurrences of the GAIN-OR-LOSS work field are retrieved by type 5 parameters.
- References to each occurrence of the GAIN-OR-LOSS work field on type 6 lines obtain the accumulated totals for each quarter.

Complete Code

col. 2

IN 80 F 320 PS(TAPE)
REC STORE 1 3 2
REC INCOME 4 GROUP AA 8.4
REC QTR 1 8 2 DP=2 ELMNT AA
REC EXPEND 36 GROUP BB 8.4
REC AMT 1 8 2 DP=2 ELMNT BB
010 GAIN-OR-LOSS.4 DP=2
010 INDEX 1
013 PROFIT/LOSS REPORT yyyy
0151*010 STORE HH 'STORE'
0151*020 GAIN-OR-LOSS.1 SZ=11 F3 HH 'FIRST QUARTER'
0151*030 GAIN-OR-LOSS.2 SZ=11 F3 HH 'SECOND QUARTER'
0151*040 GAIN-OR-LOSS.3 SZ=11 F3 HH 'THIRD QUARTER'
0151*050 GAIN-OR-LOSS.4 SZ=11 F3 HH 'FOURTH QUARTER'
0151*060 GAIN-OR-LOSS.5 $Quarterly totals
0161*020 GAIN-OR-LOSS.1 SZ=11 F3
0161*030 GAIN-OR-LOSS.2 SZ=11 F3
0161*040 GAIN-OR-LOSS.3 SZ=11 F3
0161*050 GAIN-OR-LOSS.4 SZ=11 F3
017100 COMPUTE QTR.INDEX - AMT.INDEX GAIN-OR-LOSS.INDEX
017 INDEX + 1 INDEX
017 IF INDEX LE 4 100
017 MOVE 1 TO INDEX
017 TAKE
Obtaining Specific Field Values

You can obtain the current value of a subscripted numeric work field in output processing when the work field is specified on type 6 or 8 parameters only.

How to Do It

Code:

- The work field
- A type 6 or a type 8 parameter with the name of the work field and the subscript indicating the occurrence to be retrieved

In the following example, AMT.1 and AMT.2 on the type 6 parameter return the current values of the first and second work field occurrences:

```
010 AMT.5
0161*005 AMT.1
0161*010 AMT.2
```

Demonstration 23

Objective

The following report lists the quarterly gain or loss for each of four stores. The annual gain or loss, by store, is printed across the bottom of the report.

Procedure

- **GAIN-OR-LOSS.4** is a subscripted work field that holds the quarterly income computation for each store.
- **STORE-TOTAL** is a subscripted work field that holds the annual income of each store.
- Each occurrence of STORE-TOTAL is printed out by references on type 6 parameters.

Complete Code
### Result

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>PROFIT/LOSS REPORT</th>
<th>yyyy</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORE</td>
<td>QUARTERLY</td>
<td>GAIN/LOSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>19,900.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10,788.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>53,899.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>19,895.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,388.41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>99.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2,895.76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5,099.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>34,788.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>17,890.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>19,895.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>44,900.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>10,211.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>123,899.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>70,184.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>264,835.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE TOTAL SUMMARY:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Obtaining Sort-key Values

You can use one element of a subscripted field as a sort key. The value of the sort key associated with the last detail line processed before the control break is returned when type 6 or type 8 parameters reference the field.

How to Do It

Code:

1. The subscripted field on a **SORT** parameter

2. The subscripted field on a **type 6** or **type 8** parameter

The following example uses a subscripted work field, AMT.5, on the SORT parameter. The type 6 reference to the field returns the current value of AMT.5:

```
010 AMT.5 01SORT AMT.5 0151*010 ' ' 0161*010 AMT.5
```

Demonstration 24

**Objective**

This report prints the third quarter profits for each of six stores.

**Procedure**

- The work field **GAIN-OR-LOSS.4** holds the results of quarterly income calculations.
- An occurrence of the subscripted work field **GAIN-OR-LOSS.3** is used as the sort key. The value of that particular field is written to the extract file when a control break occurs.
- A **type 6** parameter references GAIN-OR-LOSS.3 to retrieve the subscripted field value from the extract file.

**Complete Code**

```
col. 2
IN 80 F 320 PS(TAPE)
REC STORE 1 3 2  
REC INCOME 4  
REC QTR 1 8 2 DP=2  
REC EXPEND 36  
REC AMT 1 8 2 DP=2  
01OUT T
010 GAIN-OR-LOSS.4 DP=2  $Four quarters
010 INDEX 1
01SORT STORE 0 GAIN-OR-LOSS.3  $Third occurrence in work field
013 THIRD QUARTER EARNINGS yyyy
0151*010 ' '  $One type 5 line is always required
0161*010 STORE HH 'STORE'
* $  
0161*020 GAIN-OR-LOSS.3 SZ=11 F3 HH 'THIRD QUARTER PROFIT'
```
1. **COMPUTE QTR.INDEX - AMT.INDEX GAIN-OR-LOSS.INDEX**
2. **IF INDEX EQ 4 200**
3. **INDEX + 1 INDEX**
4. **IF INDEX LE 4 100**
5. **MOVE 1 TO INDEX**
6. **IF LEVL EQ 1 100**
7. **IF LEVL EQ 2 200**
8. **TAKE 1**
9. **DROP**

**Note:** Third Quarter Earnings would be the year in yyyy format.

**Result**

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>THIRD QUARTER EARNINGS STORE yyyy mm/dd/yyPAGE</th>
<th>1 THIRD QUARTER PROFIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The report would show the actual dates in the format shown.

**File Matching**

Two or more *presorted* conventional files having at least one common data item between them can be compared. When a match occurs, the matching records are brought into the input buffer as a single record.

**How to Code Match-file Runs**

**Step 1**

Sort all files in ascending match-key sequence.

See the section *Additional Standard File Facilities (see page 164)* for creating sorted files.

**Step 2**

Code:

1. One **INPUT** parameter with the **MK=** option for each file to be matched.
   Match keys must be either all numeric or all alphanumeric. Numeric types can be mixed:
1. Specify the DD= option on all or none of the IN parameters.

2. Specify MB=KEEP if a record is to be matched against multiple occurrences of another file.

3. Correlate the coding sequence of the CA Culprit file definitions with the sequence of JCL statements that define the files (SYS010, SYS011, and additional increments, as needed).

2. REC parameters immediately following the associated INPUT parameter:

   - Use unique names for all fields.

   - Use start positions that are relative to the beginning of the record (not to the beginning of the input buffer).

3. Optional SELECT/BYPASS parameters:

   - After the REC parameters of the specific file to be processed.

   - With the BUFFER option immediately after the last input file definition if the processing applies to all files after matching takes place.

For more information, see the following topics:

- How File Matching Works (see page 124)
- Matching Single-Occurrence Files (see page 128)
- Matching Multiple Transactions with a Single-Entry Master (see page 129)
- Listing Accounts Without Transactions (see page 133)
- Listing Transactions not on the Master File (see page 135)
- Using the Match-file Facility for Table Initialization (see page 137)
- Using Files Defined to the Data Dictionary (see page 140)
- Creating Unique Match-key Names (see page 140)
- Qualified Fields (see page 141)

How File Matching Works

When CA Culprit encounters more than one INPUT parameter, it recognizes a match run and proceeds as follows:

1. Searches the first record on all files for a match:
   - If a SELECT statement is used on any file, the SELECT logic is performed before any match-file processing.
   - If a SELECT BUFFER statement is used, the SELECT logic is performed after the match processing and before type 7 logic is completed.

2. Searches the remaining file for a match:
   - If a match among all files is found, the matching records are delivered to the input buffer and type 7 processing is performed.
   - If a match is not found,
a. The record with the lowest key value is moved into the input buffer.

b. The remainder of the input buffer is nulled out by setting numeric key fields to zeros, alphanumeric key fields to spaces, and the rest of the record to spaces.

c. Type 7 processing is performed.

The figure below shows the contents of the input buffer during a simple match-file run:

**Buffer Contents During a Match-file Run**

<table>
<thead>
<tr>
<th>PHYSICAL RECORD SEQUENCE</th>
<th>INPUT BUFFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE 1 (SYS010)</td>
<td>FILE 2 (SYS011)</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>* B</td>
</tr>
<tr>
<td>D1</td>
<td>C</td>
</tr>
<tr>
<td>D2</td>
<td>* E1</td>
</tr>
<tr>
<td>E2</td>
<td>* E2</td>
</tr>
<tr>
<td>G</td>
<td>* G1</td>
</tr>
<tr>
<td>G2</td>
<td>* G2</td>
</tr>
<tr>
<td>H</td>
<td>* H</td>
</tr>
</tbody>
</table>

Note: D1, D2, E1, and so on, represent records with duplicate key values. An asterisk (*) indicates a null section of the buffer.

**Keeping a Record in the Buffer**

When a record needs to be processed against more than one record with a matching key value, the MB=KEEP option of the INPUT parameter is used. MB=KEEP is most commonly used on one file only.

The contents of the input buffer when the MB=KEEP option is used is shown in the figure below:

**Buffer Contents with the KEEP Option**

<table>
<thead>
<tr>
<th>PHYSICAL RECORD SEQUENCE</th>
<th>INPUT BUFFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE 1 (SYS010)</td>
<td>FILE 2 (SYS011)</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>* B</td>
</tr>
</tbody>
</table>
Checking the File Status

When CA Culprit reformat the input buffer for a match file run, it adjusts the starting position of the input fields by inserting two additional bytes for use in checking the status of the files:

- The first byte is an input file status byte specific to each file in the run.
- The second byte, called M*ID, is a composite input file status indicator consisting of a binary combination of the file-specific status bytes for all files in the run.

If the return value of M*ID is not 0, test the file-specific status bit to locate the file or files containing an error condition:

1. Define the file-specific status bit on a REC parameter:

   If binary: REC FILE4-STATUS 0 1 1
   
   If bit: REC FILE4-STATUS 0 81 5
   (Bit format uses a 2-digit length specification, indicating FILE4-STATUS is a single bit that occurs in the eighth bit position of start position 0.)

2. Test the value of the bit in type 7 logic for a non-zero (0) value.

The bit layout within the M*ID and the file-specific status bit fields are shown in the table below.

### The Bit Layout Within M*ID

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Condition</th>
<th>Decimal value of M*ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>File(s) out of sequence</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Duplicate key value</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>End of file(s) value</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
<td>0</td>
</tr>
</tbody>
</table>

Demonstration

---

Note: D1, D2, E1, and E2 represent records with duplicate key values. An asterisk (*) indicates a null section of the buffer.
The following examples list the values of the file-specific status byte and the M*ID under error conditions:

- The first example attempts a match-file run with an unsorted transaction file.
- The second example attempts a match-file run with an unsorted transaction file and an unsorted master file.

**Complete Code**

```plaintext
IN 80 MK=M-ACCOUNT
REC M-FILE-STATUS 0 1 1 'M-FILE' 'STATUS'
REC M-ACCOUNT 1 5 'ACCOUNT'
REC M-NAME 620

IN 80 MK=T-ACCOUNT
REC T-FILE-STATUS 0 1 1 'T-FILE' 'STATUS'
REC T-ACCOUNT 1 5 'ACCOUNT'
REC T-PAYMENT 6 8 2 DP=2 'PAYMENT'

02OUT D
HH 'M*ID'
HR
0251*005 M*ID
0251*010 T-FILE-STATUS
0251

*015 M-FILE-STATUS
0251*020 M-ACCOUNT

**Unsorted Transaction File**

```

```

**Unsorted Master and Transaction Files**

```

```

```
Matching Single-Occurrence Files

Demonstration 25

Objective

This report contains account information from both a master file and a matching transaction file. Each file has a match key occurrence.

Procedure

The contents of the input buffer, represented by account number, show how the files are matched:

<table>
<thead>
<tr>
<th>Master file</th>
<th>Transaction file</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SYS010)</td>
<td>(SYS011)</td>
</tr>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>21056</td>
<td>21056</td>
</tr>
<tr>
<td>29557</td>
<td>29557</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>33470</td>
<td>33470</td>
</tr>
<tr>
<td>69876</td>
<td>69876</td>
</tr>
<tr>
<td>99083</td>
<td>99083</td>
</tr>
</tbody>
</table>

Complete Code

col. 2

IN 80 F 320 PS(TAPE) MK=M-ACCOUNT  $Master file
REC FILE-STATUS 0 1 1  $Input file status
REC M-ACCOUNT 1 5 'ACCOUNT'
REC M-NAME 6 20 'NAME'
REC M-ADDRESS 26 37 'ADDRESS'
REC M-ZIP 63 5

IN 80 F 320 PS(TAPE) MK=T-ACCOUNT  $Transaction file
REC T-ACCOUNT 1 5 'ACCOUNT'
REC T-PAYMENT 6 8 2 DP=2 'PAYMENT'
REC T-DATE 14 6 2 'DATE'
02OUT D
023 JANUARY PAYMENTS RECEIVED
0251+010 M-ACCOUNT HR
0251+020 M-NAME HR
0251+030 M-ADDRESS HR
0251+035 M-ZIP
0251+040 T-PAYMENT SZ=10 HR
0251+050 T-DATE FD HR
027010 IF FILE-STATUS EQ 1 DROP $EOF test

Result
Matching Multiple Transactions with a Single-Entry Master

Demonstration (1) Dropping Unmatched Records

Objective

This report lists customer transactions and current balances by matching a multiple-occurrence transaction file against a single-entry master file. Unmatched records are dropped.

Procedure

- **MB=KEEP** retains a master record in the input buffer.
- The **BYPASS** parameter eliminates unwanted master file records.
- **Type 7** logic drops all unmatched records.

The contents of the input buffer, represented by account number, show how the files are matched:
<table>
<thead>
<tr>
<th>Master file</th>
<th>Transaction file</th>
</tr>
</thead>
<tbody>
<tr>
<td>23055</td>
<td>23055</td>
</tr>
<tr>
<td></td>
<td>23055</td>
</tr>
<tr>
<td></td>
<td>23055</td>
</tr>
<tr>
<td>27777</td>
<td>29557</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>31113</td>
<td>31113</td>
</tr>
<tr>
<td>31275</td>
<td>31275</td>
</tr>
<tr>
<td>32115</td>
<td>33470</td>
</tr>
<tr>
<td>33470</td>
<td>33470</td>
</tr>
<tr>
<td>34440</td>
<td>34440</td>
</tr>
<tr>
<td>36682</td>
<td>36682</td>
</tr>
<tr>
<td>69876</td>
<td>69876</td>
</tr>
<tr>
<td>99083</td>
<td>99083</td>
</tr>
</tbody>
</table>

**Complete Code**

```
col. 2
IN  80 MK=M-ACCOUNT MB=KEEP $Keep the master record
REC M-ACCOUNT 1 5 'ACCOUNT'
REC M-BRANCH 6 2 'BRANCH'
REC M-NAME 19 20 'NAME'
BYP M-BRANCH NE ('32' '35' '45' '46') $Select records
IN 80 MK=T-ACCOUNT
REC T-ACCOUNT 1 5 'ACCOUNT'
REC T-TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC T-TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC T-DATE 18 6 2 'DATE'
REC T-BRANCH 24 2 'BRANCH'
REC T-NAME 26 20 'NAME'
80OUT D
80SORT T-BRANCH - T-ACCOUNT T-DATE
803 MASTER FILE MATCHED WITH MULTIPLE TRANSACTIONS
8051+010 M-BRANCH HR
8051+020 T-ACCOUNT HR
8051+030 M-NAME HR
8051+050 T-TRANS-AMT HR
```
Demonstration (2) Selecting Matching Account Numbers

This is the same report presented in Demonstration (1). Only this time records with matching account numbers are selected.

Procedure

- MB=KEEP retains a master record in the input buffer.
- The SELECT parameter selects master file records.
- The SELECT parameter with the BUFFER option selects only records that have matching account numbers.

Complete Code

col. 2

IN 80 F 320 PS(TAPE) MK=M-ACCOUNT MB=KEEP $Keep the master record
REC M-ACCOUNT 1 5 'ACCOUNT'
REC M-BRANCH 6 2 'BRANCH'

Note: The report would show actual dates in the format shown.
CA IDMS - 19.0

REC M-NAME 19 20 ‘NAME’
SEL M-BRANCH EQ (‘32’ ‘35’ ‘45’ ‘46’) $Select records

IN 80 F 320 PS(TAPE) MK=T-ACCOUNT
REC T-ACCOUNT 1 5 ‘ACCOUNT’
REC T-TRANS-IND 6 1 ‘DEPOSIT/’ ‘WITHDRAWAL’
REC T-TRANS-AMT 7 11 2 DP=2 ‘AMOUNT OF’ ‘TRANSACTION’
REC T-DATE 18 6 2 ‘DATE’
REC T-BRANCH 24 2 ‘BRANCH’
REC T-NAME 26 20 ‘NAME’
SEL BUFFER WHEN M-ACCOUNT EQ T-ACCOUNT
80OUT D
80SORT T-BRANCH - T-ACCOUNT T-DATE
803 MASTER FILE MATCHED WITH MULTIPLE TRANSACTIONS
8051+010 M-BRANCH HR
8051+020 T-ACCOUNT HR
8051+030 M-NAME HR
8051+050 T-TRANS-AMT HR
8051+060 T-TRANS-IND HR
8051+070 T-DATE FD HR

Result

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>ACCOUNT</th>
<th>NAME</th>
<th>DEPOSIT/ WITHDRAWAL</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>10,000.00</td>
<td>D</td>
</tr>
<tr>
<td>32</td>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>10,000.00</td>
<td>D</td>
</tr>
<tr>
<td>32</td>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>100,000.00</td>
<td>D</td>
</tr>
<tr>
<td>32</td>
<td>16070</td>
<td>ARTHUR LINK</td>
<td>1,000.00</td>
<td>D</td>
</tr>
<tr>
<td>32</td>
<td>19235</td>
<td>GARY NOBLES</td>
<td>80.00</td>
<td>W</td>
</tr>
<tr>
<td>32</td>
<td>21056</td>
<td>AMOS JOHNSON</td>
<td>1.02</td>
<td>W</td>
</tr>
<tr>
<td>35</td>
<td>23055</td>
<td>JACK JACKSON</td>
<td>500.00</td>
<td>W</td>
</tr>
<tr>
<td>35</td>
<td>23055</td>
<td>JACK JACKSON</td>
<td>50,000.00</td>
<td>D</td>
</tr>
<tr>
<td>35</td>
<td>23055</td>
<td>JACK JACKSON</td>
<td>30.00</td>
<td>D</td>
</tr>
<tr>
<td>35</td>
<td>29557</td>
<td>IRWIN TRIMBLE</td>
<td>500,000.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>33470</td>
<td>VICTORIA DAY</td>
<td>7,590,000.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>33470</td>
<td>VICTORIA DAY</td>
<td>1.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>34448</td>
<td>HELEN WIGHT</td>
<td>195.00</td>
<td>W</td>
</tr>
<tr>
<td>45</td>
<td>34448</td>
<td>HELEN WIGHT</td>
<td>2,000.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>34448</td>
<td>HELEN WIGHT</td>
<td>95.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>34448</td>
<td>HELEN WIGHT</td>
<td>100.00</td>
<td>D</td>
</tr>
<tr>
<td>45</td>
<td>34682</td>
<td>JEAN WREN</td>
<td>4,000.78</td>
<td>W</td>
</tr>
<tr>
<td>46</td>
<td>69876</td>
<td>BRUCE THORPE</td>
<td>9,000,001.15</td>
<td>D</td>
</tr>
<tr>
<td>46</td>
<td>69876</td>
<td>BRUCE THORPE</td>
<td>637.55</td>
<td>D</td>
</tr>
<tr>
<td>46</td>
<td>99083</td>
<td>HELEN SANTOVEC</td>
<td>50,001.30</td>
<td>D</td>
</tr>
<tr>
<td>46</td>
<td>99083</td>
<td>HELEN SANTOVEC</td>
<td>6,468.52</td>
<td>D</td>
</tr>
</tbody>
</table>

Note: The report would show actual dates in the format shown.
Listing Accounts Without Transactions

Demonstration 26

Objective

This report lists accounts that do not have transactions. The match-file run matches a multiple-transaction file against a single-occurrence master file.

Procedure

- **MB=KEEP** retains a master file in the input buffer.
- A **SELECT BUFFER** statement selects those accounts without transactions.

⚠️ **Note:** Type 7 logic (IF M-ACCOUNT LE T-ACCOUNT DROP) can be used instead of the SELECT BUFFER statement.

The contents of the input buffer:

<table>
<thead>
<tr>
<th>Master file (SYS010)</th>
<th>Transaction file (SYS011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15999</td>
<td></td>
</tr>
<tr>
<td>16070</td>
<td>16070</td>
</tr>
<tr>
<td>19235</td>
<td>19235</td>
</tr>
<tr>
<td>21056</td>
<td>21056</td>
</tr>
<tr>
<td>23055</td>
<td>23055</td>
</tr>
<tr>
<td>23055</td>
<td>23055</td>
</tr>
<tr>
<td>23055</td>
<td>23055</td>
</tr>
<tr>
<td>27777</td>
<td></td>
</tr>
<tr>
<td>29557</td>
<td>29557</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>31113</td>
<td>31113</td>
</tr>
<tr>
<td>31113</td>
<td>31113</td>
</tr>
</tbody>
</table>
Master file | Transaction file
---|---
31275 | 31275
32115 | 32115
33470 | 33470
33470 | 33470
34440 | 34440
34440 | 34440
34440 | 34440
34440 | 34440
36682 | 36682
69876 | 69876
69876 | 69876
99083 | 99083
99083 | 99083

Complete Code

col. 2

IN 80 MK=M-ACCOUNT MB=KEEP $Keep master record
REC M-ACCOUNT  1   5       'ACCOUNT'
REC M-BRANCH    6   2       'BRANCH'
REC M-NAME      19  20      'NAME'

IN 80 MK=T-ACCOUNT
REC T-ACCOUNT   1   5       'ACCOUNT'
SEL BUFFER WHEN T-ACCOUNT EQ ' ' $Accounts without transactions
80OUT D
80SORT M-BRANCH -
803 LIST OF ACCOUNTS WITHOUT TRANSACTIONS
8051*010 M-BRANCH   HR
8051*020 M-ACCOUNT  HR
8051*030 M-NAME     HR

Result

<table>
<thead>
<tr>
<th>Branch</th>
<th>Account</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>15999</td>
<td>SHELLY BROWN</td>
</tr>
<tr>
<td>35</td>
<td>27777</td>
<td>ANDY PIGGOTT</td>
</tr>
<tr>
<td>40</td>
<td>32115</td>
<td>BOB DATO</td>
</tr>
</tbody>
</table>
Listing Transactions not on the Master File

Demonstration 27

Objective

This report lists transactions not on the master file. Multiple transactions are matched against a single-entry master file.

Procedure

- **MB=KEEP** retains a master file in the input buffer.
- A **SELECT BUFFER** statement selects transactions that cannot be matched with the master file.

⚠️ **Note:** Type 7 logic (IF M-ACCOUNT GE T-ACCOUNT DROP) can be used instead of the SELECT BUFFER statement.

The contents of the input buffer for the run (K denotes master records kept in the input buffer):

<table>
<thead>
<tr>
<th>Master file (SYS010)</th>
<th>Transaction file (SYS011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15060</td>
<td>15060</td>
</tr>
<tr>
<td>15999</td>
<td>15999</td>
</tr>
<tr>
<td>16070</td>
<td>16070</td>
</tr>
<tr>
<td>19235</td>
<td>19235</td>
</tr>
<tr>
<td>21056</td>
<td>21056</td>
</tr>
<tr>
<td>23055</td>
<td>23055</td>
</tr>
<tr>
<td>23055K</td>
<td>23055K</td>
</tr>
<tr>
<td>23055K</td>
<td>23055K</td>
</tr>
<tr>
<td>27777</td>
<td>27777</td>
</tr>
<tr>
<td>29557</td>
<td>29557</td>
</tr>
<tr>
<td>30115K</td>
<td>30115K</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>30115K</td>
<td>30115K</td>
</tr>
<tr>
<td>30115K</td>
<td>30115K</td>
</tr>
<tr>
<td>30115</td>
<td>30115</td>
</tr>
<tr>
<td>Master file</td>
<td>Transaction file</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
</tr>
<tr>
<td>31275</td>
<td>31275</td>
</tr>
<tr>
<td>32115</td>
<td>33470</td>
</tr>
<tr>
<td>33470</td>
<td>33470</td>
</tr>
<tr>
<td>33470K</td>
<td>33470</td>
</tr>
<tr>
<td>34440</td>
<td>34440</td>
</tr>
<tr>
<td>34440K</td>
<td>34440</td>
</tr>
<tr>
<td>34440K</td>
<td>34440</td>
</tr>
<tr>
<td>34440K</td>
<td>36682</td>
</tr>
<tr>
<td>69876</td>
<td>69876</td>
</tr>
<tr>
<td>69876K</td>
<td>99083</td>
</tr>
<tr>
<td>99083</td>
<td>99083</td>
</tr>
</tbody>
</table>

**Complete Code**

```plaintext
col. 2
IN 80 MK=M-ACCOUNT MB=KEEP REC M-ACCOUNT 1 5 'ACCOUNT'

IN 80 MK=T-ACCOUNT
REC T-ACCOUNT 1 5 'ACCOUNT'
REC T-TRANS-IND 6 1 'DEPOSIT' 'WITHDRAWAL'
REC T-TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC T-DATE 18 6 2 'DATE'
REC T-BRANCH 24 2 'BRANCH'
REC T-NAME 26 20 'NAME'

SEL BUFFER WHEN T-ACCOUNT GT M-ACCOUNT
80OUT D
80SOR T-BRANCH - T-DATE D T-ACCOUNT
803 LISTING OF TRANSACTIONS WITHOUT MASTERS
8051+010 T-BRANCH HR
8051+020 T-ACCOUNT HR
8051+030 T-NAME HR
8051+050 T-TRANS-AMT HR
8051+060 T-TRANS-IND HR
8051+070 T-DATE FD HR
```

**Result**

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>ACCOUNT</th>
<th>NAME</th>
<th>AMOUNT OF TRANSACTION</th>
<th>DEPOSIT/</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>100,998.11</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>32</td>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>10,999.11</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>40</td>
<td>31113</td>
<td>MARY CATREY</td>
<td>36.80</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>40</td>
<td>31113</td>
<td>MARY CATREY</td>
<td>36.80</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>45</td>
<td>36682</td>
<td>JEAN WREN</td>
<td>4,000.78</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>
Using the Match-file Facility for Table Initialization

You can use match keys to control the order in which files are read in a match run. If you do not specify match keys for a file, CA Culprit assumes low key values for the missing match keys on that file and reads it first.

How to Do It

1. Omit the MK= specification on the IN parameter for the file you want read first.
2. Code the MK= specification on the IN parameter for subsequent files.

What You Can Do

You can use match keys to control the order in which files are read in a match run. If you do not specify match keys for a file, CA Culprit assumes low key values for the missing match keys on that file and reads it first.

How to Do It

1. Omit the MK= specification on the IN parameter for the file you want read first.
2. Code the MK= specification on the IN parameter for subsequent files.

Demonstration

Objective

This example produces two reports:

- An interest rate table from the first file processed
- A listing of the first month's interest for new accounts, using the interest rate table and the input data from the second file

Procedure

- No match key is used for the interest rate file.
A match key on ACCOUNT-NO is used for the second file.

The contents of the input buffer for this run:

<table>
<thead>
<tr>
<th>File 1 (SYS010)</th>
<th>File 2 (SYS011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0500 .0525 .0550</td>
<td>rule=no.</td>
</tr>
<tr>
<td>.0525 .0550 .0575</td>
<td>rule=no.</td>
</tr>
<tr>
<td>.0550 .0575 .0600</td>
<td>rule=no.</td>
</tr>
<tr>
<td>.0550 .0575 .0625</td>
<td>rule=no.</td>
</tr>
</tbody>
</table>

| 15060 | 15999 |
| 16070 | 19235 |
| 21056 | 23055 |
| 27777 | 29557 |

Complete Code

col. 2

IN 80 $Read first, no match key
REC FILE-TEST 0 1 1 REC INTEREST-RATE-DATA 1 GROUP AA 4.3
REC INTEREST-RATE 1 4 2 DP=4 ELMNT AA

IN 150 MK=ACCOUNT-NO $Read second, one match key
REC ACCOUNT-NO 1 5 'ACCOUNT' 'NUMBER'
REC ACCT-TYPE 6 2 2 'ACCOUNT' 'TYPE'
REC OPEN-BAL 8 11 2 DP=2 'OPENING' 'BALANCE'
REC NAME 19 20 'NAME'
REC DATE-OPENED 39 6 2 'DATE' 'OPENED'
GW0 RATE.12 DP=4 COUNT ZERO
00OUT 65 D

003 INTEREST RATE TABLE $First report
0051*010 INTEREST-RATE.1 HH 'RATE CODE 1' '(ACCOUNT TYPE 1)'
0051*020 INTEREST-RATE.2 HH 'RATE CODE 2' '(ACCOUNT TYPE 2)'
0051*030 INTEREST-RATE.3 HH 'RATE CODE 3' '(ACCOUNT TYPE 3)'
92SORT ACCT-TYPE 0 OPEN-BAL D
92OUT 120 D
920 NTH
920 FIRST-MO-INT DP=3

923 FIRST MONTH'S INTEREST $Second report
9241+010 '
9251*010 ACCOUNT-NO HR
9251*020 NAME HR
9251*030 DATE-OPENED FD HR
9251*040 ACCT-TYPE HR
9251*050 RATE.NTH SZ=4 HH 'INTEREST' 'RATE'
9251*060 OPEN-BAL HR
9251*070 FIRST-MO-INT SZ=8 HH 'INTEREST' '(FIRST' 'MONTH)'
007010 IF FILE-TEST EQ 1 DROP
007 COUNT + 1 COUNT
007 MOVE INTEREST-RATE.1 TO RATE.COUNT
007 COUNT + 1 COUNT
007 MOVE INTEREST-RATE.2 TO RATE.COUNT
007 COUNT + 1 COUNT
007 MOVE INTEREST-RATE.3 TO RATE.COUNT
007  IF COUNT GT  040
007  TAKE
007040  ZERO DIVIDE ZERO ZERO  $Forced buffer dump if more than 12 entries
007  STOP-RUN  $Error, stop processing
927  IF FILE-TEST NE 1 DROP
927  IF DATE-OPENED GE mmddyy 100
927  IF DATE-OPENED GE mmddyy 110
927  IF DATE-OPENED GE mmddyy 120
927  IF DATE-OPENED GE mmddyy 130
927  DROP
927100  ACCT-TYPE + 9 NTH
927  B 150
927110  ACCT-TYPE + 6 NTH
927  B 150
927120  ACCT-TYPE + 3 NTH
927  B 150
927130  ACCT-TYPE + 0 NTH
927150  COMPUTE (OPEN-BAL X RATE.NTH) / 12 FIRST-MO-INT

Note: For 'IF DATE-OPENED GE', the code would be actual dates in the format shown.

Result

<table>
<thead>
<tr>
<th>REPORT NO. 08</th>
<th>INTEREST RATE TABLE</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATE CODE 1</td>
<td>RATE CODE 2</td>
<td>RATE CODE 3</td>
<td></td>
</tr>
<tr>
<td>(ACCOUNT TYPE 1)</td>
<td>(ACCOUNT TYPE 2)</td>
<td>(ACCOUNT TYPE 3)</td>
<td></td>
</tr>
<tr>
<td>.0500</td>
<td>.0525</td>
<td>.0550</td>
<td></td>
</tr>
<tr>
<td>.0525</td>
<td>.0550</td>
<td>.0575</td>
<td></td>
</tr>
<tr>
<td>.0550</td>
<td>.0575</td>
<td>.0600</td>
<td></td>
</tr>
<tr>
<td>.0550</td>
<td>.0575</td>
<td>.0625</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REPORT NO. 92</th>
<th>FIRST MONTH’S INTEREST</th>
<th>mm/dd/yy PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT NUMBER</td>
<td>NAME</td>
<td>DATE OPENED</td>
<td>ACCOUNT TYPE</td>
</tr>
<tr>
<td>33470</td>
<td>VICTORIA DAY</td>
<td>mm/dd/yy</td>
<td>1</td>
</tr>
<tr>
<td>16070</td>
<td>ARTHUR LINK</td>
<td>mm/dd/yy</td>
<td>1</td>
</tr>
<tr>
<td>99883</td>
<td>HELEN SANTOVEC</td>
<td>mm/dd/yy</td>
<td>1</td>
</tr>
<tr>
<td>69876</td>
<td>BRUCE THORPE</td>
<td>mm/dd/yy</td>
<td>2</td>
</tr>
<tr>
<td>19235</td>
<td>GARY NOBLES</td>
<td>mm/dd/yy</td>
<td>2</td>
</tr>
<tr>
<td>34448</td>
<td>HELEN WRIGHT</td>
<td>mm/dd/yy</td>
<td>2</td>
</tr>
<tr>
<td>36682</td>
<td>JEAN WREN</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>15999</td>
<td>SHELLY BROWN</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>30115</td>
<td>IRIA DOONES</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>31113</td>
<td>MARY CATHEY</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>15868</td>
<td>SHARON ARMSTRONG</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>21856</td>
<td>ARON JOHNSON</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>23955</td>
<td>JACK JACKSON</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>27777</td>
<td>ANY PIKGOTT</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>29557</td>
<td>IRWIN TRIMBLE</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>31275</td>
<td>FOR ANOTHER</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
<tr>
<td>32115</td>
<td>BOB DATO</td>
<td>mm/dd/yy</td>
<td>3</td>
</tr>
</tbody>
</table>

Note: The report would show actual dates in the format shown.
Using Files Defined to the Data Dictionary

What You Can Do 31

During match file runs, you can reference files defined to the data dictionary.

How to Do It

Code:

1. A DATABASE parameter with the DICTNAME= option if an alternate dictionary is used.
2. An INPUT parameter with the FN= option to specify the file name.
3. INPUT parameter options and REC parameters if redefinition or supplementary definitions are needed.

Demonstration 29

Objective

This report matches a student master file (STUDENT-MAST), which is defined to the data dictionary, with a VSAM file containing course descriptions and prerequisite courses.

Procedure

- A DATABASE parameter with the DICTNAME= option specifies the alternate dictionary DOCUDICT.
- The FN= option on the IN parameter specifies the STUDENT-MAST file.

Partial Code

```
col. 2

DATABASE DICTNAME=DOCUDICT IN 257 FN=STUDENT-MAST MK=COURSE-CODE
IN 96 96 VS MK=COURSE-NUM
REC COURSE-NUm 1 3
REC DESC 4 30
REC PREQ 34 GROUP AA 3.4
REC COURSE 1 3 ELMNT AA
```

Creating Unique Match-key Names

You can assign unique names to reference entities in the data dictionary. You may use either the primary (data dictionary) name or the assigned name. Just be consistent.
How to Do It

Use a **REC** parameter to assign a unique name to a field.

What You Can Do 32

Demonstration 30

**Objective**

This example matches two files that are defined to the data dictionary. The match key for both files is **PART-NUMBER**. **PART-NUMBER** in the **PARTS-INVENTORY** file is assigned **STOCK-REF-NUM** as a unique name.

**Procedure**

- A **DD=** specification on the **IN** parameter provides alternative file assignments for each file.
- A **REC** parameter renames the **PART-NUMBER** field of the **PARTS-INVENTORY** file.
- An **MK=** specification of the new name of the **PART-NUMBER** (**STOCK-REF-NUM**) is the match key for the **PARTS-INVENTORY** file.

**Partial Code**

```
col. 2
DATABASE DICTNAME=DOCUDICT
IN 130 DD=SYS011 FN=PARTS-INVENTORY MK=STOCK-REF-NUM* WAREHOUSE-CODE BIN-POINTER
   REC STOCK-REF-NUM 1 10 IN 170 DD=SYS012 FN=PART-DESC MK=PART-NUMBER WAREHOUSE-ID BIN-LOCATION
```

**Qualified Fields**

You can associate one record name in the data dictionary with fields that are specified on two or more **INPUT** parameters.

**How to Do It**

Unique field names can be provided in a particular CA Culprit run by coding:

- The **FN=** option of the **INPUT** parameter with the data dictionary file name
- The **MK=** option of the **INPUT** parameter with the field name followed by the record name known to the data dictionary, a comma, and a level number enclosed in parentheses
- SYS010 in the JCL for the first file (level 1); SYS011 in the JCL for the second file (level 2), and so on

What You Can Do 33

Demonstration 31

Objective

This example matches three files that are defined in the data dictionary. Two files have MON-TRAN-FILE as a file name and TRAN-CODE as a key field name. All three files have ACCT-NO as a key field.

Procedure

The code uses level numbers to distinguish between field references (ACCOUNT,1, ACCOUNT,2, TRANSACTION,1, and TRANSACTION,2).

Partial Code

```
col. 2

DATABASE DICT NAME=DOCUDICT

IN 100 FN=MON-TRAN-FILE MK=ACCT-NO(ACCOUNT,1) TRAN-CODE
  *(TRANSACTION,1)

IN 100 FN=MON-TRAN-FILE MK=ACCT-NO(ACCOUNT,2) TRAN-CODE
  *(TRANSACTION,2)

IN 150 FN=CUST-FILE MK=ACCT-NO(ACC-REC) MB=KEEP
```

Using and Modifying Copied Code

Frequently used sections of CA Culprit code can be stored for use by several reports or users. Use of stored parameters helps to establish standard file definitions, procedures, and reports. For example, you can store INPUT and REC parameters that define a shared input file, type 7 or 8 procedural code that performs an operation needed by several reports, or a set of commonly used extract functions.

Parameters to be inserted are maintained in card-image format and in data dictionaries, partitioned data sets (z/OS), source statement libraries (z/VSE), AllFusion CA-Panvalet libraries, and AllFusion CA-Librarian libraries. Stored code can be copied and modified to meet the requirements of a particular report by using one of the copied code parameters:

- USE
- =COPY
- =MACRO

For more information see the following topics:
- Copying Stored Code (USE Parameter) (see page 143)
Assigning Values to Symbolic Fields (USE Parameter) (see page 143)
Assigning Default Values to Symbolic Parameters (USE Parameter) (see page 146)
Nesting the USE Parameter (see page 148)
Modifying Code (USE Parameter) (see page 149)
Copying and Modifying Code (=COPY Parameter) (see page 154)
Copying and Modifying Code (=MACRO Parameter) (see page 156)
Listing the Contents of a Data File (see page 160)

Copying Stored Code (USE Parameter)

The USE parameter allows you to copy code that is stored on 80-byte records.

How to Do It

Code a basic USE parameter by entering the following in the job stream:

1. The keyword USE

2. The name of the file or module that contains the CA Culprit source code:
   
   USE CULCODE
   /*

Assigning Values to Symbolic Fields (USE Parameter)

You can use symbolic fields on the parameters in your stored or inline CA Culprit code for values that are likely to change.

How to Do It

Code:

1. Source code that incorporates symbolic parameter references:

   010  START 661

2. A standard USE parameter that names the source code to be copied:

   USE CULCODE
   /*

3. The WITH VALUES keyword phrase:

   USE CULCODE
   WITH VALUES . . .
   /*
4. Keyword arguments (data values) or keyword expressions (labels), as needed, on the same line as WITH VALUES. The argument sequence must correspond to the parameter sequence:

```
USE CULCODE
WITH VALUES (mmddyy,MONTHLY REPORT)/*
```

**Note:** The value would show the actual date in the format shown.

Demonstration 32

**Objective**

This is a monthly report of branch office transactions created from code stored as RPT624. The report title and the dates printed in the heading of the report change each time the report code is run.

**Procedure**

- The **USE** parameter copies stored code (RPT624) containing symbolic parameters for values likely to change.
- The **WITH VALUES** clause provides arguments that are substituted for the symbolic parameters.

**Complete Code**

Below follows the job stream and the stored code:

**The Job Stream**

```
USE RPT624
WITH VALUES (mmddyy,mmddyy,mmddyy,MONTHLY REPORT)
/*
```

**Note:** The value would show actual dates in the format shown.

**The Stored Code**

```
col. 2

IN 80 F
REC ACCOUNT 1 5 'ACCOUNT'
REC TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC TRANS-DATE 18 6 2 'DATE'
REC BRANCH 24 2 'BRANCH'
REC NAME 26 20 'NAME'
800OUT 80
800SORT 80 ACCOUNT 1 BRANCH NAME RPT-DAY START END
800 START $61 $First date (mmddyy)
800 END $62 $Second date (mmddyy)
800 RPT-DAY $63 $Third date (mmddyy)
803 $64 $Title (MONTHLY REPORT)
80410010 NAME
80410030 ACCOUNT
```
Note: The first, second and third dates would be the actual dates in the format shown.

The sequential parameter listing, which is produced in a CA Culprit run, shows the substitutions made for the symbolic parameters.

**Sequential Parameter Listing**

<table>
<thead>
<tr>
<th>mm/dd/yy</th>
<th>SEQUENTIAL PARAMETER LISTING</th>
<th>volser Vnn.n</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 ** SYSIN **</td>
<td>USE RPT624</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>01 RPT624</td>
<td>WITH VALUES (mmddyy,mmddyy,mmddyy,MONTHLY REPORT)</td>
<td>IN 80 F</td>
<td></td>
</tr>
<tr>
<td>C200138 INSTALLATION SECURITY OPTION IS NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC ACCOUNT 1 5</td>
<td>'ACCOUNT'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC TRANS-IND 6 1</td>
<td>'DEPOSIT/' 'WITHDRAWAL'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC TRANS-AMT 7 11 2 DP=2</td>
<td>'AMOUNT OF' 'TRANSACTION'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC TRANS-DATE 18 6 2</td>
<td>'DATE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC BRANCH 24 2</td>
<td>'BRANCH'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REC NAME 26 20</td>
<td>'NAME'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80OUT 80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80SORT ACCOUNT 1 BRANCH NAME RPT-DAY START END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>808 START mmddyy</td>
<td>$First date (mmddyy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>808 END mmddyy</td>
<td>$Second date (mmddyy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>808 RPT-DAY mmddyy</td>
<td>$Third date (mmddyy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>803 MONTHLY</td>
<td>$Title (MONTHLY REPORT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80410010 NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80410030 ACCOUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80410040 BRANCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80410050 RPT-DAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80420010 'PERIOD: FROM'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80420024 START</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80420034 'TO'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80420038 END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8051*010 TRANS-DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8051*020 TRANS-IND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8051*030 TRANS-AMT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8051*070 TRANS-DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8061*0200 'TOTAL TRANSACTIONS'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8061*030 TRANS-AMT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>806108000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The first, second and third dates would be the actual dates in the format shown.

**The Report**

REPORT NO. 80 MONTHLY | $Title | mm/dd/yy | PAGE |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARON ARMSTRONG</td>
<td>15060</td>
<td>32 mm/dd/yy</td>
<td>1</td>
</tr>
<tr>
<td>PERIOD: FROM</td>
<td>mm/dd/yy</td>
<td>TO</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>DEPOSIT/</td>
<td>AMOUNT OF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assigning Default Values to Symbolic Parameters (USE Parameter)

You can assign default values to symbolic parameters in your stored or inline code and use keywords to provide documentation.

How to Do It

Code:

- The word DEFAULT on the first line of stored code.
  If used inline, the DEFAULT clause is entered as the last of the USE parameter clauses.

- Symbolic parameter names entered sequentially, starting with & & 1.

- A keyword for each symbolic parameter name. (Any keywords in the WITH VALUES clause must match those in the DEFAULT clause.)

- The value for the keyword.

Demonstration 33

Objective

This example uses a DEFAULT clause to produce the same report as the example using a WITH VALUES clause.

Procedure
Default values are used for symbolic parameters whose values do not appear on the WITH VALUES clause.

Keywords START, END, and TITLE provide documentation.

RPT-DAY, the keyword for &3, is specified on both the WITH VALUES clause and the DEFAULT clause.
The value given on the WITH VALUES clause (mmddyy) overrides the value on the DEFAULT clause, where mmddyy would be actual dates in this format.

Complete Code

Below follows the job stream and the stored code:

The Job Stream

```
USE RPT627
WITH VALUES (RPT-DAY=mmddyy) $Overriding value
/```

Note: The rpt-day value would show actual dates in the format shown.

The Stored Code

```
col. 2
DEFAULT &1=START=mmddyy $Default values if not overridden
&2=END=mmddyy
&3=RPT-DAY=mmddyy
&4=TITLE='MONTHLY REPORT' IN 8
0
RECF ACCOUNT 15 'ACCOUNT'
RECF TRANS-IND 61 'DEPOSIT/WITHDRAWAL'
RECF TRANS-AMT 7 11 2 DP=2 'AMOUNT OF TRANSACTION'
RECF TRANS-DATE 18 6 2 'DATE'
RECF BRANCH 24 2 'BRANCH'
RECF NAME 26 20 'NAME'
80OUT 80
80SORT ACCOUNT 1 BRANCH NAME RPT-DAY START END
8000 START &61800 END &62800 RPT-DAY &63800 &6480410010 NAME
80410030 ACCOUNT 80410040 BRANCH
80410050 RPT-DAY FD
80420010 'PERIOD: FROM' FD
80420024 START FD
80420034 'TO' FD
80420038 END FD
8051*010 TRANS-DATE FD HR
8051*020 TRANS-IND HR
8051*030 TRANS-AMT HR
8051*070 TRANS-DATE FD HR
8061*0200 'TOTAL TRANSACTIONS' FD
8061*030 TRANS-AMT
```

Note: The code would show actual dates for start, end, and rpt-day in the format shown.

Result
**Nesting the USE Parameter**

You can nest the USE parameter. One or more USE parameters and WITH VALUES clauses can be included in stored code.

If one USE parameter appears in the stored code, it is treated as a nested parameter in the job stream at run-time.

**How to Do It**

**Code:**

1. The **USE** parameter followed by an asterisk (*), and a **WITH VALUES** clause at the beginning of the code affected

2. **END** at the end of the code affected by the value assignments

**Demonstration 34**

**Objective**

This is a partial example that produces the same report as that shown earlier for the reports generated from the WITH VALUES and DEFAULT clauses.

**Procedure**

- A **USE** parameter in the job stream copies the stored code.
• **USE *** parameter and a **WITH VALUES** clause assign values to the symbolic parameters.

• An **END** clause signals the end of the code affected by the USE parameter.

### The Job Stream

USE RPT624/*

### The Stored Code

```plaintext
col. 2 $Start of code sequence affected by changes
USE * WITH VALUES
(mmddyy, mmddyy, mmddyy, 'MONTHLY REPORTS') IN 80
REC ACCOUNT 1 5 'ACCOUNT'
.
.
END $End of code incorporating changes
```

**Note:** The code would specify actual dates in the format shown.

---

### Modifying Code (USE Parameter)

You can change report numbers and character strings, drop or keep parameters, and renumber type 7 and type 8 sequence numbers with the USE parameter.

### How to Do It

• To modify code, use one or more **CHANGE** clauses for each USE parameter. Only one clause can specify a report number change.

• To eliminate or retain code, use one or more **DROP** or **KEEP** clauses. If both DROP and KEEP are coded, precedence is given to the DROP clause.

• To renumber type 7 and type 8 sequence numbers, use one or more **RENUMBER** clauses. The RENUMBER clause must be the last clause coded on the USE parameter.

### Demonstration (1) Changing Report Numbers and Character Strings

**Objective**

This report is derived from stored code. The code is changed to provide updated branch office account activity information.

**Procedure**
• The report number is changed from 80 to 01.

• The title is changed from ACCOUNT ACTIVITY BY BRANCH to BRANCH ACCOUNTS.

• A type 5 parameter is changed to a type 4 to allow the branch number to print as a heading.

**Complete Code**

col. 2

IN 80 F

USE *

CHANGE RPTNO TO 01 AND

$Chained together with AND

'S1*010' 4/9 TO '410010' AND

'ACCOUNT ACTIVITY BY BRANCH' TO 'BRANCH

ACCOUNTS'

REC T-ACCOUNT 1 5 'ACCOUNT'
REC T-TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC T-TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC T-DATE 18 6 2 'DATE'
REC T-BRANCH 24 2

$Report number changed

800OUT D

805SORT T-BRANCH 1

803 ACCOUNT ACTIVITY BY BRANCH

$Title changed

8051*010 T-BRANCH HR

$Changed to a type 4 parameter

8051*020 T-ACCOUNT HR
8051*030 T-NAME HR
8051*050 T-TRANS-AMT HR
8051*060 T-TRANS-IND HR
8051*070 T-DATE FD HR

END
Demonstration (2) Using a KEEP Clause

Objective

This example changes inline code to produce a report of individual account activity.

Procedure

A KEEP clause is used to retain the title and three type 5 fields (ACCOUNT, TRANS-AMT, and TRANS-IND). All other fields are dropped.

Complete Code

```
col. 2
IN 80 F
USE *
CHANGE RPTNO TO 01 AND
'ACCOUNT ACTIVITY BY BRANCH' TO 'ACCOUNT ACTIVITY'
KEEP 3 AND 'ACCOUNT' AND 'TRANS-AMT' AND 'TRANS-IND'
REC ACCOUNT 1 5
'REPORT NO. 01'
REC TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC DATE 18 6 2 'DATE'
REC BRANCH 24 2
REC NAME 26 20 'NAME'
80OUT D $Report number changed
80SORT NAME 0
803 ACCOUNT ACTIVITY BY BRANCH $Title changed
8051*010 BRANCH
8051*020 ACCOUNT HR $Kept
8051*030 NAME HR
8051*050 TRANS-AMT HR $Kept
8051*060 TRANS-IND HR $Kept
8051*070 DATE FD HR
END
```

Sequential Parameter Listing

```
mm/dd/yy  SEQUENTIAL PARAMETER LISTING  volser Vnn.n PAGE 1
00 ** SYsin ** IN 80 F
C200138 INSTALLATION SECURITY OPTION IS NO
```
Demonstration (3) Using a DROP Clause

Objective

This example uses the DROP clause to produce the same report as that shown for the KEEP clause (see page 151).

Procedure

A DROP clause is used to leave out DATE, BRANCH, and NAME. ACCOUNT, TRANS-IND, and TRANS-AMT are kept. ADD A LABEL HERE

col. 2

IN 80 F

USE *

CHANGE RPTNO TO 01 AND

‘ACCOUNT ACTIVITY BY BRANCH’ TO ‘ACCOUNT ACTIVITY’

DROP ‘DATE’ AND ‘BRANCH’ AND ‘NAME’

REC ACCOUNT 1 5 ‘ACCOUNT’

REC TRANS-IND 6 1 ‘DEPOSIT/’ ‘WITHDRAWAL’

REC TRANS-AMT 7 11 2 DP=2 ‘AMOUNT OF’ ‘TRANSACTION’
Demonstration (4) Using the RENUMBER Clause

Objective

This example renumbers type 7 sequence numbers in inline code to produce a current Account Activity Report.

Procedure

The RENUMBER clause is used to change type 7 sequence numbers from 010, 020, 040, and 050 to 100, 200, 300, and 400, respectively.

Complete Code

```plaintext
col. 2
IN 80 F
USE *
CHANGE 'ACCOUNT ACTIVITY BY BRANCH' TO 'ACCOUNT ACTIVITY'
DROP 'BRANCH' AND 'NAME'
RENUMBER 7 1/100 TO 100 BY 100
RENUMBER 10 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
RENUMBER 10 18 6 2 'DATE'
RENUMBER 24 2 'NAME'
01OUT D
01SORT NAME 0
013 ACCOUNT ACTIVITY BY BRANCH
0151*010 BRANCH
0151*020 ACCOUNT HR
0151*030 NAME HR
0151*050 TRANS-AMT HR
0151*060 TRANS-IND HR
0151*070 DATE FD HR
01OUT D
01SORT NAME 0
010 MMDD
010 YY
010 YMMDD
013 ACCOUNT ACTIVITY BY BRANCH
0151*010 BRANCH
0151*020 ACCOUNT HR
0151*030 NAME HR
0151*050 TRANS-AMT HR
0151*060 TRANS-IND HR
0151*070 YMMDD FD HH 'DATE' 017010 IF TRANS-AMT LE 100 DROP
017020 COMPUTE MMDDYY - (MMDD X 100) YY017050 COMPUTE (YY X 10000) + MMDD Y YMMDD
END
```

Result

Sequential Parameter Listing
00 ** SYSIN **

C280138 INSTALLATION SECURITY OPTION IS NO

USE *

CHANGE 'ACCOUNT ACTIVITY BY BRANCH' TO 'ACCOUNT ACTIVITY'
DROP 'BRANCH' AND 'NAME'
RENUMBER 7 1/100 TO 100 BY 100

01 ** SYSIN **

REC ACCOUNT 1 5 'ACCOUNT'
REC TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC TRANS-AMT 7 11 2 DP=2 'AMOUNT OF ' 'TRANSACTION'
REC MMDDYY 18 6 2
01OUT D
010 MMDD
010 YY
010 YYMMDD
013 ACCOUNT ACTIVITY $Changed
0151*020 ACCOUNT HR
0151*050 TRANS-AMT HR
0151*060 TRANS-IND HR
0151*070 YYMMDD FD HH 'DATE'
017100 IF TRANS-AMT LE 100 DROP $Renumbered
017200 COMPUTE TRUNC MMDDYY / 100 MMDD
017300 COMPUTE MMDDYY - (MMDD X 100) YY
017400 COMPUTE (YY X 10000) + MMDD YYMMDD
END

Report

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>ACCOUNT ACTIVITY</th>
<th>AMOUNT</th>
<th>$Ch</th>
<th>mm/dd/yy</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT</td>
<td>TRANSACTION</td>
<td>DEPOSIT/</td>
<td>WITHDRAWAL</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>15060</td>
<td>10,099.01</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15060</td>
<td>999.11</td>
<td>W</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15060</td>
<td>100,990.11</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16070</td>
<td>1,080.04</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23055</td>
<td>500.00</td>
<td>W</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23055</td>
<td>50,000.00</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29557</td>
<td>500,001.00</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30115</td>
<td>500,001.00</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30115</td>
<td>500,001.00</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31275</td>
<td>5,000,000.00</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33478</td>
<td>5,590,001.30</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34440</td>
<td>195.01</td>
<td>W</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34440</td>
<td>2,000.01</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36682</td>
<td>4,000.78</td>
<td>W</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69876</td>
<td>9,009,901.15</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69876</td>
<td>637.55</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99883</td>
<td>50,001.30</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99883</td>
<td>6,468.52</td>
<td>D</td>
<td>yy/mm/dd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The report would show actual dates in the format shown.

Copying and Modifying Code (=COPY Parameter)

You can copy stored code, change report numbers, and suppress INPUT parameters with the =COPY parameter.
How to Do It

Code =COPY in the SYSIN file, beginning in column 1.

On the same line, include one or all of the following:

- Source of the code to be used:
  - The name of the file
  - The name of the module
  - An asterisk (*) if inline code follows
- A RPTNO= keyword followed by the 2-digit report number being assigned to this report

Demonstration 35

Objective

This example copies stored record descriptions and parameters.

Procedure

- An =COPY parameter is used to copy record descriptions from the file RECS.
- An =COPY parameter is used to copy parameters
- The report number of PARAMS is changed to 02 (RPTNO=02).

Complete Code

col. 2
= COPY RECS = COPY PARAMS RPTNO=02/*

Result

A plus sign (+), in the sequential parameter listing below, identifies copied lines:

Sequential Parameter Listing

<table>
<thead>
<tr>
<th>mn/dd/yy</th>
<th>SEQUENTIAL PARAMETER LISTING</th>
<th>volser</th>
<th>Vnn.n</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 ** SYSIN **</td>
<td>=COPY RECS</td>
<td>C200138 INSTALLATION SECURITY OPTION IS NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ IN 200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ REC EMPLOYEE</td>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ REC FIRST-NAME</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ REC LAST-NAME</td>
<td>15</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ REC DEPARTMENT</td>
<td>115</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ REC SALARY</td>
<td>160</td>
<td>5</td>
<td>3</td>
<td>DP=2</td>
</tr>
<tr>
<td>+ REC TITLE</td>
<td>171</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=COPY PARAMS RPTNO=02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Report

REPORT NO. 02
EMPLOYEE SALARY LISTING BY DEPARTMENT

<table>
<thead>
<tr>
<th>DEPARTMENT NAME</th>
<th>EMPLOYEE NAME</th>
<th>JOB TITLE</th>
<th>ANNUAL SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>JENSON</td>
<td>RUPERT</td>
<td>MGR ACCTNG/PAYROLL</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>KIMBALL</td>
<td>MARIANNE</td>
<td>ACCOUNTANT</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>HUTTON</td>
<td>EDWARD</td>
<td>FINANCIAL ANALYST</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>BLOOMER</td>
<td>JUNE</td>
<td>PAYROLL CLERK</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>KING</td>
<td>DORIS</td>
<td>AR CLERK</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>NICEMAN</td>
<td>BRIAN</td>
<td>AP CLERK</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>WILCO</td>
<td>ROGER</td>
<td>MGR THERMOREGULATION</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>FINN</td>
<td>PHINEAS</td>
<td>KEEPER OF BALLOONS</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>CLOTH</td>
<td>TERRY</td>
<td>HUMIDITY CONTROL CLK</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>TIME</td>
<td>MARK</td>
<td>WINTERIZER</td>
</tr>
<tr>
<td>THERMOREGULATION</td>
<td>KASPAR</td>
<td>JOE</td>
<td>WINTERIZER</td>
</tr>
</tbody>
</table>

214,500.00
227,000.00
2,522,500.00

Copying and Modifying Code (=MACRO Parameter)

You can copy or modify code for a single CA Culprit run with the =MACRO parameter. Symbolic parameters can be used in copied or inline code as a substitute for arguments; parameters can be changed or suppressed.

How to Do It

- To code changes affecting copied code, enter =MACRO after the INPUT parameter.

- To code changes affecting inline code, enter =MACRO * at the beginning of the code to be changed.

- To modify code, enter the values for symbolic parameters, enclosed in parentheses, on the same line as the =MACRO (*) parameter.

- To eliminate or retain code, enter =DROP and =CHANGE on the line immediately following the =MACRO (*) parameter.

- Enter =MEND immediately after the =MACRO clauses.

- Code a second =MACRO */ =MEND sequence before inline code that is not to be changed.
Demonstration (1) Providing Symbolic Parameter Values

Objective

This report uses copied code that contains symbolic fields to list employees with salaries less than $60,000 in all departments except Brainstorming.

Procedure

The =MACRO * parameter provides the values for the symbolic parameters.

Complete Code

```
col. 2
IN  200 F 400 PS(TAPE)
REC EMP-NAME  5  25 'EMPLOYEE NAME'
REC DEPARTMENT 115 45 'DEPARTMENT'
REC SALARY 160 5 3 DP=2
013 EMPLOYEE SALARY LISTING
010 COUNT 1=MACRO * (DEPARTMENT 'BRAIN STORMING ' EMP-NAME SALARY 60000)=MEND 0151*00
0 COUNT
0151*010 $63 $EMP-NAME
0151*020 $64 $SALARY
0161*010 'TOTAL COUNT'
0161*020 COUNT $SZ=8
017010 IF $61 EQ $62 DROP $If DEPARTMENT equal BRAINSTORMING
017 IF $64 GE $65 DROP $If SALARY greater than 60000
```

Result

In the sequential parameter listing, below:

- A plus sign (+) identifies the substitutions made.
- An asterisk (*) indicates a continuation line for parameters having more than one substitution.

Sequential Parameter Listing
CA IDMS - 19.0

++ 017    IF SALARY
++* GE 60000
++* DROP   $If SALARY greater than 60000

Report

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE SALARY LISTING</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SALARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>15,000.00</td>
<td></td>
</tr>
<tr>
<td>EDWARD</td>
<td>HUTTON</td>
<td>44,000.00</td>
<td></td>
</tr>
<tr>
<td>MARIANNE</td>
<td>KIMBALL</td>
<td>45,000.00</td>
<td></td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td>14,500.00</td>
<td></td>
</tr>
<tr>
<td>BREAN</td>
<td>NICERMAN</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>JANE</td>
<td>FERIDALE</td>
<td>22,500.00</td>
<td></td>
</tr>
<tr>
<td>GEORGE</td>
<td>FONRAD</td>
<td>14,750.00</td>
<td></td>
</tr>
<tr>
<td>ROBIN</td>
<td>GARDNER</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>20,000.00</td>
<td></td>
</tr>
<tr>
<td>TERENCE</td>
<td>KLMELLEN</td>
<td>43,000.00</td>
<td></td>
</tr>
<tr>
<td>SANDY</td>
<td>KRAAMER</td>
<td>14,000.00</td>
<td></td>
</tr>
<tr>
<td>JAMES</td>
<td>GALLWAY</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>PERCY</td>
<td>GRANGER</td>
<td>34,500.00</td>
<td></td>
</tr>
<tr>
<td>VLADIMIR</td>
<td>HEAROWITZ</td>
<td>33,000.00</td>
<td></td>
</tr>
<tr>
<td>JAMES</td>
<td>JACOBI</td>
<td>55,000.00</td>
<td></td>
</tr>
<tr>
<td>JULIE</td>
<td>JENSEN</td>
<td>37,000.00</td>
<td></td>
</tr>
<tr>
<td>LARRY</td>
<td>LITERATA</td>
<td>37,500.00</td>
<td></td>
</tr>
<tr>
<td>KATHERINE</td>
<td>0' HEARN</td>
<td>42,500.00</td>
<td></td>
</tr>
<tr>
<td>RALPH</td>
<td>TYRO</td>
<td>20,000.00</td>
<td></td>
</tr>
<tr>
<td>TOTAL COUNT</td>
<td></td>
<td>44</td>
<td></td>
</tr>
</tbody>
</table>

Demonstration (2) Modifying Parameters and the Report Number

Objective

This example modifies the report number and parameter of copied code to produce a report on employee salaries.

Procedure

- An **=DROP** clause is used to omit the INPUT and type 6 parameters.
- An **=CHANGE** clause is used to change the report number from 01 to 07.

Complete Code

<table>
<thead>
<tr>
<th>col. 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IN 200</td>
<td></td>
</tr>
<tr>
<td>=MACRO RPT631A</td>
<td></td>
</tr>
<tr>
<td>(DEPARTMENT 'BRAIN STORMING ' EMP-NAME SALARY 60000)=DROP INPUT</td>
<td></td>
</tr>
<tr>
<td>TYPE=6</td>
<td></td>
</tr>
<tr>
<td>$Drop INPUT and type 6 parameters=CHANGE RPTNO=07 $Change the report number</td>
<td></td>
</tr>
<tr>
<td>=MEND</td>
<td></td>
</tr>
<tr>
<td>/*</td>
<td></td>
</tr>
</tbody>
</table>

Result

In the sequential parameter listing below:

- A plus sign (+) indicates copied parameters.
A double plus sign (++) indicates a copied parameter and a value substitution.

An asterisk (*) indicates a continuation line when more than one substitution occurs.

Sequential Parameter Listing

<table>
<thead>
<tr>
<th>mm/dd/yy</th>
<th>SEQUENTIAL PARAMETER LISTING</th>
<th>volser Vnn.n PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 ** SYSSIN **</td>
<td>IN 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C200138 INSTALLATION SECURITY OPTION IS NO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MACRO RPT631A (DEPARTMENT 'BRAIN STORMING' EMP-NAME SALARY 60000)

DROPOUT INPUT TYPE=6

=CHANGE RPTNO=07

MEND

W C200052 INPUT CARD DROPPED

V C200052 INPUT CARD DROPPED

RECO Emp-NAME 5 25 'EMPLOYEE NAME'

REC DEPARTMENT 115 45 'DEPARTMENT'

REC SALARY 160 5 3 DP=2

073 EMPLOYEE SALARY LISTING

070 COUNT 1

0751*010 **3 HR

0751*010 EMP-NAME HR

0751*020 **4 SZ=7 F2 HF

0751*020 SALARY SZ=7 F2 HF

0761*010 'TOTAL COUNT'

0761*020 COUNT SZ=8

077010 IF &&1 EQ &&2 DROP

077010 IF DEPARTMENT

*EQ 'BRAIN STORMING'

*DROPOUT

077 IF &&4 GE &&5 DROP

077 IF SALARY

**GE 60000

**DROPOUT

Report

<table>
<thead>
<tr>
<th>REPORT NO. 07</th>
<th>EMPLOYEE SALARY LISTING</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EMPLOYEE NAME</td>
<td>SALARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>15,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDWARD</td>
<td>HUTTON</td>
<td>44,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIANNE</td>
<td>KINBALL</td>
<td>45,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td>14,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRIAN</td>
<td>NICEMAN</td>
<td>14,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAYNE</td>
<td>FERNDALE</td>
<td>22,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEORGE</td>
<td>FONRAD</td>
<td>14,750.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROBIN</td>
<td>GARDNER</td>
<td>14,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>20,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERENCE</td>
<td>KLWELLON</td>
<td>46,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SANDY</td>
<td>KREAMER</td>
<td>14,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HERBERT</td>
<td>LIPSICH</td>
<td>18,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NANCY</td>
<td>TERNER</td>
<td>13,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BETH</td>
<td>CLOUT</td>
<td>52,750.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERRY</td>
<td>CLOTH</td>
<td>36,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHINEAS</td>
<td>FINN</td>
<td>45,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOE</td>
<td>KASPAR</td>
<td>31,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td>TIME</td>
<td>33,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAYNE</td>
<td>DOUGH</td>
<td>33,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAMES</td>
<td>GALLWAY</td>
<td>33,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERCY</td>
<td>GRANGER</td>
<td>34,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLADIMIR</td>
<td>HEARWITZ</td>
<td>33,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAMES</td>
<td>JACOBI</td>
<td>35,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULIE</td>
<td>JENSEN</td>
<td>37,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LARRY</td>
<td>LITTERATA</td>
<td>37,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KATHERINE</td>
<td>O'HEARN</td>
<td>42,500.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RALPH</td>
<td>TYRO</td>
<td>20,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>93,500.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Listing the Contents of a Data File

You can list the contents of a data file before CA Culprit reports are run against a file. Three to ten fields for a specified number of records can be printed in the order of appearance in the file.

You can also sort records, select records, and print total lines.

How to Do It

Code:

1. An =MACRO or USE parameter with:
   - AMLIST immediately followed by a number from 3 to 10, depending on how many fields are to be listed
   - The WITH VALUES clause if you code the USE parameter
   - An argument list enclosed in parentheses (the number of records to be listed, the subtitle, the field names in the order of listing)

2. The =MEND parameter, if =MACRO is used

Demonstration (1) Printing Fields with AMLIST3

Objective

This report prints three fields (EMP-NAME, TITLE, and SALARY) of the first ten records in the input file.

Procedure

- =MACRO is used to copy the input file.

- AMLIST3 calls the AMLIST routine to print three fields (EMP-NAME, TITLE, and SALARY) from the first ten records in the input file.

- The heading of the report is COMPANY EMPLOYEES.

- Column headings are the field names.

- =MEND signals the end of the code affected by the =MACRO parameter.

Complete Code

```
col. 2

IN  200
REC  EMP-NAME  5  25
REC  SALARY    160  5  3 DP=2
```
Sequential Parameter Listing

COUNT RECORDS READ
GET MAXIMUM
TAKE ALL RECORDS
DROP IF OVER MAXIMUM

Report

COMPANY EMPLOYEES

NAME TITLE SALARY
JUNE BLOOMER PAYROLL CLERK 15,000.00
EDWARD HUTTON FINANCIAL ANALYS 44,000.00
RUPERT JENSON MGR ACCTNG /PAYROLL 82,000.00
MARIANNE KIMBALL ACCOUNTANT 45,000.00
DORIS KING AR CLERK 14,500.00
BRIAN NICEMAN AP CLERK 14,000.00
HERBERT CRANE MGR COMPUTER OPS 75,000.00
JANE FERNADE COMPUTER OPERATO 22,500.00
Procedure

Demonstration (2) Modifying Code Using WITH VALUES and CHANGE

Objective

This report is generated from the source code used for Printing Fields with AMLIST3 (see page 160). Three fields from the first ten records of the input file are listed.

Procedure

- The USE parameter is coded with the AMLIST routine.
- The WITH VALUES clause specifies a subtitle for the report by enclosing the literal that is to appear on the generated type 4 parameter ('COMPANY EMPLOYEES') in double quotation marks.
- The CHANGE clause changes the report number from 03 to 01.
- The SORT parameter sorts the data by employee title.
- A subtotal prints for every title.
- Type 7 process code selects the first ten records in the file in which the salary amount is less than or equal to $25,000.

⚠️ Note: CA Culprit sorts process parameters in ascending order by sequence number. Therefore, be sure to code a sequence number that reflects the sequence in which record selection is to occur.

Complete Code

col. 2

IN 200
REC EMP-NAME 5 25
REC SALARY 160 5 3 DP=2
REC TITLE 171 20
USE AMLIST3 WITH VALUES (10 "'COMPANY EMPLOYEES'" EMP-NAME TITLE SALARY) CHANGE RPTNO TO 01
01SORT TITLE -
017010 SALARY GT 25000 DROP
0161*030 SALARY

Input Parameter Listing
Sequential Parameter Listing

REPORT NO. 01 DETAIL LIST mm/dd/yy PAGE 1

COMPANY EMPLOYEES

<table>
<thead>
<tr>
<th>EMP-NAME</th>
<th>TITLE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIAN</td>
<td>NICEMAN</td>
<td>AP CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>DORIS</td>
<td>KING</td>
<td>AR CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,500.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,500.00</td>
</tr>
<tr>
<td>JANE</td>
<td>FERNDALE</td>
<td>COMPUTER OPERATOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22,500.00</td>
</tr>
<tr>
<td>DOUGLAS</td>
<td>KAHALY</td>
<td>COMPUTER OPERATOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20,000.00</td>
</tr>
<tr>
<td>HERBERT</td>
<td>LIPSICH</td>
<td>COMPUTER OPERATOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18,500.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61,000.00</td>
</tr>
<tr>
<td>GEORGE</td>
<td>FONRAD</td>
<td>DATA ENTRY CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,750.00</td>
</tr>
<tr>
<td>ROBIN</td>
<td>GARDNER</td>
<td>DATA ENTRY CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>SANDY</td>
<td>MAHNER</td>
<td>DATA ENTRY CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14,000.00</td>
</tr>
<tr>
<td>NANCY</td>
<td>TERNER</td>
<td>DATA ENTRY CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55,750.00</td>
</tr>
<tr>
<td>JUNE</td>
<td>BLOOMER</td>
<td>PAYROLL CLERK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15,000.00</td>
</tr>
</tbody>
</table>
Additional Standard File Facilities

CA Culprit allows you to create and store nonprint reports and vary headings on printed reports. You can also access non-database files that are defined in the data dictionary. For more information, see the following topics:

- Creating Nonprint Report Output (see page 164)
- Creating Variable Headings (see page 170)
- Accessing Non-database Files Defined in the Data Dictionary (see page 173)

Creating Nonprint Report Output

You can write and store CA Culprit reports on conventional files on tape or disk.

These nonprint reports can be created from:

- Selected report information, including final report summary information
- Variable-length records
- Sequential files created from database information

The output from a CA Culprit run can be stored on tape or disk as:

- A sequential (PS) file
- An indexed sequential (IS) file
- A card (CARD) file
- A VSAM file
- A data table

How to Do It

Use:

- OUTPUT parameter options to set the output record size, block size, type of file, and the device to be used.
  If the device is TAPE, the label type is also specified on the OUT parameter.
- Special formatting codes on type 5 and 6 parameters:
- **FB** to convert an integer field value to binary output.

- **FU** to convert a numeric field value to a single precision floating point value. (SZ= and DP= cannot be used with the FU format code.)

- **FW** to convert a numeric field value to a double precision floating point value. (SZ= and DP= cannot be used with the FW format code.)

- **FP** to convert a numeric output field value into a signed packed decimal number.

- **FZ** to convert a numeric output field value into a signed zoned decimal number.

- No titles or headings.

- A plus sign (+) for a break code if you want total records for the output.

- Absolute column placement.

- The JCL label SYS020 to define the output file.

- No labels

- Edit Labels (http://wiki-dev.ca.com/display/IDMS/What+You+Can+Do+42)

### Demonstration (1) Writing to a Sequential File

**Objective**

This example writes out selected data items listed on REC parameters to a 40-byte sequential file. A hexadecimal dump shows how the data is stored on the file.

**Procedure**

- An **OUTPUT** parameter defines a sequential file that will store detail information extracted from the type 5 detail lines:

  - Records are 40 bytes, fixed length.

- **DD=SYS020** specifies the name of the output file in execution JCL.
  
  In a z/VSE environment, the logical unit number of the device receiving the output file (DD=SYS020,20) must also be specified if the file type is not CARD.

- Type 5 lines specify **absolute column placement** for the data items written out. The column specifications become the byte positions of the items on the record.

- **Format codes** are used on numeric fields.

**Complete Code**

<table>
<thead>
<tr>
<th>col. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN 150 F 1500 PS(TAPE)</td>
</tr>
<tr>
<td>REC CUST-NO 1 5 3</td>
</tr>
</tbody>
</table>
Demonstration (2) Converting to Packed Decimal Format

Objective

This example writes the report in Demonstration (1) to a sequential file.

Procedure

- The record length is specified as 29 bytes to accommodate the signed packed decimal numeric fields.
- FP converts the numeric output into signed packed decimal format.

Complete Code
Demonstration (3) Writing Complete Records

Objective
This example writes input records to a file by defining the entire input record on one REC parameter.

**Objective**

This example writes input records to a file by defining the entire input record on one REC parameter.

**Procedure**

- **ENTIRE-RECORD** defines the whole input record.

- CUST-NO is defined separately for use on the SORT parameter.

- TYPE is defined separately for use in type 7 logic.

**Complete Code**

```plaintext
IN 80 F 400
REC CUST-NO       1     9  2
REC TYPE          38     3 $Defines all input fields
01OUT 80 D
01SORT CUST-NO
01510001 ENTIRE-RECORD017010 IF TYPE = ('E00' TO 'T99') DROP

Result
```

```plaintext
0010766150002469000008222000144523008A27
0010766150000525000000474250283356401B44
0010766150001559000001327500324836977B49
0013269680000385000000355000422283501B27
0013269680023950000021950000144523008A27
0013269680000909500000811500262973311852
0058856830000000000000822000424304486X12
0058856830010000000850000262973311852
00588568300445000000490500669811172C06
```

**Demonstration (4) Writing Totals-only**

**Objective**

This example writes a sorted totals-only report to a tape sequential file.

**Procedure**

A plus sign (+) inserted after CUST-NO indicates a control break, which prevents a blank record from being written each time the break occurs.

```plaintext
IN 80 F 400
REC CUST-NO       1     9  2
REC GROSS-AMT     10     9  2
REC NET           19     9  2
REC ITEM-NO      28     10  2
REC TYPE         38     3
010 DIFF
01OUT 40 4000 T PS(TAPE)01SORT CUST-NO + $Control break on CUST-NO
01510000 GROSS-AMT
01510000 NET
01510000 DIFF
017010 IF TYPE = ('E00' TO 'T99') DROP
017015 GROSS-AMT - NET DIFF
01610001 CUST-NO FN
01610010 GROSS-AMT FZ
```
Demonstration (5) Writing Variable-length Records

**Objective**

The following example reads in variable-length records and writes out variable-length records to a sequential file.

**Procedure**

- Input records are sorted by customer number and selectively written, depending on the value of TYPE.

- REC parameters define the complete input file in three segments, none of which exceed 256 bytes:
  - The first four bytes of the record contain the RDW. A new RDW is not required in this example. User module CULLUS33 can be used if a new record descriptor word (RDW) must be generated.
  - The sort key CUST-NO and the type 7 field TYPE are the only individual fields defined.

- Variable length output files are created by coding:
  - A sequential output file type **OUT PS**
  - JCL output specifications of:
    - **RECFM=V or VB (SYS020)**
    - An increased LRECL size (a multiple of 4 but not larger than 4096 bytes or the block size) in SYS006 of the JCL to accommodate the SORT parameters

**Complete Code**

```assembly
IN 640 V
REC CUST-NO 5 5 3 $Used as a sort key
REC TYPE 70 3 $Used in type 7 logic
PART-ONE 1 250
PART-TWO 251 250
PART-THREE 501 140
OUT 640 D PS
PART-ONE 1 250
PART-TWO 251 250
PART-THREE 501 140
CUST-NO OUT 640 D PS01510001
PART-ONE
01510251 PART-TWO
01510501 PART-THREE
017010 IF TYPE EQ ('E00' TO 'T99') DROP
```

Demonstration (6) Writing from the Database

**Objective**

This example creates a sequential data set, consisting of employee ID, employee name, and department name from data stored on the employee database.
Procedure

A conventional file is created from data that resides on the database by coding:

- The **DATABASE** parameter with the name of an alternate dictionary in a multiple dictionary environment.
- The **INPUT (IN) DB** parameter with the name of the subschema to be accessed.
- **SYS020** in the JCL.
- Regular CA Culprit parameters.

See the section [Generating Reports From Database Files (see page 85)] for information on coding the database parameters and options.

**Complete Code**

```plaintext
col. 2
DATABASE DICTNAME=DOCUDICTIN DB SS=EMPSS01,EMPSCHM,100PATHAA DEPARTMENT EMPLOYEE01OUT
80 D PS Dd=SYS0201SORT DEPT-NAME-0410  EMP-NAME-0415
01510001 EMP-ID-0415  FZ
01510005 EMP-NAME-0415
01510030 DEPT-NAME-0410
```

Creating Variable Headings

You can change the values that appear in report headings.

**How to Do It**

**Code:**

- The field to be used as a heading on a **SORT** parameter
- **Type 4** parameters to reference the sort key
  The current value will be obtained at each control break.

When a field to be used as a variable heading cannot be included on the **SORT** parameter, a **BRANCH** to **HEAD** instruction can be used

**Demonstration (1) Using the SORT Parameter**

**Objective**

This report lists monthly account activity by branch. Each branch is listed on a separate page with the branch number as a subtitle.

**Procedure**
A SORT parameter specifies:

- BRANCH as a sort key
- A control break (1), which starts a new report page

A type 4 parameter specifies

- Absolute column placement
- The word BRANCH to be printed
- The value of BRANCH to be printed on the same line

Complete Code

col. 2

IN 80 F 400
REC ACCOUNT 1 5 'ACCOUNT'
REC TRANS-IND 6 1 'DEPOSIT/ ' 'withdrawal'
REC TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC DATE 18 6 2 'DATE'
REC BRANCH 24 2
REC NAME 26 20 'NAME'
800OUT805SORT BRANCH 1 ACCOUNT $Control break on BRANCH
803 ACCOUNT ACTIVITY BY BRANCH$Print BRANCH80410001 'BRANCH'
804 ACCOUNT ACTIVITY BY BRANCH80410009 $Print BRANCH
805 ACCOUNT ACTIVITY BY BRANCH80410009 $Print BRANCH
8051*020 ACCOUNT HR
8051*030 NAME HR
8051*050 TRANS-AMT HR
8051*060 TRANS-IND HR
8051*070 TRANS-DATE FD HR

Results

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>AMOUNT OF TRANSACTION</th>
<th>DEPOSIT/ WITHDRAWAL</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>10,099.01</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>990.11</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>15060</td>
<td>SHARON ARMSTRONG</td>
<td>180,990.11</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>19235</td>
<td>GARY NOBLES</td>
<td>80.04</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>21056</td>
<td>AMOS JOHNSON</td>
<td>1.02</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>AMOUNT OF TRANSACTION</th>
<th>DEPOSIT/ WITHDRAWAL</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>23055</td>
<td>JACK JACKSON</td>
<td>500.00</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>23055</td>
<td>JACK JACKSON</td>
<td>50,000.00</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>23055</td>
<td>JACK JACKSON</td>
<td>30.00</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>29557</td>
<td>IRWIN TRIMBLE</td>
<td>580,001.00</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>

Note: The report would show actual dates in the format shown.
Demonstration (2) Using the SORT/NOSORT Parameters

Objective

This is the same report as that shown in Demonstration (1). In this example, the input file is presorted and an actual sort is not needed.

Procedure

- A sort with the NOSORT option occurs on BRANCH:
  - SORT BRANCH 1 creates the control break, which makes the current value of BRANCH available to the type 4 reference.
  - NOSORT prevents the resorting of the already sorted file.

- Type 4 parameters:
  - Place the word BRANCH on the subtitle line.
  - Obtain and print the current value of BRANCH.

Complete Code

col. 2

IN 80 F 400
REC ACCOUNT 1 5 'ACCOUNT'
REC TRANS-IND 6 1 'DEPOSIT/' 'WITHDRAWAL'
REC TRANS-AMT 7 11 2 DP=2 'AMOUNT OF' 'TRANSACTION'
REC DATE 18 6 2 'DATE'
REC BRANCH 24 2
REC NAME 26 20 'NAME'
80OUT
80SORT BRANCH 1 NOSORT $Control break without a sort
803 ACCOUNT ACTIVITY BY BRANCH80410 'BRANCH' $Prints the branch number
809 BRANCH $Prints BRANCH80410
8051*020 ACCOUNT HR
8051*030 NAME HR
8051*050 TRANS-AMT HR
8051*060 TRANS-IND HR
8051*070 TRANS-DATE FD HR

Results

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>ACCOUNT ACTIVITY BY BRANCH</th>
<th>80</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>ACCOUNT ACTIVITY BY BRANCH</td>
<td>80</td>
<td>mm/dd/yy</td>
<td>PAGE</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>AMOUNT OF TRANSACTION</th>
<th>DEPOSIT/ WITHDRAWAL</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>15069</td>
<td>SHARON ARMSTRONG</td>
<td>10,099.01</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>15069</td>
<td>SHARON ARMSTRONG</td>
<td>990.11</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>15069</td>
<td>SHARON ARMSTRONG</td>
<td>180,990.11</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>16070</td>
<td>ARTHUR LINK</td>
<td>1,080.04</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>19235</td>
<td>GARY NOBLES</td>
<td>80.04</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>21056</td>
<td>AMOS JOHNSON</td>
<td>1.02</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>ACCOUNT ACTIVITY BY BRANCH</th>
<th>80</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>ACCOUNT ACTIVITY BY BRANCH</td>
<td>80</td>
<td>mm/dd/yy</td>
<td>PAGE</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>AMOUNT OF TRANSACTION</th>
<th>DEPOSIT/ WITHDRAWAL</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>23055</td>
<td>JACK JACKSON</td>
<td>500.00</td>
<td>W</td>
<td>mm/dd/yy</td>
</tr>
<tr>
<td>23055</td>
<td>JACK JACKSON</td>
<td>500,000.00</td>
<td>D</td>
<td>mm/dd/yy</td>
</tr>
</tbody>
</table>
Accessing Non-database Files Defined in the Data Dictionary

When a file is defined in the data dictionary, the file attributes are automatically supplied to any CA Culprit program referencing the file. The file attributes include record size, record type, block size, file type, device type, user module name, and label type.

You can use standard files that are defined in the data dictionary and revise already existing data dictionary definitions.

How to Do It

To use the data dictionary definitions of a standard file, code:

- The DATABASE parameter with the DICTNAME= option if an alternate dictionary is used
- The FN= option on the INPUT parameter
- REC parameters if the fields in the file are to be renamed or redefined
- HR on edit parameters to print column headings stored in the data dictionary

To revise data dictionary definitions, code:

- A redefined file definition on the INPUT parameter:
  
  Data dictionary file definition: record size is 100.
  
  CA Culprit code redefinition: IN 80

- A redefined field definition on a REC parameter with a field name identical to that used in the data dictionary:
  
  Dictionary field definition: T-ACCOUNT with PIC X(5).
  
  CA Culprit code redefinition: REC T-ACCOUNT 1 10 ‘ACCOUNT’

What You Can Do 44
Additional CA IDMS/DB Facilities

CA Culprit allows you to access the database, retrieve all record types, and write reports using standard CA Culprit code.

How to Prepare for Record Retrieval

Code:

1. The data dictionary name if using an alternate dictionary

2. The \texttt{INPUT (IN) DB} parameter specifying:
   - The subschema name
   - The size of the input buffer if more than 1000 bytes is needed
     The buffer must be large enough to hold:
     - Thirty-eight bytes of overhead
     - The sum of the length of each record type specified on the PATH parameter with the most records
     - Four additional bytes per record for the database key

3. The \texttt{path} or paths to navigate for retrieval of data

Retrieving Partial Paths

You can retrieve a segment of a path (partial string) when the entire path cannot be completed.

How to Do It

Specify a shorter (alternate) path.

A special type of alternate path ID, the null path identifier, can be used to indicate that an occurrence of the entry record was either not found or not selected.

The null path identifier is discussed later in this section.

Code:

1. A \texttt{PATH} parameter with the path identifier and the records to be accessed
2. An alternate path identifier placed one record after the last record of the shorter path

PATH01 EMPLOYEE EXPERTISE (02) SKILL $Shorter path is EMPLOYEE

Example: Retrieving Partial Paths

Objective

This report:

- Lists all employees, their insurance coverage, and hospital claims each time the complete path is navigated
- Lists all employees and their insurance coverage each time the alternate path is navigated
- Prints the path navigated for each report line

Procedure

- PATHAA EMPLOYEE COVERAGE HOSPITAL-CLAIM is the primary path for retrieval of employees having hospital claims.
- PATHAA EMPLOYEE COVERAGE HOSPITAL-CLAIM (BB) is an alternate path (PATHBB) for retrieval of insurance coverage codes.
- The value of PATH-ID is tested in the type 7 procedure logic for selecting one or both detail lines for printing.

Complete Code

col. 2

DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01
PATHAA EMPLOYEE COVERAGE HOSPITAL-CLAIM (BB) 013 EMPLOYEE LISTING
010 EMP-NAME \\
010 INS-CODE \\
0151*005 PATH-ID HH 'PATH' 'USED'
0151*020 EMP-NAME-0415 HH 'EMPLOYEE' 'NAME'
0151*030 INS-PLAN-CODE-0400 HH 'INSURANCE' 'PLAN CODE'
0152*040+HOSPITAL-NAME-0430 HH 'HOSPITAL'
017010 IF EMP-NAME-0415 EQ EMP-NAME AND INS-PLAN-CODE-0400 EQ
* INS-CODE DROP $Eliminate duplicate entries
017020 MOVE EMP-NAME-0415 TO EMP-NAME
017030 MOVE INS-PLAN-CODE-0400 TO INS-CODE
017100 IF PATH-ID EQ 'AA' 300 017200 TAKE 1 017300 TAKE (1 2)

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEE LISTING</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH USED</td>
<td>EMPLOYEE NAME</td>
<td>INSURANCE PLAN CODE</td>
<td>HOSPITAL</td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>KATHERINE O’HEARN</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>KATHERINE O’HEARN</td>
<td>002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>ROBBY WILDER</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>BURT LANCHESTER</td>
<td>004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>BURT LANCHESTER</td>
<td>003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>BURT LANCHESTER</td>
<td>002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ GENERAL HOSPITAL
Retrieving Record Types by Name

You can specify selection criteria for database records named on one or more PATH parameters.

How to Do It

Code SELECT or BYPASS parameters after the PATH parameters:

- The IN PATH clause specifies selection criteria for a specific path.
- The BUFFER specifies selection criteria for every path. (BUFFER is discussed further later in this section.)

Example: Retrieving Record Types by Name

Objective

This example creates two reports:

- Report 01 lists all employees for each department.
- Report 02 lists the skills of employees who work in departments 4000 and 3200.

Procedure

- Two paths are used to access data:
  for departments 4000 and 3200.
- A SELECT statement retrieves path BB when the department id is 4000 or 3200.
Complete Code

col. 2

DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01PATHAA DEPARTMENT EMPLOYEE PATHBB DEPARTMENT EMPLOYEE EXPERTISE SKILL
LECT DEPARTMENT IN PATH BB WHEN DEPT-ID-0410 EQ (4000 3200)
013 EMPLOYEES BY DEPARTMENT
0151*020 DEPT-NAME-0410 HH 'DEPARTMENT'
0151*030 EMP-NAME-0415 HH 'EMPLOYEE'
017010 IF PATH-ID NE 'AA' DROP 023 EMPLOYEE SKILL LEVELS FOR DEPTS 4000 AND 3200
0251*010 DEPT-ID-0410 FN HH 'DEPT ID'
0251*020 DEPT-NAME-0410 HH 'DEPARTMENT'
0251*030 EMP-NAME-0415 HH 'EMPLOYEE'
0251*040 START-YEAR-0415 HH 'START YEAR'
0251*050 SKILL-NAME-0455 HH 'SKILL'
027010 IF PATH-ID NE 'BB' DROP

Result

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>EMPLOYEES BY DEPARTMENT</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
<th>DEPARTMENT</th>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>JUNE</td>
<td>BLOOMER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>EDWARD</td>
<td>HUTTON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>RUPERT</td>
<td>JENSON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>MARIANNE</td>
<td>KIMBALL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>DORIS</td>
<td>KING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACCOUNTING AND PAYROLL</td>
<td>BRIAN</td>
<td>NICEMAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>CRANE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>JANE</td>
<td>FERNADE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>GEORGE</td>
<td>FORRAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>ROBIN</td>
<td>GARDNER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>TERENCE</td>
<td>KLMWELLEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>SANDY</td>
<td>KRAANER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>LIPSICH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTER OPERATIONS</td>
<td>NANCY</td>
<td>TERNER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>BETH</td>
<td>CLOUD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>ALAN</td>
<td>DONOVAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLUE SKIES</td>
<td>DANIEL</td>
<td>MOON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRAINSTORMING</td>
<td>HARRY</td>
<td>ARM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REPORT NO. 02</th>
<th>EMPLOYEE SKILL LEVELS FOR DEPTS 4000 AND 3200</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
<th>DEPT ID</th>
<th>DEPARTMENT</th>
<th>EMPLOYEE</th>
<th>START YEAR</th>
<th>SKILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>CRANE</td>
<td>77</td>
<td>IDMSDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>CRANE</td>
<td>77</td>
<td>ASSEMBLER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>CRANE</td>
<td>77</td>
<td>DEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>CRANE</td>
<td>77</td>
<td>RPGII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>JANE</td>
<td>FERNADE</td>
<td>79</td>
<td>IDMSDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>JANE</td>
<td>FERNADE</td>
<td>79</td>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>GEORGE</td>
<td>FORRAD</td>
<td>80</td>
<td>DATA ENTRY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>GEORGE</td>
<td>FORRAD</td>
<td>80</td>
<td>FILING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>ROBIN</td>
<td>GARDNER</td>
<td>81</td>
<td>DATA ENTRY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>79</td>
<td>KEYPUNCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>79</td>
<td>FILING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>DOUGLAS</td>
<td>KAHALLY</td>
<td>79</td>
<td>TYPING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>TERENCE</td>
<td>KLMWELLEN</td>
<td>78</td>
<td>COBOL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>TERENCE</td>
<td>KLMWELLEN</td>
<td>78</td>
<td>ASSEMBLER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>SANDY</td>
<td>KRAANER</td>
<td>81</td>
<td>KEYPUNCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>SANDY</td>
<td>KRAANER</td>
<td>81</td>
<td>TYPING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>HERBERT</td>
<td>LIPSICH</td>
<td>81</td>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>NANCY</td>
<td>TERNER</td>
<td>82</td>
<td>FILING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>NANCY</td>
<td>TERNER</td>
<td>82</td>
<td>PHONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>COMPUTER OPERATIONS</td>
<td>NANCY</td>
<td>TERNER</td>
<td>82</td>
<td>TYPING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>MICHAEL</td>
<td>ANGELO</td>
<td>79</td>
<td>RAINBOW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>MICHAEL</td>
<td>ANGELO</td>
<td>79</td>
<td>ART DESIGN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>MICHAEL</td>
<td>ANGELO</td>
<td>79</td>
<td>PASTEUP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>MONTE</td>
<td>BANK</td>
<td>78</td>
<td>PUB REL MGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>CHARLES</td>
<td>BOWER</td>
<td>77</td>
<td>PHOTO COLOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
<td>CHARLES</td>
<td>BOWER</td>
<td>77</td>
<td>PHOTO B/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Retrieving Record Types by Key

You can retrieve the entry record on a PATH parameter directly by using a specific key value or a range of key values. Key values allow you to avoid sweeping a database area for a particular record.

The following types of key values can be specified:

- **CALC-key** values when entry records are stored in the CALC location mode
- **Index-key** values when entry records are indexed
- **Db-key** values for any location mode (VIA, CALC, or DIRECT) unless the entry records are indexed or are logical records
- **Any logical record fields** when entry records are logical records

How to Do It

Retrieve specific occurrences of the PATH entry record by using:

- **KEY** parameters to name key fields and key values
  To code a KEY parameter, supply the following information:
  1. The KEY keyword (beginning in column 2)
  2. The key field name
  3. A key value, a list of values, or a range of values

- A **KEYFILE** parameter to access a sequential file containing the key values
  To code a KEYFILE parameter, supply the following information:
  1. The KEYFILE keyword (beginning in column 2)
  2. The size, in bytes, of the key file record
  3. The KF= keyword expression, followed by the starting position of the key field in the file and its length in bytes
  4. Other options:
     - The **FN=** keyword expression to name a file defined to the data dictionary:

```
PATH01 DEPT-EMP-LR WHERE EMP-NAME EQ EMP-KEY
KEYFILE FN=EMPLOYEE-KEYS
```
The **LRFNAME=** keyword expression to name a logical record when the entry record is a logical record and you are supplying key information to the WHERE clause on the PATH parameter:

```
PATH01 DEPT-EMP-LR WHERE EMP-LAST-NAME EQ KEY-VALUE
```

You can also use the **KEYFILE** parameter to:

- Define fields in a sequential key file that are not defined to the data dictionary:

  ```
  REC EMP-ID (KEYFILE) 20 4
  ```

- Define selection criteria applicable to key-file records:

  ```
  KEYFILE 80 KF=4 SELECT DEPT-LOCATION EQ 'DEDHAMPLAC'
  ```

### What You Can Do 47

#### Demonstration (1) Accessing Records by Key

**Objective**

This example uses a KEY parameter to list the job ids, job titles, and job openings for three selected departments.

**Procedure**

- A **KEY** parameter is used to directly access DEPARTMENT records with ids of 0100, 3200, and 5100.
- The **key field** on the KEY parameter is DEPT-ID-0410 (the CALC-key value for the DEPARTMENT record).

**Complete Code**

```
col. 2
DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01
PATH01 DEPARTMENT EMPLOYEE EMPOSITION JOB
013 JOB OPENINGS
0151*010 DEPT-ID-0410 FN HH 'DEPARTMENT'
0151*020 JOB-ID-0440 FN HH 'JOB ID'
0151*030 TITLE-0440 HH 'JOB TITLE'
0151*040 NUMBER-OPEN-0440 HH 'JOB OPENINGS'
```

**Result**

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>JOB OPENINGS</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENT</td>
<td>JOB ID</td>
<td>JOB TITLE</td>
<td>OPENINGS</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>9001</td>
<td>PRESIDENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>9007</td>
<td>DIR WEATHER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demonstration (2) Accessing Records with a Key File

Objective

This report lists all employees associated with specific departments located at Dedham Place and University Avenue. Department IDs and building locations are stored in a key file.

Procedure

- A **KEYFILE** parameter defines a key file with records 80 bytes long. The key field is a CALC-key value that begins in position 1 and is four bytes long.

- A user-supplied **REC** parameter identifies **DEPT-LOCATION** as an alphanumeric field. **DEPT-LOCATION** starts in position 5 of the key file record for a length of ten bytes.

- A **SELECT** parameter uses **DEPT-LOCATION** to select only those records in the key file that specify Lost Brook or University Avenue.

- A **SORT** parameter sorts the output by building location. Building location appears as a heading on each page.

Complete Code

col. 2

```
DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01
PATHAA DEPARTMENT EMPLOYEE KEYFILE 80 KF=1 4 REC DEPT-LOCATION(KEYFILE) 5 10 SELECT DEPT-LOCATION EQ ('DEDHAMPLAC' 'UNIVERSITY') ORDER DEPT-LOCATION,1
01410005 'BUILDING LOCATION'
01410025 DEPT-LOCATION
0151*001 DEPT-ID-0410 FM '9999' HH 'DEPT ID'
0151*003 EMP-ID-0415 FM '9999' HH 'EMPLOYEE ID'
0151*005 EMP-NAME-0415 HH 'EMPLOYEE NAME'

The standard file used to store the keys is specified in the JCL:
```
Result

<table>
<thead>
<tr>
<th>BUILDING LOCATION</th>
<th>DEPT ID</th>
<th>EMPLOYEE ID</th>
<th>EMPLOYEE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDHAMPLAC</td>
<td>5200</td>
<td>0479</td>
<td>TERRY CLOTH</td>
</tr>
<tr>
<td></td>
<td>5200</td>
<td>0329</td>
<td>PHINEAS FINN</td>
</tr>
<tr>
<td></td>
<td>5200</td>
<td>0469</td>
<td>JOE KASPAR</td>
</tr>
<tr>
<td></td>
<td>5200</td>
<td>0355</td>
<td>MARK TIME</td>
</tr>
<tr>
<td></td>
<td>5200</td>
<td>0349</td>
<td>ROGER WILCO</td>
</tr>
<tr>
<td></td>
<td>5200</td>
<td>0371</td>
<td>BETH CLOUD</td>
</tr>
<tr>
<td></td>
<td>5300</td>
<td>0366</td>
<td>ALAN DONOVAN</td>
</tr>
<tr>
<td></td>
<td>5300</td>
<td>0321</td>
<td>DANIEL MOON</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0024</td>
<td>JANE DOUGH</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0029</td>
<td>JAMES GALLWAY</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0003</td>
<td>JENNIFER GARFIELD</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0028</td>
<td>PERCY GRANGER</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0027</td>
<td>VLADIMIR HEARWETZ</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0020</td>
<td>JAMES JACOBIE</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0019</td>
<td>JULIE JENSEN</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0035</td>
<td>LARRY LITERATA</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0023</td>
<td>KATHERINE O'HEARN</td>
</tr>
<tr>
<td></td>
<td>3100</td>
<td>0021</td>
<td>RALPH TYRO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BUILDING LOCATION</th>
<th>DEPT ID</th>
<th>EMPLOYEE ID</th>
<th>EMPLOYEE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIVERSITY</td>
<td>5100</td>
<td>0466</td>
<td>ROY ANDALE</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0457</td>
<td>HARRY ARM</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0467</td>
<td>C. BREEZE</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0334</td>
<td>CAROLYN CROW</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0301</td>
<td>BURT LANCHESTER</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0015</td>
<td>RENE MAKER</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0341</td>
<td>RICHARD MUNYON</td>
</tr>
<tr>
<td></td>
<td>5100</td>
<td>0458</td>
<td>RICHARD WAGNER</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0004</td>
<td>HERBERT CRANE</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0032</td>
<td>JANE FERNDALE</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0045</td>
<td>GEORGE FONRAD</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0053</td>
<td>ROBIN GARDNER</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0049</td>
<td>DOUGLAS KAHALLY</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0016</td>
<td>TERENCE KLEWELLEN</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0074</td>
<td>SANDY KRAEMER</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0031</td>
<td>HERBERT LIPPSICH</td>
</tr>
<tr>
<td></td>
<td>3200</td>
<td>0048</td>
<td>NANCY TERNER</td>
</tr>
</tbody>
</table>

Retrieving Stand-alone Records

You can retrieve record occurrences that do not participate in defined paths, as well as path-related records, by calling or branching to the DB-EXIT facility in type 7 logic.

How to Do It

To use the DB-EXIT facility, code:

1. A dummy PATH parameter (PATH--) with:
   - One or more database records
   - One or more logical records
2. A **STOP-RUN** command on a type 7 parameter if the only paths coded for the run are dummy paths.

3. A **CALL** or **BRANCH** to the DB-EXIT facility on a type 7 parameter
   A branch to DB-EXIT is preceded by a series of logic statements that move values to the DB-EXIT arguments.

4. **IDMS-STATUS** and **LR-STATUS** value checking in procedure logic statements after each DB-EXIT call.

**What You Can Do 48**

**Example: Retrieving Stand-alone Records**

**Objective**

This report lists the start date of five employees selected from the database EMPLOYEE record type. The selection occurs by means of the DB-exit facility.

**Procedure**

The code for this report includes:

- A dummy path, **PATH--**, allocates space in the input buffer for the retrieved records.
- The work field **IDS.5** contains employee ID values.
- A call to the **DB-EXIT** facility has the following selection criteria arguments:
  - **Argument 1** ("CALC") generates an OBTAIN CALC EMPLOYEE retrieval command.
  - **Argument 2** ("EMPLOYEE") specifies the EMPLOYEE record. The trailing blank is required (the record name is less than 16 characters).
  - **Argument 3** (EMP-ID-0415) specifies the name of the CALC-key field for the EMPLOYEE record.
  - **Argument 4** ("IDS.INDEX") specifies an occurrence of the work field IDS.5 that has the value of the EMP-ID-0415 field being selected.
  - **Argument 5** specifies the length of the EMP-ID-0415 field.
- **IDMS-STATUS NE '0000' 400** checks the IDMS Status for values that indicate error conditions or records not retrieved. If the condition is true, CA Culprit generates a buffer dump and stops the run.
- **STOP-RUN** terminates the run.

**Complete Code**
Retrieving All Record Occurrences

You can retrieve all the occurrences of a database record type by navigating (walking) through each set of occurrences with the DB-EXIT facility.

How to Do It

Code the arguments of the DB-EXIT facility with:

- **ARG1** -- A DB-EXIT argument that generates a CA IDMS/DB retrieval command
- **ARG2** -- The field to be obtained
- **ARG3** -- The area or set to be accessed

What You Can Do 49
Example: Retrieving All Record Occurrences

Objective

This example uses the DB-EXIT facility to retrieve all employees working in each office.

Procedure

The DB-EXIT facility is used to walk the OFFICE-EMPLOYEE set until the end of the set is reached:

- **CALL DB-EXIT ('FIRST-AREA' 'OFFICE ' 'ORG-DEMO-REGION ')** retrieves the first OFFICE record in the ORG-DEMO-REGION area.

- **CALL DB-EXIT ('FIRST' 'EMPLOYEE ' 'OFFICE-EMPLOYEE ')** retrieves the first EMPLOYEE record in the OFFICE-EMPLOYEE set if the first office retrieval was successful.

- **CALL DB-EXIT ('NEXT' 'EMPLOYEE ' 'OFFICE-EMPLOYEE ')** retrieves the next EMPLOYEE record until the end of the set is reached.

- **CALL DB-EXIT ('NEXT-AREA' 'OFFICE ' 'ORG-DEMO-REGION ')** retrieves the next OFFICE record and, if successful, returns to retrieve the first EMPLOYEE record for that office.

Complete Code

col. 2

DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01
PATH-- OFFICE EMPLOYEE
010 MSG 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQ'
010 JUNK
0151*005 IDMS-STATUS HH 'IDMS-STATUS' 'FIELD'
0151*010 OFFICE-CODE-0450 HH 'OFFICE CODE'
0151*020 EMP-NAME-0415 HH 'EMPLOYEE' 'NAME'
0152*005 IDMS-STATUS
0152*010 MSG
01OUT D
01SORT OFFICE-CODE-0450 0 EMP-NAME-0415
017 CALL DB-EXIT ('FIRST-AREA' 'OFFICE ' 'ORG-DEMO-REGION ')017 IF IDMS-
STATUS EQ '0000' 050
017 MOVE 'ERROR IN RETRIEVAL OF FIRST OFFICE' MSG
017 RELS 2
017 PERFORM 800
017050 CALL DB-EXIT ('FIRST' 'EMPLOYEE ' 'OFFICE-EMPLOYEE ')017 B 150
017100 CALL DB-EXIT ('NEXT' 'EMPLOYEE ' 'OFFICE-EMPLOYEE ')017150 IF IDMS-
STATUS EQ '0000' 200 $Test for record found
017 IF IDMS-STATUS EQ '0307' 300 $Test for end of set
017 MOVE 'ERROR IN RETRIEVAL OF EMPLOYEE' MSG
017 RELS 2
017 PERFORM 800
017200 RELS 1
017 B 100
017300 CALL DB-EXIT ('NEXT-AREA' 'OFFICE ' 'ORG-DEMO-REGION ')017 IF IDMS-
STATUS EQ '0000' 050
017 IF IDMS-STATUS EQ '0307' STOP
017 MOVE 'ERROR IN RETRIEVAL OF NEXT OFFICE' MSG
017 RELS 2
017 PERFORM 800
017800 JUNK / 0 JUNK $Forces a buffer dump
017 STOP

Result
### Testing for Record Occurrences

You can identify record occurrences that are not found in the database.

**How to Do It**

Create a **null path** by placing the path identifier immediately after the entry record.

Test the **current value of PATH-ID** in type 7 logic.

**What You Can Do 50**

**Example: Testing for Record Occurrences**

**Objective**

This report lists selected employees and their start year. If an employee record is not found, the message **EMPLOYEE NOT FOUND** is returned.

**Procedure**

- A primary path, **PATHOK**, returns the employee name, ID, and date of hire.

#### Table: Employee Status

<table>
<thead>
<tr>
<th>IDMS-STATUS FIELD</th>
<th>OFFICE CODE</th>
<th>EMPLOYEE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>001</td>
<td>BETSY ZEDI</td>
</tr>
<tr>
<td>0000</td>
<td>001</td>
<td>HERBERT CRANE</td>
</tr>
<tr>
<td>0000</td>
<td>001</td>
<td>HERBERT LIPSICH</td>
</tr>
<tr>
<td>0000</td>
<td>001</td>
<td>JAMES GALLOW</td>
</tr>
<tr>
<td>0000</td>
<td>001</td>
<td>JAMES JACOBI</td>
</tr>
<tr>
<td>0000</td>
<td>002</td>
<td>DOUGLAS KAHALLY</td>
</tr>
<tr>
<td>0000</td>
<td>002</td>
<td>SANDY KRAMER</td>
</tr>
<tr>
<td>0000</td>
<td>002</td>
<td>TOM FITZHUGH</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>ALAN DONOVAN</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>BET CLAUD</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>BURT LANCHESTER</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>CAROLYN CROW</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>DANIEL MOON</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>RENE MAKER</td>
</tr>
<tr>
<td>0000</td>
<td>005</td>
<td>TERRY CLOTH</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>C. BREEZE</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>HARRY ARM</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>JOE KASPAR</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>MARK TIME</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>RICHARD MUNYON</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>RICHARD WAGNER</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>ROGER WILCO</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>ROY ANDALE</td>
</tr>
<tr>
<td>0000</td>
<td>008</td>
<td>THEMIS PAPAZEUS</td>
</tr>
</tbody>
</table>
A null path, **NO**, places NO in the input buffer if the employee ID is not found. selects one of two possible print lines.

**Complete Code**

col. 2

```plaintext
DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01
REC JOB-START(EMPOSITION) 1 6 2 'START' 'DATE'
PATH OK EMPOSITION EMPLOYEE JOB
KEY EMP-ID-0415 (0003 0023 0100 0301 3200 5100)
01OUT 80 D
013 SELECTED EMPLOYEE DETAIL LIST
010 WK-ID
0151*0100 EMP-NAME-0415 HH 'EMPLOYEE NAME'
0151*020 EMP-ID-0415 FM '9999' HH 'EMPLOYEE' 'ID'
0151*030 JOB-START FD HR
0151*040 TITLE-0440 HH 'JOB' 'TITLE'
0152*010 'EMPLOYEE NOT FOUND '
0152*020 EMP-ID-0415 FM '9999'
017005 IF PATH-ID EQ 'OK' 100
017010 IF PATH-ID NE 'NO'
017020 TAKE 2
017100 IF EMP-ID-0415 EQ WK-ID DROP
017110 MOVE EMP-ID-0415 TO WK-ID
017120 TAKE 1
```

**Result**

<table>
<thead>
<tr>
<th>REPORT NO. 01</th>
<th>SELECTED EMPLOYEE DETAIL LIST</th>
<th>mm/dd/yy</th>
<th>PAGE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE NAME</td>
<td>START</td>
<td>JOB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JENNIFER GARFIELD</td>
<td>0003</td>
<td>82/01/01</td>
<td>MGR INTERNAL SOFTWARE</td>
<td></td>
</tr>
<tr>
<td>KATHERINE O'HEARN</td>
<td>0023</td>
<td>79/05/05</td>
<td>PROGRAMMER/ANALYST</td>
<td></td>
</tr>
<tr>
<td>EDWARD HUTTON</td>
<td>0100</td>
<td>77/09/07</td>
<td>FINANCIAL ANALYST</td>
<td></td>
</tr>
<tr>
<td>BURT LANCHESTER</td>
<td>0301</td>
<td>80/02/03</td>
<td>RAINMAKER</td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE NOT FOUND</td>
<td>3200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE NOT FOUND</td>
<td>5100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Accessing Bill-of-Materials Structures**

You can access database information stored as a bill-of-materials data structure.

The bill-of-materials structure, which can be thought of as a series of nested set relationships, allows a path to be established between one record type and another of the same type. Different set relationships that exist between the record types are used.

**How to Do It**

**Code:**

1. **PATH** parameters using set names that define the level of information to be retrieved

2. **Field names** using level numbers to specify the level of information to be acted upon
What You Can Do 51

Example: Accessing Bill-of-Materials Structures

Objective

This report lists:

- The managers of the company (first level)
- The supervisors reporting to each manager (second level)
- The workers reporting to each supervisor (third level)

Procedure

- **PATHAA** defines a three-level bill-of-materials relationship:
  1. EMPLOYEE STRUCTURE(MANAGES)
  2. EMPLOYEE(REPORTS-TO) STRUCTURE(MANAGES)
  3. EMPLOYEE(REPORTS-TO)

- **Type 5** and **type 7** parameters specify the level from which the names are extracted:
  - EMP-NAME-0415 (1) prints manager's names.
  - EMP-NAME-0415 (2) prints supervisor's names.
  - EMP-NAME-0415 (3) prints worker's names.

Complete Code

col. 2

DATABASE DICTNAME=DOCUDICT
IN DB SS=EMPSS01 $RPT708
PATHAA EMPLOYEE STRUCTURE(MANAGES) EMPLOYEE(REPORTS-TO) STRUCTURE(MANAGES) EMPLOYEE (REPORTS-TO) 01OUT 120
010 MGR '           
010 SUP '           
010 EMP '           
0141*010 'MANAGER'
0141*020 'SUPERVISOR'
0141*030 'WORKER'
0142*001  
0151*010 EMP-NAME-0415 (1) 0152*020 EMP-NAME-0415 (2) 0153*030 EMP-NAME-0415 (3) 0170
10 IF EMP-NAME-0415 (1) NE MGR 100
017020 IF EMP-NAME-0415 (2) NE SUP 200
017030 IF EMP-NAME-0415 (3) NE EMP 300
017 DROP
017100 RELS 1
017200 RELS 2
017300 RELS 3
Accessing Multiple-Member Sets

You can access records in a set that has one owner record and two or more member records.

How to Do It

Code:
1. A PATH parameter placing the **ALL-MEMBERS** option after the name of the owner record

2. An optional SELECT/BYPASS BUFFER parameter after the **INPUT**, **KEYFILE**, and **PATH** parameters that includes the **REC-NAME** keyword

### What You Can Do 52

#### Example: Accessing Multiple-Member Sets

**Objective**

This report lists employee hospital and dental claims. The **HOSPITAL-CLAIM** and **DENTAL-CLAIM** records are owned by the **COVERAGE** record.

**Procedure**

- The **INPUT** parameter specifies an input buffer size of 4000 bytes to accommodate the five record types (**EMPLOYEE**, **COVERAGE**, **HOSPITAL-CLAIM**, **NON-HOSP-CLAIM**, and **DENTAL-CLAIM**), overhead, and DB-keys that are read into the buffer. The NON-HOSPITAL-CLAIM record, while not used for this report, must be included in the buffer size.

- The **ALL-MEMBERS** option on the **PATH** parameter specifies access to each occurrence of the records in the multiple-member set **COVERAGE-CLAIMS**.

- The **SELECT BUFFER** parameter with **REC-NAME** specifies access to two of the three records of the **COVERAGE-CLAIMS** set.

- **REC** parameters redefine the claim date fields as numeric to allow for automatic date formatting. Field redefinition is based upon the starting point in the record, as opposed to the position in the input buffer, by including the record type name immediately after the field name.

**Complete Code**

```
col. 2
IN 4000 DB SS=EMPSS01
PATH01 EMPLOYEE COVERAGE ALL-MEMBERS (COVERAGE-CLAIMS)SELECT BUFFER WHEN REC-NAME EQ ('HOSPITAL-CLAIM' 'DENTAL-CLAIM')REC CLAIM-DATE-0430(HOSPITAL-CLAIM) 1 6 2REC CLAIM-DATE-0405(DENTAL-CLAIM) 1 6 2
0141*010 'PATIENT'
0141*020 'HOSPITAL CLAIMS'
0141*030 'DENTAL CLAIMS'
0141*040 'EMPLOYEE'
0142*001 '
0151*010 PATIENT-NAME-0430
0151*020 CLAIM-DATE-0430 FD
0151*040 EMP-NAME-0415
0152*010 PATIENT-NAME-0405
0152*030 CLAIM-DATE-0405 FD
0152*040 EMP-NAME-0415
017010 IF REC-NAME EQ 'HOSPITAL-CLAIM' 300
017 TAKE 2
017300 TA
LANCESTER
HELOISE HENDON 77/05
```
Retrieving Data With SQL

This section provides many useful procedures and CA Culprit coding techniques to help you write complex reports using SQL and CA Culprit.

- Using the AS Clause to Rename SQL Columns (see page 191)
- Retrieving Floating Point Data from SQL Columns (see page 191)
- Displaying an SQL Column with Data Type BINARY (see page 192)
- How to Test and Display Null Values (see page 194)
- Handling Null Values at Total Time (see page 194)
- Understanding CA Culprit Decimal Point Handling (see page 196)
- Interpreting Common Error Messages in SQL Retrieval (see page 197)

Getting the Exact Column Names for an SQL Table

If you do not know the exact column names in the INV.STOCKS table, try running the following simple CA Culprit report.

```
col. 2
PRO EX=NO
IN DB(Q) DICTIONARY=TSTDICT SCHEMA=INV
SQL SELECT * FROM STOCKS;
0151*001 ' '
```

The EX=NO tells CA Culprit to simply compile the report. No data extraction will be performed.

SQL SELECT * FROM STOCKS tells CA Culprit to describe all of the columns and null indicators for the STOCKS table. The generated REC cards will appear on the INPUT PARAMETER LISTING.

**Generated REC Cards**

<table>
<thead>
<tr>
<th>REC</th>
<th>START</th>
<th>SIZE</th>
<th>TYPE</th>
<th>DP</th>
<th>FIELD-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>00005</td>
<td>3</td>
<td>3</td>
<td></td>
<td>CLOSE_PRICE</td>
</tr>
<tr>
<td>0105</td>
<td>00004</td>
<td>1</td>
<td></td>
<td></td>
<td>CLOSE_PRICE_NULL_IND</td>
</tr>
<tr>
<td>0069</td>
<td>00005</td>
<td>3</td>
<td>3</td>
<td></td>
<td>COMPANY_ID</td>
</tr>
<tr>
<td>0074</td>
<td>00004</td>
<td>1</td>
<td></td>
<td></td>
<td>COMPANY_ID_NULL_IND</td>
</tr>
<tr>
<td>0049</td>
<td>00020</td>
<td>1</td>
<td></td>
<td></td>
<td>COMPANY_NAME</td>
</tr>
<tr>
<td>0078</td>
<td>00010</td>
<td></td>
<td></td>
<td></td>
<td>TRADING_DATE</td>
</tr>
<tr>
<td>0088</td>
<td>00004</td>
<td>1</td>
<td></td>
<td></td>
<td>TRADING_DATE_NULL_IND</td>
</tr>
<tr>
<td>0092</td>
<td>00004</td>
<td>1</td>
<td></td>
<td></td>
<td>VOLUME</td>
</tr>
<tr>
<td>0096</td>
<td>00004</td>
<td>1</td>
<td></td>
<td></td>
<td>VOLUME_NULL_IND</td>
</tr>
</tbody>
</table>

From this report, you can deduce that the STOCKS table was created using the following SQL command:
SQL Command

CREATE TABLE INV.STOCKS
(COMPANY_NAME CHAR(20) NOT NULL,
 COMPANY_ID CHAR(5),
 TRADING_DATE DATE,
 VOLUME INTEGER,
 CLOSE_PRICE DECIMAL(9,3));

Using the AS Clause to Rename SQL Columns

Long SQL column names can present a problem to CA Culprit, since you can create a null indicator name by appending _NULL_IND to the end of the previously generated REC card name. You can avoid potential errors by using an AS clause to assign an alias to each SQL column.

AS Clause Example

col. 2

INPUT DB(Q) DICTIONARY=TSTDICT SCHEMA=INV
SQL SELECT "EMP-ID-0415" AS "ID",
* "EMP-FIRST-NAME-0415" AS "FNAME",
* "EMP-LAST-NAME-0415" AS "LNAME"
* FROM EMPLOYEE
* WHERE "EMP-ID-0415" = 2000
* ORDER BY ID;
0151*001 ID
0151*002 FNAME
0151*003 LNAME

Please note that you may not use the alias names ID, FNAME, and LNAME on most clauses in the SQL statement (Aliases are allowed on the order by clause). However, the aliases will be used to generate REC parameters, and must be used on all subsequent CA Culprit statements.

Generated REC Cards

******
REC START SIZE TYPE DP FIELD-NAME
******
REC 0057 00015 FNAME
REC 0072 00004 1 FNAME_NULL_IND
REC 0049 00004 2 ID
REC 0053 00004 1 ID_NULL_IND
REC 0076 00020 LNAME
REC 0096 00004 1 LNAME_NULL_IND

Retrieving Floating Point Data from SQL Columns

Although CA Culprit does not support real and double precision as REC parameter data types, there is a simple coding technique to work around the deficiency.

CA Culprit will treat real SQL columns as four byte alphanumeric fields. Double precision columns will be treated as eight byte alphanumeric fields. 16 byte packed decimal work fields can be defined, and CULLUS36 can be used to convert the floating point values to packed decimal. The CA Culprit program below illustrates the technique.
Converting to Packed Decimal

```
col. 2

IN DB(Q)  DICTIONARY=TSTDICT  SCHEMA=INV
SQL SELECT * FROM FLOAT_TABLE;
$  CA Culprit will generate the REC cards shown below.
$REC REAL DATA  49 4  $ GENERATED
$REC DOUBLE_DATA  53 8  $ GENERATED
$
  010  WK_REAL  0.000000
  010  WK_DOUBLE  0.000000000000
  010  RETCODE '
  0151*001  WK_REAL HF
  0151*002  WK_DOUBLE HF
  017   CALL US36(REAL_DATA, 'S', WK_REAL, 6, RETCODE)
  017   IF RETCODE NE ' ' 200
  017   CALL US36(DOUBLE_DATA, 'D', WK_DOUBLE, 12, RETCODE)
  017   IF RETCODE NE ' ' 200
  017   TAKE
  017200  CALL US48(' ERROR IN FLOATING POINT CONVERSION%%')
  017   DROP
```

The arguments for CULLUS36 are described below:

**US36 Arguments**

- **ARG1 field-name**
  Name of input field containing the floating point value

- **ARG2 indicator**
  - 'S' = Single Precision:
  - 'D' = Double Precision

- **ARG3 result-field**
  Name of 16 byte packed decimal work field

- **ARG4 decimal-pts**
  Integer indicating number of decimal places in result field

- **ARG5 return-code**
  Name of four byte alpha work field

Displaying an SQL Column with Data Type BINARY

The BINARY data type is a CA IDMS/DB extension to ANSI SQL. It is used to store hexadecimal literals up to 32,760 bytes in length.

During retrieval, CA Culprit treats columns with SQL data type BINARY as an alphanumeric field. To display the binary data on a report, use CULLUS31 to obtain a hexadecimal representation of the actual data.

In the example below, the SYSTEM.CONSTRAINT table has a column named REFCOLUMNS defined as BINARY(64). To print the data in hex on our report, displaying 10 bytes on each detail line.

**Report Output**
Report Syntax

col. 2

PRO USER=MZC PW=MZC
IN 300 DB(O) SCHEMA=SYSTEM DICTIONARY=TSTDICT
REC HEX REF COL 150 128
REC HEX REF COL1 150 20
REC HEX REF COL2 170 20
REC HEX REF COL3 190 20
REC HEX REF COL4 210 20
REC HEX REF COL5 230 20
REC HEX REF COL6 250 20
REC HEX REF COL7 270 20
SQL SELECT NAME, REFCOLUMNS FROM CONSTRAINT;
01 OUT 80 D
013 CONTRAINT TABLE REPORT
0151*001 NAME HH 'CONSTRAINT' 'NAME'
0151*002 'BYTES 1 - 10'
0151*003 HEX REF COL1 HH 'REFCOLUMNS' '01......05......10'
0152*002 'BYTES 11 - 20'
0152*003 HEX REF COL2
0153*002 'BYTES 21 - 30'
0153*003 HEX REF COL3
0154*002 'BYTES 31 - 40'
0154*003 HEX REF COL4
0155*002 'BYTES 41 - 50'
0155*003 HEX REF COL5
0156*002 'BYTES 51 - 60'
0156*003 HEX REF COL6
0157*002 'BYTES 61 - 64'
017 CALL US31(REFCOLUMNS, 64, HEX REF COL)

The arguments for CULLUS31 are described below:

**US31 Arguments**

Error: Don’t know what to do with column width of "20," (3).

- **ARG1 field-name**
  Name of input field containing the value to be converted

- **ARG2 length**
  Numeric literal or 8 byte work field containing the length of the input field. For release 12.0, this value must be in the range 1 through 127.

- **ARG3 result-field**
  Name of the output field. Result-field must be twice as long as the length of the input field.
How to Test and Display Null Values

The sales table below contains daily sales data for specific products.

Table Definition

CREATE TABLE INV.SALES
(PRODUCT CHAR(20) NOT NULL,
SALES_DATE DATE NOT NULL,
UNITS_SOLD INTEGER);

Notice that the UNITS_SOLD field may contain null values. A null value means "We don't know how many units we sold". That is much different than the value zero which means "We didn't sell any product on this date".

Here's how to create a detail report that displays null values on the detail line.

Detail Report

col. 2

IN DB(Q) DICTIONARY=TSTDICT SCHEMA=INV
SQL SELECT * FROM SALES;
01OUT 80 D
0151*001 PRODUCT HF
0151*002 SALES_DATE HF
0151*003 UNITS_SOLD SZ=6 F- HF
0152*001 PRODUCT
0152*002 SALES_DATE
0152*003 '---null---'
017 IF UNITS_SOLD_NULL_IND = -1 200
017100 TAKE 1
017200 TAKE 2

The UNITS_SOLD_NULL_IND will be set to minus one to indicate a null value. A zero value indicates that UNITS_SOLD is not null.

Report Output

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>SALES_DATE</th>
<th>UNITS_SOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHING MACHINES</td>
<td>yyyy-mm-dd</td>
<td>14</td>
</tr>
<tr>
<td>Washing Machines</td>
<td>yyyy-mm-dd</td>
<td>null---</td>
</tr>
<tr>
<td>WASHING MACHINES</td>
<td>yyyy-mm-dd</td>
<td>0</td>
</tr>
<tr>
<td>WASHING MACHINES</td>
<td>yyyy-mm-dd</td>
<td>3</td>
</tr>
<tr>
<td>WASHING MACHINES</td>
<td>yyyy-mm-dd</td>
<td>12</td>
</tr>
</tbody>
</table>

Handling Null Values at Total Time

You can enhance the previous example to produce weekly summary information for each product. You also want to print the average number of units sold per day. The final report should look like this:

Report Output
### WHERE Clause

First, add a WHERE clause to select data for one week:

```
SQL SELECT * FROM SALES
* WHERE SALES_DATE >= 'yyyy-mm-dd' AND
* SALES_DATE <= 'yyyy-mm-dd';
```

### Subtotals

Next, add a title parameter and a SORT parameter to request subtotals by product. A COUNT field has been initialized to one. It will be used to determine how many detail records have been processed for each group. You can automatically accumulate subtotals for COUNT and UNITS_SOLD_NULL_IND if they appear on detail lines. However, since you don't want the detail values to print, use print position zero:

```
col. 2
01 SORT PRODUCT 0, SALES_DATE
010 COUNT 1
013 WEEKLY SALES BY PRODUCT
0151*000 COUNT
0151*000 UNITS_SOLD_NULL_IND
```

Based on the information in the report shown above, the total-time values for PRODUCT, COUNT, and UNITS_SOLD_NULL_IND are:

### Subtotal Values

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>COUNT</th>
<th>UNITS_SOLD_NULL_IND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washing Machines</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>Clothes Driers</td>
<td>5</td>
<td>-2</td>
</tr>
</tbody>
</table>
These values let you determine how many UNITS_SOLD for each product are not null. This in turn provides you with the divisor you need to compute the average daily sales for each product. The total-time logic is shown below:

**Total-time Logic**

<table>
<thead>
<tr>
<th>col. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 NOT NULL_COUNTER 0</td>
</tr>
<tr>
<td>010 AVERAGE 0.00</td>
</tr>
<tr>
<td>018 IF LEVL = 2 DROP</td>
</tr>
<tr>
<td>018 COMPUTE COUNT + UNITS_SOLD NULL_IND NOT NULL_COUNTER</td>
</tr>
<tr>
<td>018 IF NOT NULL_COUNTER = 0 400</td>
</tr>
<tr>
<td>018 COMPUTE UNITS_SOLD / NOT NULL_COUNTER AVERAGE</td>
</tr>
<tr>
<td>018 TAKE (1,2,3)</td>
</tr>
<tr>
<td>018400 TAKE (1,4,5)</td>
</tr>
<tr>
<td>0161*003 '-----------'</td>
</tr>
<tr>
<td>0162*002 'Total Weekly Sales'</td>
</tr>
<tr>
<td>0162*003 UNITS_SOLD SZ=6 F.</td>
</tr>
<tr>
<td>0163*002 'Average Daily Sales'</td>
</tr>
<tr>
<td>0163*003 AVERAGE SZ=6 F.</td>
</tr>
<tr>
<td>0164*002 'Total Weekly Sales'</td>
</tr>
<tr>
<td>0164*003 '---null---'</td>
</tr>
<tr>
<td>0165*002 'Average Daily Sales'</td>
</tr>
<tr>
<td>0165*003 '---null---'</td>
</tr>
</tbody>
</table>

**Understanding CA Culprit Decimal Point Handling**

In our STOCKS table example, the CLOSE_PRICE field is defined as having three implied decimal digits.

**Table Definition**

```sql
CREATE TABLE INV.STOCKS
(COMPANY_NAME CHAR(20) NOT NULL,
 COMPANY_ID CHAR(5),
 TRADING_DATE DATE,
 VOLUME INTEGER,
 CLOSE_PRICE DECIMAL(9,3));
```

This allows you to represent 1/8 point fractions as a decimal value. However when formatting reports, you often want to print the value as a money field, dollars and cents. This means converting the number from DP=3 to DP=2.

To obtain the correctly rounded value, move the CLOSE_PRICE field to a work field, WK_CLOSE defined with two decimal points. Then, reference WK_CLOSE on the detail line. The report below shows why:

**Report Syntax**

```sql
IN DB(Q) DICTIONARY=TSTDICT SCHEMA=INV
SQL SELECT * FROM STOCKS;
010 OUT 80 D
010 WK CLOSE 0.00
0151*001 COMPANY_NAME HF
0151*002 CLOSE_PRICE HF
```
As you can see, coding CLOSE_PRICE DP=2 on the edit parameter changes the location of the implied decimal point, and produces erroneous results. To insure correct rounding, move the value to a work variable that was defined with the correct number of decimal points.

### Interpreting Common Error Messages in SQL Retrieval

For retrieval processing, SQL Prepare and Describe commands are issued during the pre-compile phase of CA Culprit. If any errors are detected, you will receive the following messages on your Sequential Parameter Listing.

**Messages 55, 56, and 57 print the error codes and text contained in the SQL Communications Area (SQLCA).**

See the [SQL Communication Area](https://docops.ca.com/display/IDMSCU/SQL+Communication+Area), for more information about the SQLCA error code fields.

Any E or F level errors associated with the SQL statement will prevent CA Culprit from generating REC parameters to describe the SQL columns and null indicators. That means you will probably receive messages like "Field Not Defined" on the Input Parameter Listing. Ignore these extraneous errors for now. They will probably go away as soon as you correct the SQL statement.

During the extract phase of CA Culprit, you issue SQL commands to perform the bulk fetch operation. Any errors at this time appear under Run Time Messages. The SQL error codes and text are be formatted exactly as shown above.
Creating New SQL Data Tables

This section illustrates many CA Culprit coding techniques used to create new CA IDMS SQL tables.

- Converting ASF Tables to SQL Tables (see page 198)
- Assigning Null Values to an SQL Column (see page 199)
- How CA Culprit Handles Data Insertion Errors (see page 200)

Create Table Syntax Generated by CA Culprit

Information from the OUTPUT parameter and the EDIT parameters is used to generate SQL Create Table syntax during the compile phase of CA Culprit. To help illustrate this correspondence, a CA Culprit program fragment is included below along with the SQL commands that are generated:

**CA Culprit Syntax**

```
col. 2
010 ZERO 0
01OUT SQLTABLE=INVENTORY TYPE=CREATE
* DICTIONARY=TSTDICT SCHEMA=INV
0151*010 WAREHOUSE SZ=10 NOT NULL
0151*020 PRODUCT CODE SZ=10 NOT NULL
0151*030 UNITS ON HAND FP SZ=4
0151*031 ZERO FB SZ=4
```

**Compile-phase SQL Syntax**

```
CONNECT TO TSTDICT;
SET SESSION CURRENT SCHEMA "INV";
CREATE TABLE "INVENTORY"
("WAREHOUSE" CHAR(0010) NOT NULL,
"PRODUCT CODE" CHAR(0010) NOT NULL,
"UNITS ON HAND" DECIMAL(0007,0000));
COMMIT RELEASE;
```

These SQL commands are generated and issued directly by CA Culprit; nothing is required of the user.

Converting ASF Tables to SQL Tables

The job of converting a ASF table to an SQL table is really quite simple using CA Culprit. Simply read the ASF table as input, and create an OUTPUT SQL table with TYPE=CREATE.

**CA Culprit Syntax**

```
col. 2
PRO USER=MZC PW=
INPUT TABLE=INVENTORY TYPE=COPY CATALOG=ASFDICT
01OUT SQLTABLE=INVENTORY TYPE=CREATE
* DICTIONARY=TSTDICT SCHEMA=INV
```
Assigning Null Values to an SQL Column

The technique for assigning null values to an SQL column is quite simple: set the column null indicator to minus one (-1).

Consider the example of a phonelist. For the WORK_PHONE field, you could use blanks to mean "This person has no work phone" and null to mean "He may have a work phone, but we don't know the number".

On the sequential input file, null values are represented by a question mark (?) in the WORK_PHONE field. The example below reads the input file, converts question marks to null values, and creates a new SQL table containing the phonelist:

Code Example

<table>
<thead>
<tr>
<th>col. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN 80</td>
</tr>
<tr>
<td>REC FIRST_NAME 1 12</td>
</tr>
<tr>
<td>REC LAST_NAME 13 15</td>
</tr>
<tr>
<td>REC IN_HOMEPH 28 12</td>
</tr>
<tr>
<td>REC IN_WORKPH 40 12</td>
</tr>
<tr>
<td>010 HOME_PHONE ' '</td>
</tr>
<tr>
<td>010 WORK_PHONE ' '</td>
</tr>
<tr>
<td>010 SPACES ' '</td>
</tr>
<tr>
<td>010 HOME_PHONE NULL_IND 0</td>
</tr>
<tr>
<td>010 WORK_PHONE NULL_IND 0</td>
</tr>
<tr>
<td>01OUT SQLTABLE=PHONELIST TYPE=CREATE</td>
</tr>
<tr>
<td>* DICTIONARY=TSTDICT SCHEMA=DEMO</td>
</tr>
<tr>
<td>0151*010 FIRST_NAME NOT NULL WITH DEFAULT</td>
</tr>
<tr>
<td>0151*020 LAST_NAME NOT NULL WITH DEFAULT</td>
</tr>
<tr>
<td>0151*030 HOME_PHONE</td>
</tr>
<tr>
<td>0151*031 HOME_PHONE NULL_IND FB SZ=4</td>
</tr>
<tr>
<td>0151*040 WORK_PHONE</td>
</tr>
<tr>
<td>0151*041 WORK_PHONE NULL_IND FB SZ=4</td>
</tr>
<tr>
<td>017 IF IN_HOMEPH = '?' 100</td>
</tr>
<tr>
<td>017 MOVE IN_HOMEPH TO HOME_PHONE $ Not Null - Valid Number</td>
</tr>
<tr>
<td>017 MOVE 0 TO HOME_PHONE NULL_IND</td>
</tr>
<tr>
<td>017 B 200</td>
</tr>
<tr>
<td>017100 MOVE SPACES TO HOME_PHONE $ Assign Null Value</td>
</tr>
<tr>
<td>017 MOVE -1 TO HOME_PHONE NULL_IND</td>
</tr>
<tr>
<td>017$</td>
</tr>
<tr>
<td>017200 IF IN_WORKPH = '?' 300</td>
</tr>
</tbody>
</table>
How CA Culprit Handles Data Insertion Errors

During the output phase of CA Culprit, it is possible to encounter errors during the SQL bulk insert. If this occurs, CA Culprit produces a diagnostic dump of the output data record and continue processing, as shown below:

**Output Phase Diagnostic Error**

```
E C760001 THE FOLLOWING ROWS WERE NOT STORED DUE TO INSERT ERRORS:

<table>
<thead>
<tr>
<th>OUTPUT RECORD NUMBER</th>
<th>ERROR CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0005</td>
<td>1023</td>
</tr>
</tbody>
</table>

ZONE 4444444444FFFF000130000CCCCC0000
DIGIT 0000000000FFFF0002C0000123450000
```

In the above example, the **Output Record Number** indicates that the fifth row being written contained an error.

The **Record Layout** provides a complete vertical hex dump of the CA Culprit output buffer to assist in debugging. The **Error Code** 1023 indicates a constraint violation. This value was obtained from the SQLCERC field in the SQL Communications Area. A list of common error codes is given below:

**SQLCERC Codes**

- **1008**
  - No storage to load data

- **1010**
  - Load of Table Deadlocked

- **1023**
  - Constraint Violation

- **1025**
  - Data Exception

- **1026**
  - Data Conversion Error

- **1051**
  - Cardinality Violation

- **1058**
  - Duplicate Key Violation
Adding, Replacing, and Dropping Data on SQL Tables

This section illustrates many CA Culprit coding techniques used to add, replace, and drop data in CA IDMS SQL tables.

- Getting the Exact Syntax for Updating a SQL Table (see page 202)
- Drop Table Example (see page 202)

Updating SQL Columns of Type DATE

SQL Tables that contain columns of type DATE may be updated using the ADD or REPLACE functions. Data to be loaded into a DATE column should be stored in a 10 byte alphanumeric variable or literal. The format of the date value should be 'YYYY-MM-DD'. CA IDMS/DB is responsible for converting this date value into its own internal DATE format.

Existing Table Definition

```sql
CREATE TABLE STOCKS
((COMPANY_NAME CHAR(20) NOT NULL,
  COMPANY_ID CHAR(5),
  TRADING_DATE DATE,
  VOLUME INTEGER,
  CLOSE_PRICE DECIMAL(9,3));
```

Report Syntax

```sql
IN 80
REC COMPANY_NAME 1 20
REC COMPANY_ID 21 5
REC VOLUME 26 8 2
REC CLOSE_PRICE 34 9 2 DP=3
010 TRADING_DATE 'yyyy-mm-dd' 010UT SQLTABLE=STOCKS TYPE=ADD
* DICTIONARY=TSTDICT SCHEMA=INV
010 ZERO 0
0151*010 COMPANY_NAME
0151*020 COMPANY_ID
0151*021 ZERO FB SZ=4 0151*030 TRADING_DATE 0151*031 ZERO FB SZ=4
0151*040 VOLUME FB SZ=4
0151*041 ZERO FB SZ=4
0151*050 CLOSE_PRICE FP SZ=5 DP=3
0151*051 ZERO FB SZ=4
```
Getting the Exact Syntax for Updating a SQL Table

In order to update an existing SQL table, it is necessary to know the exact order of the columns, the data types, and the existence of null indicators. Fortunately, there is an easy technique to get the exact CA Culprit syntax needed to update an SQL table. Here’s how:

Code a small CA Culprit program using TYPE=ADD on the OUTPUT parameter. Deliberately make a mistake on the data type or size of an EDIT parameter. This produces E level errors during the compile phase of CA Culprit.

Report Syntax

```
01OUT SQLTABLE=STOCKS TYPE=ADD
     DICTIONARY=TSTDICT SCHEMA=INV
0151*001 ID FW $ Deliberate Error!
```

At the end of the compile phase, error messages are produced on the Input Parameter Listing. Message C700071 provides you with sample EDIT parameters to describe the output SQL table exactly. Use this as a template for coding your CA Culprit program.

```
********
EDIT   LINE CC COLUMN VALUE OR FIELD-NAME AND EDIT OPTIONS
********
01   1 *001 ID FW
E C300064 FORMAT CODE IS INVALID
E C300023 FIELD SIZE IS INVALID
```

I C700071 USE THE SAMPLE EDIT PARAMETERS BELOW TO CORRECT YOUR PROGRAM

```
0151*010 COMPANY_NAME  SZ=0020
0151*020 COMPANY_ID  S2=0005
0151*021 COMPANY_ID_NULL_IND  FB  S2=0004
0151*030 TRADING_DATE  S2=0010
0151*031 TRADING_DATE_NULL_IND  FB  S2=0004
0151*040 VOLUME  FB  S2=0004
0151*041 VOLUME_NULL_IND  FB  S2=0004
0151*050 CLOSE_PRICE  FP  S2=0005 DP=0003
0151*061 CLOSE_PRICE_NULL_IND  FB  S2=0004
```

Drop Table Example

The code needed to delete an SQL table definition using CA Culprit is given below:

```
CA Culprit Syntax
```
The Personnel File Description

The Commonwealth Corporation personnel file, which is used in examples in this manual, is described fully in the figure below:

The Commonwealth Corporation Personnel File Description

| RECORD SIZE | 200 bytes |
| BLOCK SIZE  | 400 bytes |
| RECORD FORMAT | FIXED (F) |
| FILE TYPE    | SEQUENTIAL (PS) |

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Start Position</th>
<th>Length (bytes)</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE-ID</td>
<td>1</td>
<td>4</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>EMPLOYEE-NAME</td>
<td>5</td>
<td>25</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>FIRST-NAME</td>
<td>5</td>
<td>10</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>LAST-NAME</td>
<td>15</td>
<td>15</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>STREET</td>
<td>30</td>
<td>20</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>CITY</td>
<td>50</td>
<td>15</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>STATE</td>
<td>65</td>
<td>2</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>ZIP-CODE</td>
<td>67</td>
<td>5</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>PHONE</td>
<td>72</td>
<td>10</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>STATUS</td>
<td>82</td>
<td>2</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>SS-NUMBER</td>
<td>84</td>
<td>9</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>START-MONTH</td>
<td>93</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>START-DAY</td>
<td>95</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>START YEAR</td>
<td>97</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>TERMINATION-MONTH</td>
<td>99</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>TERMINATION-DAY</td>
<td>101</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>TERMINATION-YEAR</td>
<td>103</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>BIRTH-MONTH</td>
<td>105</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>BIRTH-DAY</td>
<td>107</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>BIRTH-YEAR</td>
<td>109</td>
<td>2</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>DEPT-ID</td>
<td>111</td>
<td>4</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>DEPT-NAME</td>
<td>115</td>
<td>45</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>SALARY-AMOUNT</td>
<td>160</td>
<td>5</td>
<td>Packed Decimal</td>
</tr>
<tr>
<td>JOB-ID</td>
<td>167</td>
<td>4</td>
<td>Zoned decimal</td>
</tr>
<tr>
<td>TITLE</td>
<td>171</td>
<td>20</td>
<td>Alphanumeric</td>
</tr>
</tbody>
</table>
Coding for Efficiency

Efficient coding prevents excessive use of system resources by minimizing overhead and input/output (I/O) processing wherever possible. Factors influencing coding choices include:

- The number of records processed
- The number of reports in a single run
- The number of times an instruction is executed

This section presents some of the most common coding alternatives that affect the efficiency of a CA Culprit program.

- Eliminating Unnecessary Records (see page 204)
- When to Use SELECT and When to Use Type 7 Logic (see page 205)
- Avoiding Unnecessary Logic (see page 205)
- Combining Tasks for Multiple Reports (see page 205)
- Avoiding Unwanted Buffer Dumps (see page 206)
- Creating Subtotal Work Fields for Presorted Input Files (see page 206)
- CA Culprit JCL Considerations (see page 207)
- Miscellaneous Hints (see page 208)

Eliminating Unnecessary Records

Eliminating the most frequently occurring record first is most efficient. For example:

**Elimination of the Most Frequent Record**

<table>
<thead>
<tr>
<th>A occurs 2,000 times</th>
<th>B occurs 1,000 times</th>
<th>C occurs 50 times</th>
</tr>
</thead>
<tbody>
<tr>
<td>017 COMPANY EQ 'A' DROP (Tests 3,050 records, drops 2,000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017 COMPANY EQ 'B' DROP (Tests 1,050 records, drops 1,000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017 COMPANY EQ 'C' DROP (Tests 50 records, drops 50)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4,150 tests required to drop all records.

or

Eliminating records randomly is least efficient. For example:

**Elimination of Random Records**

<table>
<thead>
<tr>
<th>A occurs 50 times</th>
<th>B occurs 1,000 times</th>
<th>C occurs 2,000 times</th>
</tr>
</thead>
<tbody>
<tr>
<td>017 COMPANY EQ 'A' DROP (Tests 3,050 records, drops 50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017 COMPANY EQ 'B' DROP (Tests 3,000 records, drops 1,000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017 COMPANY EQ 'C' DROP (Tests 2,000 records, drops 2,000)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8,050 tests required to drop all records.
When to Use SELECT and When to Use Type 7 Logic

Selection criteria can be applied to the entire run with the SELECT or BYPASS parameter or stated in the procedure logic for each report:

```
SELECT PRODUCT EQ 'CA-CULPRIT'
```

```
017    PRODUCT EQ 'CA-CULPRIT'
027    PRODUCT EQ 'CA-CULPRIT'
037    PRODUCT EQ 'CA-CULPRIT'
```

Although it is quicker to write, the SELECT parameter uses more processing overhead than type 7 logic. SELECT only becomes efficient when six or more reports are batched together. Type 7 logic is more efficient than SELECT for a smaller number of reports.

Avoiding Unnecessary Logic

Obviously, streamlined code is more efficient than code that generates unnecessary processing.

In the example below, a call for the current date is executed for each record in the input file by processing user module US10:

```
017    CALL US10 ( 2 CURRENT-DATE )
```

The code below shows one method of calling the user module one time for the report, thereby avoiding unnecessary processing. The call executes only when the switch value is Y:

```
010    FIRST-TIME-SW 'Y'
       ...
017    FIRST-TIME-SW EQ 'N' 010
017    MOVE 'N' FIRST-TIME-SW
017    CALL US10 ( 2 CURRENT-DATE ) $Executes only on first record 017010 ...
```

Combining Tasks for Multiple Reports

Reports created from the same data source can often be grouped together to improve processing efficiency. The most common coding strategies are to:

- **Batch related reports together** to allow CA Culprit to read the data file once for all the reports. Code reports in ascending numerical sequence.

- **Use global work fields** to allow all the reports in the run access to work field contents. Calculation results can be shared between reports in this manner.

- **Use exact positioning** for edit parameters wherever possible; for example 01510045.
Keep relative column numbers low (*0002, *0004, *0006) to allow for later insertions and still minimize the number of work areas (buckets) CA Culprit must set up. CA Culprit uses the highest relative column number to allocate storage for some internal tables:

*0002
*0004 Allocates a table with 6 entries
*0006
*0100
*0200 Allocates a table with 300 entries

Avoiding Unwanted Buffer Dumps

Buffer dumps should be used for debugging purposes only. Avoid unnecessary dumps by observing the following precautions:

- Test the value of any input field that could be zero (0) and used as a divisor. Code should include a branch around the division when the divisor equals zero (0). The result should be set to zero.

- Use a work field in place of an input field that could be nonnumeric and used in calculations. Move the work field as the program logic dictates.

- Sort match files in the same sequence. Match keys must be in ascending sequence.

- Use the BRANCH to HEAD instruction only when absolutely necessary to avoid the creation of additional records in the extract file.

Creating Subtotal Work Fields for Presorted Input Files

If your input file is presorted, it is more efficient to use type 7 logic to accumulate totals in a work field rather than allow CA Culprit to automatically total the fields each time a control break occurs on a SORT parameter.

Two different techniques for reporting total amounts by city are shown below:

1. This example uses type 7 logic to accumulate the total amount in a work field until the value of CONTROL-FLD changes. This method is extremely efficient, since NOSORT is required and only 7 extract records will be written to the extract file.

   IN 200
   REC CONTROL-FLD 1 4
   REC INPUT-FLD 5 9 2
   0151*002 SAVE-FLD HH 'CONTROL'
   0151*004 BUCKET HH 'AMOUNT'
   010 SAVE-FLD '
   010 BUCKET 0
   017 EOF = TAKE $Print last work field
   017 CONTROL-FLD NE SAVE-FLD 100 $Control change?
   017 INPUT-FLD + BUCKET BUCKET $Accumulate
   017 DROP $Get next record
   017100 RELS $Previous control fld
   017 MOVE CONTROL-FLD TO SAVE-FLD $Save current information
1. Always seek to limit I/O activity by reducing the size of the extract file. DROP records in type 7 logic whenever possible.

2. Avoid coding unnecessary SORT parameters. If your input file is presorted and you want control breaks, use the NOSORT option.

3. CPU activity is less expensive than I/O activity.

CA Culprit JCL Considerations

Defined record lengths, the amount of space allocated, and blocking factors affect CA Culprit’s use of system resources. The following specifications are recommended:

- On SYS005 (parameter file), the record length must be 320 bytes. A quarter or half track is recommended for the block size.

- On SYS006 (extract file) and SYS008 (NOSORT file), record length should be at least 256 bytes (larger for nonprint records). Block size should be equal to a half or full track, large enough to avoid overrides. Contiguous space is the most efficient, but not always possible.

- On SYS007 (SORT parameters), the record length should be 80 bytes and block size should be a quarter track.

When you are coding a CA Culprit report, keep in mind these efficiency guidelines:
Miscellaneous Hints

The following suggestions may prove helpful:

- Document everything the code does. Use a dollar sign ($) for comments placed in the code.
- Use test files when developing routines that will run against large files.
- Use copied code syntax (USE, =COPY, =MACRO) to read INPUT and REC parameters into code.

Precoded CA Culprit User Modules

The following precoded CA Culprit user modules are provided on the CA Culprit installation media. These modules are compiled and link edited at installation time:

- Procedure Modules (see page 208)
- Output Modules (see page 210)

Input Modules

- CULSPAN
  Reads an input file containing spanned records in a z/VSE environment

- CULLVSAM
  Reads key or entry-sequenced VSAM files sequentially or directly

Procedure Modules

Procedure Modules

- CULLUS00
  Provides an interface between CA Culprit and user-written modules

- CULLUS01
  Retrieves records from a sequential file (z/OS and z/VM)

- CULLUS10
  Returns the system time and date to a user-defined field

- CULLUS11
  Converts a Julian date (YYDDD) to a Gregorian date (MMDDYY)

- CULLUS12
  Converts any century date to a user-specified format.
- **CULLUS14**
  Converts a Gregorian date (MMDDYY) to a Julian date (YYDDD)

- **CULLUS15**
  Converts a date in any format to a user-specified format

- **CULLUS22**
  Retrieves records from ISAM files and delivers them to a specified location

- **CULLUS25**
  Retrieves records from VSAM files and delivers them to a specified location

- **CULLUS29**
  Formats a vertical hexadecimal dump of all or part of a record, input field, or workfield

- **CULLUS31**
  Displays an input or work field in hexadecimal representation

- **CULLUS33**
  Converts a numeric field in packed signed decimal format to binary format

- **CULLUS34**
  Converts a packed signed numeric field to zoned decimal format

- **CULLUS35**
  Represents bit settings in display format

- **CULLUS36**
  Converts floating point values to decimal integers

- **CULLUS37**
  Converts a doubleword binary number to an eight-byte packed decimal number

- **CULLUS40**
  Enables z/VSE users to send messages to the console operator

- **CULLUS43**
  Moves data from a field known to CA Culprit to a position in another field

- **CULLUS45**
  Performs multiple MOVE operations on data

- **CULLUS46**
  Searches a string to locate a position of a specified character

- **CULLUS48**
  Adds a user-written message to the runtime messages section of a CA Culprit job

- **CULLUS50**
  Converts a binary string to a string of characters or workfields
Output Modules

Output Modules

- **CULEDUMP**
  Print an output line in vertical or horizontal dump format

- **CULELABL**
  Creates labels for output

- **CULEMLIN**
  Prints multiple output lines and multiple logical footer lines

- **CULEVSAM**
  Writes records to a user-defined VSAM file

- **CULEPOWR**
  Segments reports in a CA Culprit job through VSE/POWER

Employee Database Subschema

The employee database is the input for the examples and programs in this manual that reference database records or logical records. The remainder of this section lists the fields defined for each record in the database.

Subschema Listing

Below is a listing of the Employee database:

```
SCHEMA=EMPSCHM  VERSION=(100)
SUBSCHEMA=(EMPSS01)
```
RECORD NAME........ COVERAGE RLGTH= 36
RECORD VERSION..... 0100 DLGTH= 20
RECORD ID.......... 0400 KLGTH= 16
RECORD LENGTH...... FIXED DSTRT= 16
LOCATION MODE...... VIA SET EMP-COVERAGE DISPLACEMENT 0000 PAGES
WITHIN............. INS-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS... SET............. TYPE........ NEXT PRIOR OWNER
EMP-COVERAGE MEMBER 1 2 3 COVERAGE-CLAIMS OWNER 4
DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 SELECTION-DATE-0400 DISPLAY 1 8
03 SELECTION-YEAR-0400 DISPLAY 9(4) 1 4
03 SELECTION-MONTH-0400 DISPLAY 9(2) 5 2
03 SELECTION-DAY-0400 DISPLAY 9(2) 7 2
02 TERMINATION-DATE-0400 DISPLAY 9(4) 9 8
03 TERMINATION-YEAR-0400 DISPLAY 9(4) 9 4
03 TERMINATION-MONTH-0400 DISPLAY 9(2) 13 2
03 TERMINATION-DAY-0400 DISPLAY 9(2) 15 2
02 TYPE-0400 DISPLAY X 17 1
88 MASTER-0400 COND 'M' 17
88 FAMILY-0400 COND 'F' 17
88 DEPENDENT-0400 COND 'D' 17
02 INS-PLAN-CODE-0400 DISPLAY X(3) 18 3
88 GROUP-LIFE-0400 COND '001' 18
88 HMO-0400 COND '002' 18
88 GROUP-HEALTH-0400 COND '003' 18
88 GROUP-DENTAL-0400 COND '004' 18

RECORD NAME........ DENTAL-CLAIM RLGTH= 944
RECORD VERSION..... 0100 DLGTH= 936
RECORD ID.......... 0405 KLGTH= 8
RECORD LENGTH...... VARIABLE DSTRT= 12
MINIMUM ROOT....... 132 CHARACTERS
MINIMUM FRAGMENT... 932 CHARACTERS
LOCATION MODE...... VIA SET COVERAGE-CLAIMS DISPLACEMENT 0000 PAGES
WITHIN............. INS-DEMO-REGION OFFSE 5 PGS FOR 20 PGS
DBKEY POSITIONS... (FRAGMENT CHAIN) INTNL 2
COVERAGE-CLAIMS MEMBER 1...
DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 CLAIM-DATE-0405 DISPLAY 1 8
03 CLAIM-YEAR-0405 DISPLAY 9(4) 1 4
03 CLAIM-MONTH-0405 DISPLAY 9(2) 5 2
03 CLAIM-DAY-0405 DISPLAY 9(2) 7 2
02 PATIENT-NAME-0405 DISPLAY 9 25
03 PATIENT-FIRST-NAME-0405 DISPLAY X(10) 9 10
03 PATIENT-LAST-NAME-0405 DISPLAY X(15) 19 15
02 PATIENT-BIRTH-DATE-0405 DISPLAY 34 8
03 PATIENT-BIRTH-YEAR-0405 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-MONTH-0405 DISPLAY 9(2) 38 2
03 PATIENT-BIRTH-DAY-0405 DISPLAY 9(2) 40 2
02 PATIENT-SEX-0405 DISPLAY X 42 1
02 RELATION-TO-EMPLOYEE-0405 DISPLAY X(10) 43 10
02 DENTIST-NAME-0405 DISPLAY 53 25
03 DENTIST-FIRST-NAME-0405 DISPLAY X(10) 53 10
03 DENTIST-LAST-NAME-0405 DISPLAY X(15) 63 15
02 DENTIST-ADDRESS-0405 DISPLAY 78 46
03 DENTIST-STREET-0405 DISPLAY X(28) 78 20
03 DENTIST-CITY-0405 DISPLAY X(15) 98 15
03 DENTIST-STATE-0405 DISPLAY X(2) 113 2
03 DENTIST-ZIP-0405 DISPLAY 115 9
04 DENTIST-ZIP-FIRST-FIVE-0405 DISPLAY X(5) 115 5
04 DENTIST-ZIP-LAST-FOUR-0405 DISPLAY X(4) 120 4
02 DENTIST-LICENSE-NUMBER-0405 DISPLAY 9(6) 124 6
02 NUMBER-OF-PROCEDURES-0405 COMP 9(2) 130 2
02 FILLER DISPLAY X 132 1
02 DENTIST-CHARGES-0405 DISPLAY OCCURS 0 TO 10 133 800
DEPENDING ON --- NUMBER-OF-PROCEDURES-0405
03 TOOTH-NUMBER-0405 DISPLAY 9(2) 1 2
03 SERVICE-DATE-0405 DISPLAY 3 8
04 SERVICE-YEAR-0405 DISPLAY 9(4) 3 4
04 SERVICE-MONTH-0405 DISPLAY 9(2) 7 2
04 SERVICE-DAY-0405 DISPLAY 9(2) 9 2
03 PROCEDURE-CODE-0405 DISPLAY 9(4) 11 4
03 DESCRIPTION-OF-SERVICE-0405 DISPLAY X(60) 15 60
03 FEE-0405 COMP.3 S9(7)V99 75 5
03 FILLER DISPLAY X 80 1

RECORD NAME....... DEPARTMENT RLGTH= 72
RECORD VERSION..... 0100 DLGTH= 56
RECORD ID.......... 0410 KLGTH= 16
RECORD LENGTH...... FIXED DSTRT= 16
LOCATION MODE..... CALC USING DEPT-ID-0410 DUPLICATES NOT ALLOWED
WITHIN............. ORG-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............. TYPE........ NEXT PRIOR OWNER
CALC MEMBER 1 2
DEPT-EMPLOYEE INDEX OWNER 3 4

DATA ITEM........... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 DEPT-ID-0410 DISPLAY 9(4) 1 4
02 DEPT-NAME-0410 DISPLAY X(45) 5 45
02 DEPT-HEAD-ID-0410 DISPLAY 9(4) 50 4
02 FILLER DISPLAY XXX 54 3

RECORD NAME....... EMPLOYEE RLGTH= 192
RECORD VERSION..... 0100 DLGTH= 120
RECORD ID.......... 0415 KLGTH= 72
RECORD LENGTH...... FIXED DSTRT= 72
LOCATION MODE..... CALC USING EMP-ID-0415 DUPLICATES NOT ALLOWED
WITHIN............. EMP-DEMO-REGION OFFSET 5 PGS FOR 45 PGS
DBKEY POSITIONS.... SET............. TYPE........ NEXT PRIOR OWNER
CALC MEMBER 1 2
DEPT-EMPLOYEE INDEX MEMBER 3 4
EMP-NAME-NDX INDEX MEMBER 5
EMP-SSN-NDX INDEX MEMBER 6
OFFICE-EMPLOYEE INDEX MEMBER 7 8
EMP-COVERAGE OWNER 9 10
EMP-EMPOSITION OWNER 11 12
EMP-EXPERTISE OWNER 13 14
MANAGES OWNER 15 16
REPORTS-TO OWNER 17 18

DATA ITEM........... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 EMP-ID-0415 DISPLAY 9(4) 1 4
02 EMP-ADDRESS-0415 DISPLAY X(10) 5 10
03 EMP-LAST-NAME-0415 DISPLAY X(15) 15 15
02 EMP-ADDRESS-0415 DISPLAY 30 46
03 EMP-STREET-0415 DISPLAY X(20) 30 20
03 EMP-CITY-0415 DISPLAY X(15) 50 15
03 EMP-CITY-0415 DISPLAY X(15) 50 15
03 EMP-STATE-0415 DISPLAY X(2) 65 2
03 EMP-ZIP-0415 DISPLAY 67 9
04 EMP-ZIP-FIRST-FIVE-0415 DISPLAY X(5) 67 5
04 EMP-ZIP-LAST-FOUR-0415 DISPLAY X(4) 72 4
02 EMP-MOBILE-0415 DISPLAY 9(10) 76 10
02 STATUS-0415 DISPLAY X(2) 86 2
88 ACTIVE-0415 COND '01' 86
88 ST-DISABIL-0415 COND '02' 86
88 LT-DISABIL-0415 COND '03' 86
88 LEAVE-OF-ABSENCE-0415 COND '04' 86
88 TERMINATED-0415 COND '05' 86
02 SS-NUMBER-0415 DISPLAY 9(9) 88 9
02 START-DATE-0415 DISPLAY 97 8
03 START-YEAR-0415 DISPLAY 9(4) 97 4
03 START-MONTH-0415 DISPLAY 9(2) 101 2
03 START-DAY-0415 DISPLAY 9(2) 103 2
02 TERMINATION-DATE-0415 DISPLAY 105 8
03 TERMINATION-YEAR-0415 DISPLAY 9(4) 105 4
03 TERMINATION-MONTH-0415  DISPLAY  9(2)  109  2
03 TERMINATION-DAY-0415  DISPLAY  9(2)  111  2
02 BIRTH-DATE-0415  DISPLAY  113  8
03 BIRTH-YEAR-0415  DISPLAY  9(4)  113  4
03 BIRTH-MONTH-0415  DISPLAY  9(2)  117  2
03 BIRTH-DAY-0415  DISPLAY  9(2)  119  2

02 START-DATE-0420  DISPLAY  1  8
03 START-YEAR-0420  DISPLAY  9(4)  1  4
03 START-MONTH-0420  DISPLAY  9(2)  5  2
03 START-DAY-0420  DISPLAY  9(2)  7  2
02 FINISH-DATE-0420  DISPLAY  9(2)  15  2
03 FINISH-YEAR-0420  DISPLAY  9(4)  9  4
03 FINISH-MONTH-0420  DISPLAY  9(2)  13  2
03 FINISH-DAY-0420  DISPLAY  9(2)  17  2
02 SALARY-RATE-0420  DISPLAY  9(2)  19  5
02 SALARY-AMOUNT-0420  COMP-3  S9(7)V99  19  5
02 BONUS-PERCENT-0420  COMP-3  SV999  24  2
02 COMMISSION-PERCENT-0420  COMP-3  SV999  26  2
02 OVERTIME-RATE-0420  COMP-3  S9V99  28  2
02 FILLER  DISPLAY  XXX  30  3

02 START-DATE-0425  DISPLAY  1  8
03 START-YEAR-0425  DISPLAY  9(4)  1  4
03 START-MONTH-0425  DISPLAY  9(2)  5  2
03 START-DAY-0425  DISPLAY  9(2)  7  2
02 SKILL-LEVEL-0425  DISPLAY  XX  1  2
88 EXPERT-0425  COND  '04'  1
88 PROFICIENT-0425  COND  '03'  1
88 COMPETENT-0425  COND  '02'  1
88 ELEMENTARY-0425  COND  '01'  1
02 EXPENSE-RATE-0425  DISPLAY  9(4)  3  8
03 EXPENSE-YEAR-0425  DISPLAY  9(2)  7  2
03 EXPENSE-MONTH-0425  DISPLAY  9(2)  9  2
02 FILLER  DISPLAY  XX  11  2

02 CLAIM-DATE-0430  DISPLAY  1  8
03 CLAIM-YEAR-0430  DISPLAY  9(4)  1  4
03 CLAIM-MONTH-0430  DISPLAY  9(2)  5  2
03 CLAIM-DAY-0430  DISPLAY  9(2)  7  2
02 PATIENT-NAME-0430  DISPLAY  9  25
03 PATIENT-FIRST-NAME-0430  DISPLAY  X(10)  9  10
03 PATIENT-LAST-NAME-0430 DISPLAY X(15) 19 15
02 PATIENT-BIRTH-DATE-0430 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-YEAR-0430 DISPLAY 9(2) 38 2
03 PATIENT-BIRTH-MONTH-0430 DISPLAY 9(2) 40 2
02 PATIENT-SEX-0430 DISPLAY X 42 1
02 RELATION-TO-EMPLOYEE-0430 DISPLAY X(10) 43 10
02 HOSPITAL-NAME-0430 DISPLAY X(25) 53 25
03 HOSP-ADDRESS-0430 DISPLAY 78 46
03 HOSP-STREET-0430 DISPLAY X(20) 78 20
03 HOSP-CITY-0430 DISPLAY X(15) 98 15
03 HOSP-STATE-0430 DISPLAY X(2) 113 2
03 HOSP-ZIP-0430 DISPLAY 115 9
04 HOSP-ZIP-FIRST-FIVE-0430 DISPLAY X(5) 115 5
04 HOSP-ZIP-LAST-FOUR-0430 DISPLAY X(4) 120 4
02 ADMIT-DATE-0430 DISPLAY 124 8
03 ADMIT-YEAR-0430 DISPLAY 9(4) 124 4
03 ADMIT-MONTH-0430 DISPLAY 9(2) 128 2
03 ADMIT-DAY-0430 DISPLAY 9(2) 130 2
02 DISCHARGE-DATE-0430 DISPLAY 132 8
03 DISCHARGE-YEAR-0430 DISPLAY 9(4) 132 4
03 DISCHARGE-MONTH-0430 DISPLAY 9(2) 136 2
03 DISCHARGE-DAY-0430 DISPLAY 9(2) 138 2
02 DIAGNOSIS-0430 DISPLAY OCCURS 2 X(60) 140 120
02 HOSPITAL-CHARGES-0430 DISPLAY 260 41
03 ROOM-AND-BOARD-0430 DISPLAY 260 26
04 WARD-0430 DISPLAY 260 13
05 WARD-DAYS-0430 COMP-3 S9(5) 260 3
05 WARD-RATE-0430 COMP-3 S9(7)V99 263 5
05 WARD-TOTAL-0430 COMP-3 S9(7)V99 268 5
04 SEMI-PRIVATE-0430 DISPLAY 273 13
05 SEMI-DAYS-0430 COMP-3 S9(5) 273 3
05 SEMI-RATE-0430 COMP-3 S9(7)V99 276 5
05 SEMI-TOTAL-0430 COMP-3 S9(7)V99 281 5
03 OTHER-CHARGES-0430 DISPLAY 286 15
04 DELIVERY-COST-0430 COMP-3 S9(7)V99 286 5
04 ANESTHESIA-COST-0430 COMP-3 S9(7)V99 291 5
04 LAB-COST-0430 COMP-3 S9(7)V99 296 5

RECORD NAME........ INSURANCE-PLAN RLGTH= 140
RECORD VERSION..... 0100 DLGTH= 132
RECORD ID.......... 0435 KLGTH= 8
RECORD LENGTH...... FIXED DSTRT= 8
LOCATION MODE...... CALC USING INS-PLAN-CODE-0435 DUPLICATES NOT ALLOWED WITHIN............. INS-DEMO-REGION OFFSET 1 PGS FOR 4 PGS
DBKEY POSITIONS.... SET............. TYPE........ NEXT PRIOR OWNER CALC MEMBER 1 2
DATA ITEM.......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 INS-PLAN-CODE-0435 DISPLAY X(3) 1 3
88 GROUP-LIFE-0435 COND '001' 1
88 HMO-0435 COND '002' 1
88 GROUP-HEALTH-0435 COND '003' 1
88 GROUP-DENTAL-0435 COND '004' 1
02 INS-CO-NAME-0435 DISPLAY X(45) 4 45
02 INS-CO-ADDRESS-0435 DISPLAY 49 46
03 INS-CO-STREET-0435 DISPLAY X(28) 49 20
03 INS-CO-CITY-0435 DISPLAY X(15) 69 15
03 INS-CO-STATE-0435 DISPLAY X(2) 84 2
03 INS-CO-ZIP-0435 DISPLAY 86 9
04 INS-CO-ZIP-FIRST-FIVE-0435 DISPLAY X(5) 86 5
04 INS-CO-ZIP-LAST-FOUR-0435 DISPLAY X(4) 91 4
02 INS-CO-PHONE-0435 DISPLAY 9(10) 95 10
02 GROUP-NUMBER-0435 DISPLAY 9(6) 105 6
02 PLAN-DESCRIPTION-0435 DISPLAY 111 20
03 DEDUCT-0435 COMP-3 S9(7)V99 111 5
03 MAXIMUM-LIFE-COST-0435 COMP-3 S9(7)V99 116 5
03 FAMILY-COST-0435 COMP-3 S9(7)V99 121 5
03 DEP-COST-0435 COMP-3 S9(7)V99 126 5
02 FILLER DISPLAY XX 131 2
RECORD NAME........ JOB
RECORD VERSION..... 0100
RECORD ID.......... 0440
RECORD LENGTH...... FIXED (INTERNALLY VARIABLE) 24 CHARACTERS
MINIMUM ROOT...... 24 CHARACTERS
MINIMUM FRAGMENT.. 296 CHARACTERS
LOCATION MODE.... CALC USING JOB-ID-0440 DUPLICATES NOT ALLOWED
WITHIN.............. ORG-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
CALL PROCEDURES.... NAME.... WHEN.. FUNCTION
IDMSCOMP BEFORE STORE
IDMSCOMP BEFORE MODIFY
IDMSDCOM AFTER GET
DBKEY POSITIONS.... SET............. TYPE........ NEXT PRIOR OWNER
CALC MEMBER 1 2
JOB-TITLE-NDX INDEX MEMBER 3
JOB-EMPOSITION OWNER 4 5
(FRAGMENT CHAIN) INTRNL 6
DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 JOB-ID-0440 DISPLAY 9(4) 1 4
02 TITLE-0440 DISPLAY X(20) 5 20
03 DESCRIPTION-0440 DISPLAY X(60) 25 120
02 REQUIREMENTS-0440 DISPLAY X(60) 145 120
03 REQUIREMENT-LINE-0440 DISPLAY OCCURS 2 X(60) 145 120
02 MINIMUM-SALARY-0440 DISPLAY S9(6)V99 265 8
02 MAXIMUM-SALARY-0440 DISPLAY S9(6)V99 273 8
02 SALARY-GRADES-0440 DISPLAY OCCURS 4 9(2) 281 8
02 NUMBER-OF-POSITIONS-0440 DISPLAY 9(3) 289 3
02 NUMBER-OPEN-0440 DISPLAY 9(3) 292 3
02 FILLER DISPLAY XX 295 2

RECORD NAME........ NON-HOSP-CLAIM
RECORD VERSION..... 0100
RECORD ID.......... 0445
RECORD LENGTH...... VARIABLE 24 CHARACTERS
MINIMUM ROOT...... 248 CHARACTERS
MINIMUM FRAGMENT.. 1052 CHARACTERS
LOCATION MODE.... VIA SET COVERAGE-CLAIMS DISPLACEMENT 0000 PAGES
WITHIN.............. INS-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............. TYPE........ NEXT PRIOR OWNER
COVERAGE-CLAIMS MEMBER 1
(FRAGMENT CHAIN) INTRNL 2
DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 CLAIM-DATE-0445 DISPLAY 1 8
03 CLAIM-YEAR-0445 DISPLAY 9(4) 1 4
03 CLAIM-MONTH-0445 DISPLAY 9(2) 5 2
03 CLAIM-DAY-0445 DISPLAY 9(2) 7 2
02 PATIENT-NAME-0445 DISPLAY X(10) 9 25
03 PATIENT-FIRST-NAME-0445 DISPLAY X(10) 19 10
03 PATIENT-LAST-NAME-0445 DISPLAY X(15 19 15
02 PATIENT-BIRTH-DATE-0445 DISPLAY 9(4) 34 8
03 PATIENT-BIRTH-YEAR-0445 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-MONTH-0445 DISPLAY 9(2) 30 2
03 PATIENT-BIRTH-DAY-0445 DISPLAY 9(2) 42 2
02 PATIENT-SEX-0445 DISPLAY X 82 1
02 RELATION-TO-EMPLOYEE-0445 DISPLAY X(10) 43 10
02 PHYSICIAN-NAME-0445 DISPLAY 53 25
03 PHYSICIAN-FIRST-NAME-0445 DISPLAY X(10) 53 10
03 PHYSICIAN-LAST-NAME-0445 DISPLAY X(15) 63 15
02 PHYSICIAN-ADDRESS-0445 DISPLAY 70 46
03 PHYSICIAN-STREET-0445 DISPLAY X(20) 72 20
03 PHYSICIAN-CITY-0445 DISPLAY X(15) 98 15
03 PHYSICIAN-STATE-0445 DISPLAY X(2) 113 2
03 PHYSICIAN-ZIP-0445 DISPLAY 115 9
04 PHYSICIAN-ZIP-FIRST-FIVE-0445 DISPLAY X(5) 115 5
04 PHYSICIAN-ZIP-LAST-FOUR-0445 DISPLAY X(4) 120 4
02 PHYSICIAN-ID-0445 DISPLAY 9(6) 124 6
02 DIAGNOSIS-0445 DISPLAY OCCURS 2 X(60) 130 120
02 NUMBER-OF-PROCEDURES-0445 COMP 9(2) 250 2
02 FILLER         DISPLAY     X        252     1
02 PHYSICIAN-CHARGES-0445 DISPLAY OCCURS 0 TO 10     253     800

DEPENDING ON -- NUMBER-OF-PROCEDURES-0445

03 SERVICE-DATE-0445 DISPLAY 1     8
04 SERVICE-YEAR-0445 DISPLAY 9(4)  1     4
04 SERVICE-MONTH-0445 DISPLAY 9(2)  5     2
04 SERVICE-DAY-0445 DISPLAY 9(2)  7     2
03 PROCEDURE-CODE-0445 DISPLAY 9(4)  9     4
03 DESCRIPTION-OF-SERVICE-0445 DISPLAY X(60) 13     60
03 FEE-0445 COMP-3 S9(7)V99 73     5

RECORD NAME........ OFFICE                  RLGT= 92
RECORD VERSION..... 0100                   DLGT= 76
RECORD ID.......... 0450                   KLGT= 16
RECORD LENGTH...... FIXED                   DSTRT= 16
LOCATION MODE..... CALC USING OFFICE-CODE-0450 DUPLICATES NOT ALLOWED
WITHIN.......... ORG-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............. TYPE....... NEXT PRIOR OWNER
CALC MEMBER 1 2
OFFICE-EMPLOYEE INDEX OWNER 3 4

DATA ITEM......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 OFFICE-CODE-0450 DISPLAY X(3)  1     3
02 OFFICE-ADDRESS-0450 DISPLAY 4     46
03 OFFICE-STREET-0450 DISPLAY X(20) 24     15
03 OFFICE-CITY-0450 DISPLAY X(15)  39     2
03 OFFICE-STATE-0450 DISPLAY X(2)  39     2
03 OFFICE-ZIP-0450 DISPLAY X(9)  41     9
04 OFFICE-ZIP-FIRST-FIVE-0450 DISPLAY X(5)  41     5
04 OFFICE-ZIP-LAST-FOUR-0450 DISPLAY X(4)  46     4
02 OFFICE-PHONE-0450 DISPLAY OCCURS 3 9(7) 50     21
02 OFFICE-ZIP-FIRST-FIVE-0450 DISPLAY X(3)  71     3
02 OFFICE-PHONE-0450 DISPLAY X(3)  74     3

RECORD NAME........ SKILL                  RLGT= 96
RECORD VERSION..... 0100                   DLGT= 76
RECORD ID.......... 0455                   KLGT= 20
RECORD LENGTH...... FIXED                   DSTRT= 20
LOCATION MODE..... CALC USING SKILL-ID-0455 DUPLICATES NOT ALLOWED
WITHIN.......... ORG-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............. TYPE....... NEXT PRIOR OWNER
CALC MEMBER 1 2 3
SKILL-NAME-NDX INDEX MEMBER 3
SKILL-EXPERTISE INDEX OWNER 4 5

DATA ITEM......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 SKILL-ID-0455 DISPLAY 9(4)  1     4
02 SKILL-NAME-0455 DISPLAY X(12)  5     12
02 SKILL-DESCRIPTION-0455 DISPLAY X(60) 17     60

RECORD NAME........ STRUCTURE             RLGT=  36
RECORD VERSION..... 0100                   DLGT= 12
RECORD ID.......... 0460                   KLGT=  24
RECORD LENGTH...... FIXED                   DSTRT= 24
LOCATION MODE..... VIA SET MANAGES DISPLACEMENT 0000 PAGES
WITHIN.......... EMP-DEMO-REGION OFFSET 5 PGS FOR 45 PGS
DBKEY POSITIONS.... SET............. TYPE....... NEXT PRIOR OWNER
REPORTS-TO MEMBER 1 2 3
MANAGES MEMBER 4 5 6

DATA ITEM......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 STRUCTURE-CODE-0460 DISPLAY X(2)  1     2
88 ADMIN-0460 COND  'A'  1
88 PROJECT-0460 COND 'P1' THRU 'P9' 1
02 STRUCTURE-DATE-0460 DISPLAY 3     8
03 STRUCTURE-YEAR-0460 DISPLAY 9(4)  3     4
03 STRUCTURE-MONTH-0460 DISPLAY 9(2)  7     2
03 STRUCTURE-DAY-0460 DISPLAY 9(2)  9     2
02 FILLER         DISPLAY XX  11     2
Data Structure Diagram

The figure below shows the structure of the Employee database.

Introduction

Summary of CA Culprit Parameters

This section presents tables that summarize the CA Culprit parameters:

- Processing Operations (see page 218)
- Summary of Advanced Capabilities for Standard Files (see page 219)
- Summary of Parameters for Using Database Data (see page 221)
- Summary of Advanced Capabilities for Accessing the Database (see page 222)
- Minimum Coding Requirements for Using Data Tables (see page 222)
- Minimum Coding Requirements for Using SQL Tables (see page 224)

The Basic CA Culprit Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>What CA Culprit needs</th>
<th>What to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the basic report.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>What CA Culprit needs</td>
<td>What to use</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Record size and type</td>
<td>IN</td>
</tr>
<tr>
<td></td>
<td>Fields being used</td>
<td>REC</td>
</tr>
<tr>
<td></td>
<td>Size of the report</td>
<td>OUT</td>
</tr>
<tr>
<td></td>
<td>Output organization</td>
<td>Type 5</td>
</tr>
<tr>
<td>Add a title.</td>
<td>Title details</td>
<td>Type 3</td>
</tr>
<tr>
<td>Generate subtitles.</td>
<td>Subtitle details</td>
<td>Type 4</td>
</tr>
<tr>
<td>Generate column headings.</td>
<td>Where to find the header information</td>
<td>HH, HR, HF</td>
</tr>
<tr>
<td>Adjust column widths.</td>
<td>Column size</td>
<td>SZ=</td>
</tr>
<tr>
<td>Format numeric output.</td>
<td>A template</td>
<td>FS, FS, FD, Edit masks</td>
</tr>
<tr>
<td>Create multiple reports.</td>
<td>Report identifier</td>
<td>Report numbers</td>
</tr>
<tr>
<td>Sequence data.</td>
<td>The field(s) and their order</td>
<td>SORT</td>
</tr>
<tr>
<td>Process selected data.</td>
<td>Selection criteria</td>
<td>SELECT/BYPASS</td>
</tr>
<tr>
<td>Reference multiple detail lines.</td>
<td>Line identifier</td>
<td>Type 5 line numbers</td>
</tr>
<tr>
<td>Generate subtotals and page breaks.</td>
<td>Break codes</td>
<td>1, 0, -, +</td>
</tr>
<tr>
<td>Control the output of totals.</td>
<td>Identification of the field to be totaled</td>
<td>Type 6</td>
</tr>
</tbody>
</table>

### Processing Operations

**Processing Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acts upon input data and writes it to the extract file</td>
</tr>
</tbody>
</table>
### Summary of Advanced Capabilities for Standard Files

<table>
<thead>
<tr>
<th>Task</th>
<th>CA Culprit instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print totals only.</td>
<td>OUT T</td>
</tr>
<tr>
<td></td>
<td>Type 8 logic</td>
</tr>
<tr>
<td></td>
<td>Type 6 edit line</td>
</tr>
<tr>
<td>Manipulate system-maintained totals.</td>
<td>Type 8 parameter</td>
</tr>
<tr>
<td>Define items to be totaled.</td>
<td>Type 6 parameter</td>
</tr>
<tr>
<td>Format total lines.</td>
<td></td>
</tr>
<tr>
<td>Obtain current work field value.</td>
<td>Type 6 parameter</td>
</tr>
<tr>
<td></td>
<td>Type 8 parameter</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input processing</td>
<td>Type 7</td>
<td>parameter</td>
</tr>
<tr>
<td>Output processing</td>
<td>Type 8</td>
<td>parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acts upon data stored in the extract file for report output</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>ADD (+)</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td>MINUS (-)</td>
<td>Subtract</td>
</tr>
<tr>
<td></td>
<td>TIMES (X)</td>
<td>Multiply</td>
</tr>
<tr>
<td></td>
<td>DIVIDE (/)</td>
<td>Divide</td>
</tr>
<tr>
<td></td>
<td>COMPUTE</td>
<td>Performs compound arithmetic operations</td>
</tr>
<tr>
<td>Conditional</td>
<td>IF</td>
<td>Performs a comparison or a test</td>
</tr>
<tr>
<td>Assignment</td>
<td>MOVE</td>
<td>Transfers a value to a field</td>
</tr>
<tr>
<td>Processing control</td>
<td>TAKE</td>
<td>Prints selected edit lines and stops processing</td>
</tr>
<tr>
<td></td>
<td>DROP</td>
<td>Stops processing current record</td>
</tr>
<tr>
<td></td>
<td>RELS</td>
<td>Prints selected edit lines without interrupting processing</td>
</tr>
<tr>
<td></td>
<td>PERFORM</td>
<td>Sends the processing to a specified line or section of code</td>
</tr>
<tr>
<td></td>
<td>RETURN</td>
<td>Returns processing to code immediately following the related PERFORM statement</td>
</tr>
<tr>
<td>Task</td>
<td>CA Culprit instruction</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>Subscripted work fields on type 6 or type 8 parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain a sort key value.</td>
<td>Control break</td>
<td></td>
</tr>
<tr>
<td>Subscripted work fields on type 6 or type 8 parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference repeating groups.</td>
<td>Explicit, implied, or zero subscripts</td>
<td></td>
</tr>
<tr>
<td>Reference multiply-occurring input fields.</td>
<td>Subscripts using GROUP REC and ELMNT REC</td>
<td></td>
</tr>
<tr>
<td>Access file definitions stored in the data dictionary.</td>
<td>DATABASE DICTNAME=</td>
<td></td>
</tr>
<tr>
<td>Override parameters generated by the data dictionary.</td>
<td>IN REC GROUP REC/ELMNT REC</td>
<td></td>
</tr>
<tr>
<td>Match files.</td>
<td>IN parameter with MK=option</td>
<td></td>
</tr>
<tr>
<td>Insert the current value of a field into a heading.</td>
<td>Type 4 parameter Control breaks SORT/NOSORT B HEAD</td>
<td></td>
</tr>
<tr>
<td>Create nonprinted output.</td>
<td>OUT parameter with optional descriptors Format codes: FB, FU, FW, FP, or FZ DD=SYS020</td>
<td></td>
</tr>
<tr>
<td>Create standard files from the database.</td>
<td>DATABASE DICTNAME= IN DB SS= PATH DD=SYS020 (delete SYS010)</td>
<td></td>
</tr>
<tr>
<td>Copy the contents of a data file.</td>
<td>=MACRO AMLIST</td>
<td></td>
</tr>
<tr>
<td>Copy stored code.</td>
<td>USE =MACRO/=MEND =COPY</td>
<td></td>
</tr>
<tr>
<td>Change inline or copied code.</td>
<td>USE</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>CA Culprit instruction</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>Assign default values to symbolic parameters.</td>
<td>USE</td>
<td></td>
</tr>
<tr>
<td>Nest copied code.</td>
<td>USE */ END</td>
<td></td>
</tr>
</tbody>
</table>

### Summary of Parameters for Using Database Data

#### Summary of Parameters for Using Database Data

<table>
<thead>
<tr>
<th>Function</th>
<th>What CA Culprit needs</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Access the database.</strong></td>
<td>Data dictionary name (if not primary dictionary)</td>
<td>DATABASE DICTNAME=</td>
</tr>
<tr>
<td></td>
<td>Data source</td>
<td>IN DB</td>
</tr>
<tr>
<td></td>
<td>Subschema used</td>
<td>SS=</td>
</tr>
<tr>
<td><strong>Navigate the database.</strong></td>
<td>Route through the database</td>
<td>PATH</td>
</tr>
<tr>
<td><strong>Determine the database route used.</strong></td>
<td>Keyword</td>
<td>PATH-ID</td>
</tr>
<tr>
<td><strong>Use database information as a standard file.</strong></td>
<td>Predefined logical records</td>
<td>PATH logical-record-name</td>
</tr>
<tr>
<td><strong>Select logical records.</strong></td>
<td>Conditional statements</td>
<td>WHERE</td>
</tr>
<tr>
<td><strong>Search for defined values.</strong></td>
<td>Comparison statements</td>
<td>CONTAINS MATCHES</td>
</tr>
</tbody>
</table>

15-Jan-2018
Summary of Advanced Capabilities for Accessing the Database

<table>
<thead>
<tr>
<th>TASK</th>
<th>CA Culprit instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve partial information.</td>
<td>Alternate path</td>
</tr>
<tr>
<td>Select record types by record name.</td>
<td>SELECT/BYPASS</td>
</tr>
<tr>
<td>Direct record retrieval by key value.</td>
<td>KEY</td>
</tr>
<tr>
<td>Use a key file to retrieve records.</td>
<td>KEYFILE</td>
</tr>
<tr>
<td>Test for record occurrences.</td>
<td>PATH--</td>
</tr>
<tr>
<td></td>
<td>DB-EXIT facility</td>
</tr>
<tr>
<td>Access more than one relationship between records.</td>
<td>PATH--</td>
</tr>
<tr>
<td></td>
<td>DB-EXIT facility</td>
</tr>
<tr>
<td>Retrieve all occurrences of a set.</td>
<td>PATH--</td>
</tr>
<tr>
<td></td>
<td>DB-EXIT facility</td>
</tr>
<tr>
<td>Access multiple-member sets.</td>
<td>PATH parameter ALL-MEMBERS option</td>
</tr>
</tbody>
</table>

Minimum Coding Requirements for Using Data Tables

Minimum Input and Output Coding Requirements for Using Data Tables

<table>
<thead>
<tr>
<th>Task</th>
<th>CA Culprit parameter /keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify the source of conventional file input to create a table.</td>
<td>INPUT</td>
</tr>
<tr>
<td>Define fields from non-IDMS files to CA Culprit.</td>
<td>REC</td>
</tr>
<tr>
<td>Define printed output.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>Define an output data table.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>TABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=CREATE</td>
</tr>
<tr>
<td></td>
<td>USER=</td>
</tr>
<tr>
<td>Task</td>
<td>CA Culprit parameter /keywords</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>PW=</td>
<td>CATALOG=</td>
</tr>
</tbody>
</table>

Name the columns in a details-only table. Type 5 (at least one)

Name the columns in a totals-only table. Type 6

Retrieve a data table (copy a view). INPUT

 TABLE=
 TYPE=COPY
 USER=
 PW=
 CATALOG=

Store additional rows in a previously defined data table. OUTPUT

 TABLE=
 TYPE=ADD
 USER=
 PW=
 CATALOG=

Remove all rows and place new rows in a previously stored data table. OUTPUT

 TABLE=
 TYPE=REPLACE
 USER=
 PW=
 CATALOG=

Delete a previously stored data table and all associated data. OUTPUT

 TABLE=
 TYPE=DELETE
 USER=
 PW=


<table>
<thead>
<tr>
<th>Task</th>
<th>CA Culprit parameter /keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify the table definition of an existing table.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>TABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=GENERATE</td>
</tr>
<tr>
<td></td>
<td>USER=</td>
</tr>
<tr>
<td></td>
<td>PW=</td>
</tr>
<tr>
<td></td>
<td>CATALOG=</td>
</tr>
</tbody>
</table>

### Minimum Coding Requirements for Using SQL Tables

#### Minimum Input and Output Coding Requirements for Using SQL Tables

<table>
<thead>
<tr>
<th>Task</th>
<th>CA Culprit parameter /keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify the source of conventional file input to create an SQL table.</td>
<td>INPUT</td>
</tr>
<tr>
<td>Define fields from non-IDMS files to CA Culprit.</td>
<td>REC</td>
</tr>
<tr>
<td>Define printed output.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>Identify the input source as an SQL table; specify table location.</td>
<td>IN DB(Q);</td>
</tr>
<tr>
<td></td>
<td>DICTIONARY= SCHEMA=</td>
</tr>
<tr>
<td>Define the SQL query to CA Culprit.</td>
<td>SQL select-statement</td>
</tr>
<tr>
<td>Define an alias for an SQL column within the SQL query.</td>
<td>AS</td>
</tr>
<tr>
<td>Define an output data table.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>SQLTABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=CREATE</td>
</tr>
<tr>
<td></td>
<td>DICTIONARY=</td>
</tr>
<tr>
<td></td>
<td>SCHEMA=</td>
</tr>
<tr>
<td>Name the columns in a details-only table.</td>
<td>Type 5 (at least one)</td>
</tr>
<tr>
<td>Name the columns in a totals-only table.</td>
<td>Type 6</td>
</tr>
<tr>
<td>Indicate an SQL column cannot contain null values</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>Define a NULL indicator for SQL columns that are allowed to be NULL.</td>
<td>FB SZ=4</td>
</tr>
<tr>
<td>Task</td>
<td>CA Culprit parameter /keywords</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>Store additional rows in a previously defined SQL table.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>SQLTABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=ADD</td>
</tr>
<tr>
<td></td>
<td>DICTIONARY=</td>
</tr>
<tr>
<td></td>
<td>SCHEMA=</td>
</tr>
<tr>
<td>Remove all rows and place new rows in a previously stored SQL table.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>SQLTABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=REPLACE</td>
</tr>
<tr>
<td></td>
<td>DICTIONARY=</td>
</tr>
<tr>
<td></td>
<td>SCHEMA=</td>
</tr>
<tr>
<td>Delete a previously stored SQL table and all associated data.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>SQLTABLE=</td>
</tr>
<tr>
<td></td>
<td>TYPE=DROP</td>
</tr>
<tr>
<td></td>
<td>DICTIONARY=</td>
</tr>
<tr>
<td></td>
<td>SCHEMA=</td>
</tr>
</tbody>
</table>