## Contents

### Index

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Introduction and Tutorial</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Welcome to COBOL Report Writer</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Gentle Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.3 More about COBOL Report Writer</td>
<td>18</td>
</tr>
<tr>
<td>1.4 Some Shorter Forms</td>
<td>30</td>
</tr>
<tr>
<td>1.5 Other Features</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Report Files and RD Entries</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Report Files and RD: Keyword Table</td>
<td>41</td>
</tr>
<tr>
<td>2.2 Report Files</td>
<td>43</td>
</tr>
<tr>
<td>2.3 REPORT SECTION and RD</td>
<td>50</td>
</tr>
<tr>
<td>2.4 ALLOW clause</td>
<td>53</td>
</tr>
<tr>
<td>2.5 CODE clause</td>
<td>54</td>
</tr>
<tr>
<td>2.6 CONTROL clause</td>
<td>56</td>
</tr>
<tr>
<td>2.7 LINE LIMIT clause</td>
<td>63</td>
</tr>
<tr>
<td>2.8 OVERFLOW clauses</td>
<td>65</td>
</tr>
<tr>
<td>2.9 PAGE LIMIT clause</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>Report Group Descriptions</strong></td>
<td></td>
</tr>
<tr>
<td>3.1 Introducing Report Groups</td>
<td>77</td>
</tr>
<tr>
<td>3.2 Coding Report Group Descriptions</td>
<td>81</td>
</tr>
<tr>
<td>3.3 BLANK WHEN ZERO clause</td>
<td>86</td>
</tr>
<tr>
<td>3.4 COLUMN clause</td>
<td>87</td>
</tr>
<tr>
<td>3.5 COLUMN-COUNTER</td>
<td>93</td>
</tr>
<tr>
<td>3.6 COUNT clause</td>
<td>94</td>
</tr>
<tr>
<td>3.7 FUNCTION clause</td>
<td>96</td>
</tr>
<tr>
<td>3.8 GROUP LIMIT clause</td>
<td>103</td>
</tr>
<tr>
<td>3.9 JUSTIFIED clause</td>
<td>104</td>
</tr>
<tr>
<td>3.10 LINE clause</td>
<td>105</td>
</tr>
<tr>
<td>3.11 LINE-COUNTER</td>
<td>112</td>
</tr>
<tr>
<td>3.12 MULTIPLE PAGE clause</td>
<td>114</td>
</tr>
<tr>
<td>3.13 NEXT GROUP clause</td>
<td>116</td>
</tr>
<tr>
<td>3.14 OCCURS clause</td>
<td>120</td>
</tr>
<tr>
<td>3.15 PAGE-COUNTER</td>
<td>127</td>
</tr>
<tr>
<td>3.16 PICTURE clause</td>
<td>129</td>
</tr>
<tr>
<td>3.17 PRESENT AFTER clause</td>
<td>133</td>
</tr>
<tr>
<td>3.18 PRESENT WHEN clause</td>
<td>139</td>
</tr>
<tr>
<td>3.19 REPEATED clause</td>
<td>152</td>
</tr>
</tbody>
</table>
3.20 SIGN clause 156
3.21 SOURCE clause 158
3.22 STYLE clause 163
3.23 SUM clause 167
3.24 TYPE clause 188
3.25 USAGE clause 195
3.26 VALUE clause 196
3.27 VARYING clause 199
3.28 WRAP clause 203

4 Procedural Statements 209
4.1 Report Writer Verbs: Overview 211
4.2 GENERATE statement 213
4.3 INITIATE statement 219
4.4 Report Writer SET statements 221
4.5 SUPPRESS PRINTING statement 227
4.6 TERMINATE statement 228
4.7 USE BEFORE REPORTING directive 230

5 Special Topics 235
5.1 Multiple Reports 237
5.2 Developing User-Written Functions 244
5.3 Independent Report File Handlers 248

6 Migration from OS/VS or DOS/VS COBOL Report Writer 261
6.1 Re-compiling OS/VS and DOS/VS COBOL Sources 263
6.2 Other Considerations 274
6.3 Physical Comparison of Report Writer Output 274
6.4 Unreachable Code 275

Appendices 277
Appendix A - List of Post-1968 Extensions 279
Appendix B - List of New Reserved Words 287
Appendix C - Summary of Formats 289
Appendix D - Glossary 297
Appendix E - Precompiler Messages 303
Preface

Note for the browsable (PDF) edition. This version is a true rendering of the original printed document, but with color replacing lines and arrows, and many cross-references (or “hot spots”) inserted. When reading text that refers elsewhere, please probe for these hot spots to get there automatically. Also, these Preface sheets were simplified for convenience.

This publication is intended for programmers engaged in the writing of new COBOL programs using Report Writer, or the maintenance of old ones. Most of the text is intended for the general application programmer, but there is also information in part 5 for the systems programmer engaged in writing user extensions, such as for special output devices.

The language described in this eighth edition includes all the extensions to the Report Writer feature described in the ANS-85 standard and enhancements to the language made up to May 1995.

This publication also describes the basic (ANS-68) features used in IBM* OS/VS and DOS/VS COBOL and the many extensions introduced by IBM, Codasyl, and SPC Systems. For this reason, the product is referred to here as new Report Writer to distinguish it from the built-in Report Writer of OS/VS and DOS/VS COBOL that they contain as a subset.

This publication is a combination of all the following elements:

- **tutorial** (Introduction and Tutorial),
- **detailed language description** (Report Files and RD Entries, Report Group Descriptions, Procedural Statements and Special Topics),
- **migration guide** (Migration from OS/VS or DOS/VS COBOL Report Writer),
- **quick reference** (Appendices)

The tutorial is a step-by-step introduction, containing sufficient detail to enable programmers to write or maintain simple Report Writer code, while giving them an appreciation of what is possible using the more advanced features. Readers with a knowledge of OS/VS or DOS/VS COBOL's built-in Report Writer should also read this part.

The language description contains a formal explanation of the syntax and illustrated explanation of the usage of each clause and statement.

The migration guide is for use in the migration of programs from OS/VS or DOS/VS COBOL's built-in Report Writer to the new Report Writer described here. You can obtain notification of your use of any extensions by means of a precompiler option (see Installation and Operation).

The Appendices list and categorize the extensions, with an explanation of the error messages and a summary of syntax and reserved words.

* IBM is a trademark of International Business Machines Corporation.
This first part is a short introduction to the principles of COBOL Report Writer. After reading it, you will be able to write or maintain simple report writer code and you will have enough appreciation of the more advanced concepts to be able to locate the information quickly in the main parts.

All the information given here can also be found in the more formal context of Parts 2 to 5.
1.1 Welcome to COBOL Report Writer

1.1.1 Introduction to this Product

This product has two separate purposes:

- To improve programmer productivity in all aspects of printed output in COBOL by encouraging both experienced users and newcomers to make more use of COBOL's report writer feature.

- To help users who have had experience with a version of COBOL report writer that was an integral part of the compiler and want to continue to use the same facilities.

The Report Writer features are implemented in this product by means of a precompiler, rather than within the compiler itself. The compiler processes the intermediate source which the precompiler automatically passes to it. The precompiler phase is made as far as possible transparent to programmers, so that their attention is not distracted from the original report writer source. The precompiler and the compiler cooperate closely in a single-step operation and a final listing phase combines the output from both to produce a single source listing, enabling you to disregard the fact that two separate processes are involved. A description of this process will be found in Installation and Operation.

1.1.2 What is Report Writer?

Report Writer is COBOL's own built-in non-procedural facility for the production of printed output. It enables you to define and produce all the listings, reports, and displayed summaries that would normally be required from a COBOL application, but in far less time. It allows many more printed outputs, which might have been produced previously using stand-alone non-COBOL report generators, to be done in COBOL, because it reduces greatly the time and effort needed to code and test a COBOL program with printed output.

Report writer appeared in its original form in 1961 and later entered the 1968 ANS Standard. This version provided certain basic features that users of accounting machines were accustomed to, such as simple accumulation, cross-footing, and counter-rolling, as well as automatic page numbering. The implementation of report writer described in this volume contains all the facilities of the standard ANS-68, ANS-74, and ANS-85 report writer, plus IBM extensions, and includes many additional features which were added in various stages since 1974, many of which appear in the proposed ANS-9x standard. It is more suitable for the more varied and complex outputs needed by modern applications. There is no special term for this generalized, extended version of the language, so it is referred to in this manual simply as new Report Writer.
1.1.3 Compatibility With Built-In COBOL Report Writer

Apart from a few insubstantial differences, listed and explained in Part 6, COBOL Report Writer includes the whole of the ANS-68 Report Writer of IBM's OS/VS COBOL and DOS/VS COBOL so if you will be using sources migrated from either of these, they should work just as they did before. Customizing with the (default) option OSVS set on ensures the highest degree of compatibility with OS/VS and DOS/VS COBOL (see Installation and Operation.)

COBOL Report Writer also has many completely new features that are not a part of these standards, as well as enhancements to the original features. Several of them look forward to the next ANS standard. This volume points out which features are unique to new Report Writer in a Compatibility paragraph at the end of each section. A summary list of all the enhancements will be found at the start of parts 2, 3, and 4, and in Appendix A. Those ANS-68 features that were deleted or changed in the ANS-74 and ANS-85 Standards are nevertheless retained in this product; these cases are also listed in Appendix A.

1.2 Gentle Introduction

1.2.1 What is a Report?

Wherever you see the term report in this publication, it means any human readable output that may be produced by a program. Nowadays, the term report is normally used to mean a special printout or screen produced by a report generator. We use the term in a more general sense. Any readable output, whether long or short, "one-shot" or routine, printed or not, is a report. For instance, any of the following is a report and could be produced by COBOL Report Writer:

- Pay slips and paychecks printed on a mainframe printer;
- Invoices printed by a small remote printer;
- A small summary print produced at the end of a large update program;
- Sales of golf shoes, summarized by region and area, during the years 1985 to 1992 (a one-time, ad hoc report);
- An extremely complex print of personnel records with many variable-length lines and fields, "printed" on microfiche.

The only requirements for a report are that it should be readable (all fields USAGE DISPLAY only) and should consist of output only.

1.2.2 What Does Report Writer Do?

When you write Report Writer code, you do not write a sequence of procedural statements as you would in elementary COBOL. Instead, most of your effort is spent in specifying the appearance of the report. The DATA DIVISION syntax enables you to code the layout of your printout entirely in descriptive data-oriented terms. The only
"verbs" used are those that begin (INITIATE) and end (TERMINATE) the report and that GENERATE whole blocks of lines, known as report groups.

Report writer automatically generates your print record descriptions, your intermediate data areas, and all the procedural code needed to produce your outputs, saving you the effort that elementary COBOL would have required. For particularly difficult or challenging layouts, there are more advanced data clauses. By studying these in the later parts of this publication, you will learn to produce all your outputs with COBOL Report Writer.

Although so much is performed automatically, you still retain control at the highest level over all operations, because no report writer action takes place until one of the statements INITIATE, GENERATE, or TERMINATE is executed. However, these statements are sufficiently high-level to require only the simplest logic in your PROCEDURE DIVISION.

Since you may use COBOL Report Writer in any COBOL program, you may use it in any program that has to produce readable output - even if the program performs many other tasks. COBOL Report Writer does not extract the input data itself, unlike a report generator, which means that it may be used in partnership with all types of COBOL input: standard files, databases, and subroutine or module linkage.

1.2.3 Report Writer in Easy Steps

Step 1: Find the Report Groups

Your program may have one or several report layouts. Here is an example of one hypothetical report layout:

<table>
<thead>
<tr>
<th>CRUMBY COOKIE COMPANY</th>
<th>ORDERS</th>
<th>PAGE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DATE</strong></td>
<td><strong>TYPE</strong></td>
<td><strong>QUANTITY</strong></td>
</tr>
<tr>
<td>10/04/84</td>
<td>GINGERBREAD</td>
<td>100</td>
</tr>
<tr>
<td>06/05/84</td>
<td>CHOC. CHIPS</td>
<td>50</td>
</tr>
<tr>
<td>11/06/84</td>
<td>LEMON CREAM</td>
<td>150</td>
</tr>
</tbody>
</table>

**OUT OF STOCK**

**TOTALS: DEPOT NORTH-WEST** $148.70

Your first task is to divide up the layout into report groups. A report group is a "block" of lines, produced in one operation. Your layout may be built up from any number of different report groups. You can allow the shape and contents of each report group to vary as much as you like but, if the variations become very complex, it will be easier to define two different report groups. The following guidelines should be used to define a report group:

- It may consist of from one up to any number of lines, and may have any number of fields.
• It normally fits on one page, rather than being split by a page boundary. There are exceptions to this rule in MULTIPLE PAGE groups and REPORT HEADING and FOOTING groups, described later.

• It may contain fields whose contents come from anywhere in the DATA DIVISION, provided that all the fields are present in memory at the moment your program generates the report group.

If your report structure corresponds to records in a main file or database, remember that, unless you have a special reason for reading ahead and buffering several records, a report group should correspond to one record from your main file or database. (However, there is also a summary reporting feature that enables your program to output one report group that summarizes a whole set of records.)

Mark each report group clearly. Only one instance of each group needs to be marked, because only one description of each group is needed. You might use square brackets in the margin of your layout. In this example, let’s draw a rectangle round each report group.

Here is the result:

```
CRUMBLY COOKIE COMPANY  ORDERS  PAGE 1

<table>
<thead>
<tr>
<th>DATE</th>
<th>TYPE</th>
<th>QUANTITY</th>
<th>VALUE OF ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/04/84</td>
<td>GINGERBREAD</td>
<td>100</td>
<td>$20.50</td>
</tr>
<tr>
<td>06/05/84</td>
<td>CHOC. CHIPS</td>
<td>50</td>
<td>$18.20</td>
</tr>
<tr>
<td>11/06/84</td>
<td>LEMON CREAM</td>
<td>150</td>
<td>$110.00</td>
</tr>
<tr>
<td></td>
<td>**OUT OF STOCK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TOTALS: DEPOT NORTH-WEST $148.70
```

There are three instances of report group (B) in the picture. Only one instance needs a "box", and it is best to draw it around the most complex case, that is, the instance with the extra line "**OUT OF STOCK". (We want this line to be part of the same report group, rather than a report group in its own right, because we want to ensure that it will never be separated from the preceding line by a page advance.)

**Step 2: Decide on the TYPE of Each Report Group**

Each report group can appear in one of seven basic positions in your report, indicated by the TYPE clause. Here are their names and positions:
DETAIL or DE

This is the TYPE assumed by any group that is not of one of the special six described below. DETAIL groups usually contain the most basic data in the report. They are the only report groups that you GENERATE. TYPE DETAIL and the next two are known as body groups. (They fall between the PAGE HEADING and PAGE FOOTING, if any, on each page.)

CONTROL HEADING or CH

This group is generated automatically at the start of each different value of the corresponding control field (as explained in Step 3 below).

CONTROL FOOTING or CF

This group is generated automatically at the end of each different value of the corresponding control field.

PAGE HEADING or PH

This group will appear at the start of each page.

PAGE FOOTING or PF

This group will appear at the end of each page.

REPORT HEADING or RH

This group will appear once, on a page by itself or before the first PAGE HEADING (if any), at the very start of the printout.

REPORT FOOTING or RF

This group will appear once, on a page by itself or after the last PAGE FOOTING (if any), at the very end of the printout.

Each TYPE is optional. Your report may contain any number of different DETAIL groups, any number of different CONTROL HEADING and CONTROL FOOTING groups (up to one of each for each control level), but only one of each of the other four.
We can now assign the correct TYPEs to each group in our layout:

<table>
<thead>
<tr>
<th>DATE</th>
<th>TYPE</th>
<th>QUANTITY</th>
<th>VALUE OF ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/04/84</td>
<td>GINGERBREAD</td>
<td>100</td>
<td>$20.50</td>
</tr>
<tr>
<td>06/05/84</td>
<td>CHOC. CHIPS</td>
<td>50</td>
<td>$18.20</td>
</tr>
<tr>
<td>11/06/84</td>
<td>LEMON CREAM</td>
<td>150</td>
<td>$110.00</td>
</tr>
<tr>
<td><strong>OUT OF STOCK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TOTALS: DEPOT NORTH-WEST** $148.70

---

**Step 3: Code the RD Entry**

Your report groups are described in the **REPORT SECTION**, which is the **last** section in your program’s DATA DIVISION. The REPORT SECTION may contain any number of **Report Descriptions**. Each of these begins with an **RD** entry that starts in the A-margin:

```
REPORT SECTION.
RD
```

Follow this with a **report-name** of your choice. This name will be used to stand for the report as a whole, so choose a name that is appropriate:

```
REPORT SECTION.
RD  STOCK-SUMMARY
```

Several clauses may follow your report-name. The optional **LINE LIMIT** clause gives the maximum number of columns you expect per line and is used as a safety measure against losing data due to line overflow. The **FIRST DETAIL** clause (or its alternative spellings **FIRST DE** or **FIRST BODY GROUP**) indicates on which line the main information of each page should start. The **PAGE LIMIT** clause is required if your report is divided into pages. It gives the maximum number of lines to be written to each page. The order in which you code these clauses and phrases does not matter.
There are other clauses available to mark out different regions of the page. (See 2.9 PAGE LIMIT clause.)

Our report has control totals. That is, after the change in value of a certain control field (DEPOT), we want COBOL Report Writer to produce an extra report group (the CONTROL FOOTING). The data must arrive in the correct sequence as COBOL Report Writer does not SORT your data itself. (You might use COBOL SORT for that.) The field is called a control and a change in its value is called a control break. You may nest as many different levels of control as you need. You may also have corresponding CONTROL HEADING groups to appear before the start of the detail lines for the new control value. (In our example, there is just one level of control and no CONTROL HEADING group.) You indicate which fields are to be used to test for control breaks by means of the CONTROLS clause. Each control represents a different level. Your controls must be listed in hierarchical order from highest down to lowest. In our example it is simple because there is only one level:

```
REPORT SECTION.
RD STOCK-SUMMARY
   LINE LIMIT 132
   FIRST DETAIL 5
   PAGE LIMIT 64
   CONTROL IS DEPOT.
```

Because the CONTROL clause is the last clause of the RD entry, you write a period (".") after it. Here is another example of a CONTROL clause. This time, we have two control fields and also a special all-encompassing level, known as REPORT or FINAL, that may be used for producing grand totals for the whole report.

```
REPORT SECTION.
RD STOCK-SUMMARY
   LINE LIMIT 132
   FIRST DETAIL 5
   PAGE LIMIT 64
   CONTROLS ARE REPORT, YEAR, MONTH
```

LINE LIMIT, FIRST DETAIL, PAGE LIMIT, and CONTROL are not the only clauses you can write in the RD entry. The others are described in the chapter Report Files and RD Entries. The order in which you code the RD clauses is irrelevant.

**Step 4:** Code the Report Group Descriptions

**Step 4A:** 01-Level Entries
Each report group is coded as a series of COBOL entries. Each entry consists of a level-number, an optional data-name, any number of optional clauses, and a period. Each report group must start with a 01 level-number in the A-margin:

01

If the group is a DETAIL, follow this with a report-group data-name of your choice, followed by the optional word TYPE and the type of the group:

01  TYPE PH.
  ... etc ...
  01  COOKIE-LINE  TYPE DE.
  ... etc ...
  01  TYPE CF.
  ... etc ...

To make your program even clearer, you may spell out the TYPE clause in full; for instance: TYPE IS PAGE HEADING. Any number of entries, indicated by our "...", may follow the 01-level entry, as you will shortly see.

You indicate which level of CONTROL HEADING and CONTROL FOOTING you are describing by writing FOR name-of-control after CH and CF. (This is optional if there is only one level, or you want ALL levels in a CONTROL FOOTING). Taking our example with three levels above, you might code:

01  TYPE CH FOR YEAR.
  ... 
  01  TYPE CH FOR MONTH.
  ... 

Step 4B: LINEs and COLUMNs

After each group's 01-level entry, code a series of LINE entries, each containing a series of COLUMN entries. COLUMN may be abbreviated as COL. (You will see, in the chapter Report Group Descriptions, that you can also code a dummy group without LINEs or COLUMNs.) You indicate that an entry is at a lower level by increasing the level-number. For instance, you might choose 03 for LINE entries and 05 for COLUMN entries:

01  TYPE PH.
  03  LINE ...
  05  COL ...

After the LINE or COL keyword you may choose one of two ways to specify positioning for your line or field by writing either integer or +integer.

- LINE integer gives you an absolute, that is, fixed, LINE position, counting from 1 at the top of the page, to the maximum given in your PAGE LIMIT. If you
use absolute LINE clauses in a group, the integers must **increase**. You might use this form for the PAGE HEADING in our example.

- **LINE +** integer (+ can also be written PLUS) gives you a **relative** LINE position. It will cause the vertical position to move down that number of lines. For example, **LINE +1** means place on the next line. **LINE +2** means leave one blank line. (This form is rather like WRITE... AFTER ADVANCING..., translated entirely into data terms, of course.) You may **mix** absolute and relative LINE clauses in the same report group, provided that you begin with an absolute LINE. **LINE alone implies LINE +1**.

- **COLUMN integer** gives you an **absolute**, i.e., **fixed**, COLUMN position for the left-hand character of the field, counting character positions from 1 at the left, up to the maximum given in your **LINE LIMIT**. You can also anchor the field at its **CENTER** column by writing **COLUMN CENTER** integer, and you can fix the field at its **RIGHT** column by writing **COLUMN RIGHT** integer. If you use these absolute COLUMN clauses, the integers must **increase** within a LINE.

- **COLUMN +** integer (COLUMN can be shortened to COL and + can be written PLUS) gives you a relative COLUMN position. It will cause the current horizontal position to move right that number of positions from the last column of the preceding field (from zero if this is the first field) to the first column of the current field. For example, **COL +1** means place this field in the next column without a gap, and **COL +2** means leave one column blank.

<table>
<thead>
<tr>
<th>CRUMBLY COOKIE COMPANY</th>
<th>ORDERS</th>
<th>PAGE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>TYPE</td>
<td>QUANTITY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LINE nos</th>
<th>column numbers ...</th>
<th>&gt;&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XXXXXXXXXXXXXXXXXX</td>
<td>20 or + 3</td>
</tr>
<tr>
<td>1</td>
<td>YYYYYYYYYYYYYYYYYY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 or +4</td>
<td>222222</td>
<td></td>
</tr>
</tbody>
</table>

Report Writer writes your report groups **vertically** down the page in the order in which they are **GENERATED**. DETAIL and CONTROL groups (HEADING or FOOTING), which are known as body groups, are first checked to see whether all their lines will fit on the current page. If your report group begins with an **absolute** LINE, report writer will advance to a new page if that **LINE number has been reached or passed**. If your report group begins with a **relative** LINE, report writer checks the size of the report group. If there is no room, or if there is **not enough room for the whole group**, report writer will advance to a new page.
Advancing to a new page involves automatically generating your PAGE FOOTING, if you defined one, followed by your PAGE HEADING, if you defined one. If a body group (CH, DE or CF) begins with a relative LINE, it is positioned on the FIRST DETAIL line, irrespective of the value in the LINE clause. (If you did not code a FIRST DETAIL subclause, it is assumed to be the line immediately following our PAGE HEADING, or line 1 if there is no PAGE HEADING.)

Step 4C: VALUEs and SOURCEs

To complete your Report Group Descriptions, you need to specify the contents of the fields. The two most usual ways, which are sufficient for this example, are as follows:

If the contents of the field consist of fixed text write:

```
VALUE "literal", or simply: "literal"
```

If the contents of the report field come from a field in your COBOL DATA DIVISION, write:

```
PICTURE (or PIC) picture-symbols SOURCE name-of-field
```

Your SOURCE field may be defined in any section of your DATA DIVISION, or it may be a special register such as LINE-COUNTER. The SOURCE keyword may be omitted. You may use subscripts and qualifiers, for example: SOURCE BACK-PAY IN MASTER-RECORD (4). You may also use arithmetic expressions and the word ROUNDED, if needed; for example:

```
PIC $(9)9.99 SOURCE (MONTHLY-PAY * 12) + YEARLY-BONUS ROUNDED
```

You must code a PICTURE clause with the SOURCE clause. This specifies the format in which you would like the field displayed and is the same clause as in elementary COBOL. It can be abbreviated as PIC. Here are two examples:

```
PIC $(8)9.99 SOURCE MONTHLY-PAY
PIC X(32) SOURCE NAME-OF-STUDENT
```

The rules for storing the field work exactly as for the MOVE (or the COMPUTE) statement of elementary COBOL. If your SOURCE refers to a CONTROL field, then you will obtain the value before the control break if report writer is currently processing CONTROL FOOTING groups. This is the only case where you do not obtain the value contained in the field at that instant.

Your layout may require a page number. This is held in a special register (a dedicated internal COBOL location) called PAGE-COUNTER. This location is set up automatically by report writer. There are also LINE-COUNTER and COLUMN-COUNTER special registers.

Suppose that the record that supplies data for the layout above is defined in a standard file as follows:
FD COOKIE-FILE                LABEL RECORDS STANDARD.
  01 COOKIE-RECORD.            
    05 DEPOT                   PIC X(10).
    05 ORDER-DATE              PIC 9(6).
    05 COOKIE-TYPE             PIC X(12).
    05 QTY-ORDERED              PIC 9(4) COMP.
    05 QTY-IN-STOCK             PIC 9(4) COMP.
    05 ORDER-VALUE              PIC S9(5)99 COMP.

Now we are ready to complete the PAGE HEADING group and the first line of the DETAIL group for the layout above, using these new clauses.

REPORT SECTION
RD STOCK-SUMMARY
   LINE LIMIT 132
   FIRST DETAIL 5
   PAGE LIMIT 64
   CONTROL IS DEPOT.
  01 TYPE PH.
    03 LINE 1.
      05 COL 12 VALUE "CRUMBLY COOKIE COMPANY ORDERS".
      05 COL 47 VALUE "PAGE".
      05 COL +2 PIC Z9 SOURCE PAGE-COUNTER.
    03 LINE 3.
      05 COL 7 VALUE "DATE TYPE QUANTITY VALUE OF ORDER".
  01 COOKIE-LINE TYPE DE.
    03 LINE + 2.
      05 COL 4 PIC 99/99/99 SOURCE ORDER-DATE.
      05 COL 16 PIC X(12) SOURCE COOKIE-TYPE.
      05 COL 29 PIC ZZZ9 SOURCE QTY-ORDERED.
      05 COL 41 PIC $(5)9.99 SOURCE ORDER-VALUE.

Step 4D: Conditional Items

There is one item in our layout that we do not want to produce every time. This is the message "OUT OF STOCK". We deliberately allowed for it by including it in the "box" we drew round the typical DETAIL group. It should only be produced if the condition

QTY-ORDERED > QTY-IN-STOCK

is true. To make any item depend on a condition's being true or false, use the clause:

PRESENT WHEN condition
Report writer will then automatically test the condition when it is about to produce the item. If the condition is false, the item is **ignored**. If you put the clause on a **LINE** entry, it is the **whole line** that is ignored. (You can in fact put it at any level. When a group field is ignored, so are all the fields within the group.) In the case here, we do want the whole line to be ignored if the condition is false, so the following would be a valid description for the second line for the DETAIL group:

```
  03  LINE PRESENT WHEN QTY-ORDERED > QTY-IN-STOCK.
      05  COL 2    VALUE "**OUT OF STOCK".
```

**Step 4E: Totalling**

The sample layout tells you to produce a **total** in the **CONTROL FOOTING** group. COBOL Report Writer allows you to produce totals from virtually any numeric fields, and you may do it in any **TYPE** of group. To produce a total, follow these two simple steps:

1. Put a **data-name** at the front of the entry you want totalled.
2. In another entry, use the clause: **SUM OF** data-name instead of **SOURCE** or **VALUE**.

First, you must go back to our coding for the DETAIL group above and add a data-name to the entry for **ORDER-VALUE**. For example, you could re-write the last entry as:

```
  05  REP-ORD-VAL  COL 41  PIC $(5)9.99  SOURCE ORDER-VALUE.
```

where **REP-ORD-VAL** is a new data-name of your choice. Now you can code the **CONTROL FOOTING** group for our layout:

```
  01  TYPE CF.
      03  LINE + 3.
      05  COL 16   VALUE "TOTALS: DEPOT".
      05  COL + 2  PIC X(10)      SOURCE DEPOT.
      05  COL 41   PIC $(5)9.99   SUM OF REP-ORD-VAL.
      03  LINE.
      05  COL 41   VALUE "========".
```

You may have any number of **CONTROLS** in your program, and you may have a **CONTROL HEADING** as well as a **CONTROL FOOTING** report group for each control.

**Step 5: Code the SELECT...ASSIGN and FD**

COBOL Report Writer requires a **standard COBOL file** to which to write your output. So, to begin with, you need a **SELECT...ASSIGN** clause and an **FD**. The **FD** entry may contain any clauses that you would normally use for a report file, plus a new clause:
REPORT IS name-of-report. You should not need a record description to follow. For our example, you might write:

(D) IDENTIFICATION DIVISION.
... (other paragraphs as normal) ...
FILE-CONTROL.
   SELECT COOKIE-FILE ASSIGN TO UT-S-DATAIN.
   SELECT STOCK-PRINT ASSIGN TO UT-S-LIST01.
DATA DIVISION.
FILE SECTION.
FD COOKIE-FILE.
   ... (description as in step 4C) ...
FD STOCK-PRINT
   ...
REPORT IS STOCK-SUMMARY.
WORKING-STORAGE SECTION.
01 WS-EOF PIC X VALUE "N".

Step 6: Code the PROCEDURE DIVISION

Report Writer is entirely under the control of your program, but at a higher level than is the case with elementary COBOL. This means that no action will be taken until your program executes a report writer statement. There are three of these:

1. **INITIATE** name-of-report

This statement initializes your report at the start of the whole process. Your program must do this before it is allowed to execute any other report writer statements. It does not open the file. **Name-of-report** is the data-name you wrote right after the RD.

2. **GENERATE** detail-name

This statement generates one instance of a DETAIL report group. **Detail-name** is the data-name you used to name your DETAIL report group. The GENERATE also performs all the other actions that might be necessary before the DETAIL report group is output, namely:

- It tests for control breaks (if your report has a CONTROL clause) and, if necessary, produces CONTROL FOOTING and CONTROL FOOTING report groups.

- It checks that your DETAIL report group will fit completely on the current page (assuming that your report has a PAGE clause). If not, it produces your PAGE FOOTING report group (if there is one defined in your report), advances to a new page and produces your PAGE HEADING report group (if there is one defined in your report). Unless you explicitly allow it with a MULTIPLE PAGE clause, report writer never splits your report group over two pages.

- If it is the first occasion after the INITIATE, the GENERATE statement will output any REPORT HEADING, your PAGE HEADING, and all your CONTROL HEADING groups before generating your DETAIL report group.
Your **CONTROL HEADING** and **CONTROL FOOTING** report groups are also subject to the page-fit test. They are treated similarly to DETAIL report groups. These three **TYPES** are often referred to as **body groups**, because they fit into the "body" of the page (between the **PAGE HEADING** and **FOOTING**) and usually contain the most important information.

3. **TERMINATE** name-of-report

This statement ends your report and produces any final items that are required at the end of the report, namely:

- All **CONTROL FOOTING**s up to the highest level defined;
- The last **PAGE FOOTING** (if defined);
- The **REPORT FOOTING** (if defined).

To produce output for a simple report layout from standard files, the following logical structure should apply:

```
OPEN input and report files
INITIATE report-name

↓

for each input record

READ input file
GENERATE detail-name

↓

TERMINATE report-name
CLOSE all files
```

If your input is from a **database**, or reaches your program's DATA DIVISION by some means other than from a standard file, you will need to replace the **OPEN** and **CLOSE** for the input files and the **READ** by more appropriate statements.

To produce a more complex layout, you would probably define several different **DETAIL** report groups and decide in your program when to **GENERATE** one or the other. For our example, the following would be a suitable complete **PROCEDURE DIVISION**:
(E)

PROCEDURE DIVISION.
PROGRAM-START.
  OPEN INPUT COOKIE-FILE, OUTPUT STOCK-PRINT
  INITIATE STOCK-SUMMARY
  PERFORM NEXT-RECORD
  PERFORM GENERATE-LINE THRU NEXT-RECORD
      UNTIL WS-EOF = "Y"
  TERMINATE STOCK-SUMMARY
  CLOSE STOCK-PRINT, COOKIE-FILE
STOP RUN.
GENERATE-LINE.
  GENERATE COOKIE-LINE.
NEXT-RECORD.
  READ COOKIE-FILE AT END MOVE "Y" TO WS-EOF.

Place this code after the code in the code samples (D), (A), (B) and (C) (in that order) and you have a complete program.

You may use report writer "verbs" just as you would use any other PROCEDURE DIVISION statements. So your program may do many other tasks apart from just producing your report output.
1.3 More about COBOL Report Writer

1.3.1 More about Files and Reports

You may describe as many reports as you like per program. Each report has its own RD entry, followed by one or more Report Group Descriptions. Separate reports may either be assigned to separate files or to the same file, in which case you could write:

```
FD PRINT-FILE
REPORTS ARE MAIN-REPORT, SUMMARY-REPORT.
```

This last approach is useful where you need to produce a report that has distinct sections, perhaps with different page headings. So a single physical report (as the end-user sees it) may consist of several different logical reports (as the programmer sees them), all written to the same file.

You can also direct your report output to a special Independent Report File Handler, designed to process the output from reports in a particular way. To use the special file handler, write the extra clause: MODE IS mnemonic-name in the SELECT clause. It does not affect the FD or any other statements in your program.

1.3.2 More about the RD Entry

Apart from the clauses used in our example, there are several other clauses that you may write in your RD entry. They are all explained in detail in the chapter Report Files and RD Entries. Here is a brief summary:

- **LAST DETAIL** gives the last line on which a DETAIL or CONTROL HEADING report group may appear.
- **LAST CF** (or LAST CONTROL FOOTING or just FOOTING) gives the last line on which a CONTROL FOOTING group may appear. If you do not specify it, a default value is assumed for it. You can use it to ensure that a CONTROL FOOTING will never appear at the top of a page (since there will always be space reserved for it at the bottom of the previous page).
- **OVERFLOW** and **SUM OVERFLOW** enable you to specify the action that takes place if any arithmetic expressions or totals defined in your report groups are too large for the report field or involve dividing by zero.
- **CODE** is used when your output report data must have extra unprinted information placed at the start of each record, or when you need to pass information to a special Independent Report File Handler (see Independent Report File Handlers). For example, if your installation has provided a file handler to do spooling and restart, you may be required to provide a key by which restart would be done. You would then write: CODE IS name-of-key-field.
- **ALLOW SOURCE SUM CORR** and **ALLOW NO SOURCE SUM CORR** determine whether the ANS-68 or the ANS-85 rules will be used for calculating certain SUM fields. ALLOW NO SOURCE SUM CORR is assumed in default in the version as supplied. A description of this process will be found in Installation and Operation.
- **GLOBAL** makes the report available to nested programs.
1.3.3 More about CONTROLS

Each RD entry may have a CONTROL clause, and you may write the names of any number of fields in your program. Your control fields must have a hierarchy and must be listed in order from the highest down to the lowest. When your program issues a GENERATE, report writer tests each control field in order, beginning with the highest. If it detects a change (a control break), this process ends. Report writer will then automatically take additional action before it produces the DETAIL group, depending on which CONTROL FOOTING or CONTROL HEADING report groups (if any) you defined.

COBOL Report Writer keeps an internal copy of each of your control fields so that it can test for a control break by comparing them with the new values on each GENERATE. Before it produces your CONTROL FOOTING report groups, it temporarily stores these previous values back into the control fields. So if your CONTROL FOOTING refers to a control field, as a SOURCE for example, you will get the previous or pre-break value, even though the original control field has changed in value.

You may also want a major heading and a major footing at the very start and end of your report. For example, you may want to produce grand totals for the entire report. If so, you may use the reserved word REPORT or FINAL. This is the highest possible control level. If you use it, it must therefore be first in the list of controls in your CONTROLS clause.

There may be fields other than totals or lines that should be produced only once after a control break. These may be inside a report group, where you cannot make use of a separate CONTROL HEADING. You can define them by writing:

```
PRESENT AFTER NEW name-of-control-field
```

or

```
ABSENT AFTER NEW name-of-control-field
```

The field or line will then appear only on the first occasion after a control break at the level you indicate. (Alternatively, if you use ABSENT AFTER..., the field or line will not appear the first time and will appear every time thereafter until the next control break.) You may also write PRESENT or ABSENT AFTER NEW PAGE, indicating that you want the field or line to appear (or disappear) only on the first occasion after a page advance. Finally, you may code both the control-field and PAGE operands in one clause.

It is possible for a report to have no DETAIL groups at all. This case is called summary reporting. The only body groups coded are CONTROL FOOTING - and possibly CONTROL HEADING - groups. Therefore, since you have no name of a DETAIL group to give in your GENERATE statement, you write: GENERATE report-name.

The next example illustrates all the points made in this section. There are three levels of control, including REPORT, each with a CONTROL FOOTING.
**RD SUBSCRIPTIONS**  
FIRST DE 4 PAGE LIMIT 60  
CONTROLS REPORT, YEAR, MONTH.

01 TYPE PH ... etc ...

| 01 TYPE CF FOR MONTH LINE + 1.  
| 05 COL 1 PIC 9(4) SOURCE YEAR  
| PRESENT AFTER NEW YEAR.  
| 05 COL 6 PIC XXX  
| SOURCE W-MONTH-NAME (MONTH).  
| 05 COL 16 PIC $(5)9 SUM OF FULL.  
| 05 COL 28 PIC $(5)9 SUM OF OFFP.  
| 01 TYPE CF FOR YEAR NEXT GROUP + 1.  
| 03 LINE + 2.  
| 05 COL 1 PIC 9(4) SOURCE YEAR.  
| 05 COL 6 VALUE "TOTALS:".  
| 05 COL 16 PIC $(5)9 SUM OF FULL.  
| 05 COL 28 PIC $(5)9 SUM OF OFFP.  
| 03 LINE + 1 COLS 16 28  
| VALUE "=====".

*Blank line:*

03 LINE + 1.

| 01 TYPE CF FOR REPORT LINE + 1.  
| 05 COL 16 PIC $(5)9 SUM OF FULL.  
| 05 COL 28 PIC $(5)9 SUM OF OFFP.  

PROCEDURE DIVISION.

...  
GENERATE SUBSCRIPTIONS

**SPORTS CLUB: TENNIS SECTION**

**SUBSCRIPTIONS:** FULL OFF-PEAK

<table>
<thead>
<tr>
<th>Year</th>
<th>FULL</th>
<th>OFF-PEAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997 JAN</td>
<td>$4000</td>
<td>$10000</td>
</tr>
<tr>
<td>FEB</td>
<td>$3000</td>
<td>$9000</td>
</tr>
<tr>
<td>MAR</td>
<td>$1500</td>
<td>$8000</td>
</tr>
<tr>
<td>APR</td>
<td>$1000</td>
<td>$3000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DEC</td>
<td>$1500</td>
<td>$5500</td>
</tr>
<tr>
<td><strong>1997 TOTALS:</strong></td>
<td>$26500</td>
<td>$43000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>FULL</th>
<th>OFF-PEAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998 JAN</td>
<td>$2000</td>
<td>$8000</td>
</tr>
<tr>
<td>FEB</td>
<td>$2500</td>
<td>$9500</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DEC</td>
<td>$2000</td>
<td>$4500</td>
</tr>
<tr>
<td><strong>1998 TOTALS:</strong></td>
<td>$34700</td>
<td>$12800</td>
</tr>
<tr>
<td><strong>GRAND TOTALS:</strong></td>
<td>$61200</td>
<td>$55800</td>
</tr>
</tbody>
</table>

There is much more about CONTROLS in the main section (see 2.6 **CONTROL clause**).

### 1.3.4 More about TYPES

In the diagram on the next page, you can see all **seven** types of report group in use in the same layout. You may choose any or all of the seven types in a report, and none of them are compulsory (except that you must have at least one body group (CH, DE, or CF)). You cannot have more than **one** REPORT HEADING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING, and you cannot have more than **one** CONTROL HEADING and CONTROL FOOTING for **each** control level. But you can code **any number** of DETAIL groups.

If your report needs a particularly long or complex REPORT HEADING or REPORT FOOTING, or if your report layout changes completely at a later stage, code a second **separate RD** with its own Report Group Descriptions following and expand the REPORT clause of your FD to include the second report-name. (You may associate as many RDs as you like with the same FD.) For further details, see 2.2 **Report Files**, 2.3 **REPORT SECTION and RD**, and 5.1 **Multiple Reports**.
<table>
<thead>
<tr>
<th>SPORTS CLUB EXPENSES SUMMARY</th>
<th>1999</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPORT: GOLF</td>
<td></td>
</tr>
<tr>
<td>JAN</td>
<td></td>
</tr>
<tr>
<td>01 Mole Damage $350</td>
<td></td>
</tr>
<tr>
<td>23 New Car Park $2250</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>JAN GOLF TOTAL $2600</td>
<td></td>
</tr>
<tr>
<td>FEB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC TOTAL $1400</td>
<td></td>
</tr>
<tr>
<td>YEAR GOLF TOTAL $15000</td>
<td></td>
</tr>
<tr>
<td>SPORT: CRICKET</td>
<td></td>
</tr>
<tr>
<td>JAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTINUED ...</td>
<td></td>
</tr>
<tr>
<td>END OF EXPENSES SUMMARY</td>
<td></td>
</tr>
</tbody>
</table>

**Report showing all 7 Types of group:**

- **TYPE RH or Report Heading**
- **TYPE PH or Page Heading**
- **TYPE CH for Sport**
  - or Control Heading for Sport (higher control heading)
- **TYPE CH for Month**
  - or Control Heading for Month (lower control heading)
- **TYPE DE or Detail**
- **TYPE CF for Month**
  - or Control Footing for Month (lower control footing)
- **TYPE CF for Sport**
  - or Control Footing for Sport (higher control footing)
- **TYPE PF or Page Footing**
- **TYPE RF or Report Footing**
1.3.5 Automatic Repetition

If your report has a series of fields or lines or groups of similar layout or format, it is usually possible to save time in coding by writing one multiple clause instead of several entries with single-operand clauses. Here is a list of cases:

A. **VALUES** in Consecutive Fields

If you have consecutive fields in a line containing literals, you may code multiple **COLUMNS** and **VALUE** clauses to avoid writing several entries:

```
SPORTS CLUB SUBSCRIPTIONS
TENNIS  GOLF  SWIMMING  CRICKET
03 LINE + 1.
  05 COLS 1 9 15 25
  VALUES "TENNIS" "GOLF"
       "SWIMMING" "CRICKET".
```

B. **SOURCES** in Consecutive Fields, with Same PICTURE

If you have several consecutive fields in a line with the same PICTURE (or if you can expand shorter PICTUREs to match longer ones), you may code multiple **COLUMNS** and **SOURCE** clauses in one entry:

```
SPORTS CLUB SUBSCRIPTIONS
TENNIS  GOLF  SWIMMING  CRICKET
$238   $340    $500      $350
05 COLS 1 8 16 26  PIC $$$$9
  SOURCES TENNIS GOLF SWIMMING CRICKET.
```

Here, as usual, each **SOURCE** field is a data item defined in the DATA DIVISION of your program (in this particular case, a numeric item). **VALUE** and **SOURCE** clauses cannot be combined within the same multiple clause.

Using a single entry like this also makes it easy to total a series of fields by coding just one **SUM** entry:

```
SPORTS CLUB SUBSCRIPTIONS
TENNIS  GOLF  SWIMMING  CRICKET  TOTAL
$238   $340    $500      $350  $1428
05 R-SUBS COLS 1 8 16 26  PIC $$$$9
  SOURCES TENNIS GOLF SWIMMING CRICKET.
05 COL 35  PIC $(5)9  SUM OF R-SUBS.
```

If separate entries are used, you would have to total them by writing either: **SOURCE TENNIS + GOLF + SWIMMING + CRICKET**, or **SUM R-TENNIS R-GOLF R-SWIMMING R-CRICKET**, placing these data-names on the entries in turn.
C. Regularly Spaced COLUMNS

If the gap between successive fields is regular, you need not give a COLUMN for each one. Instead, you can combine an OCCURS clause and a STEP phrase:

```
SPORTS CLUB SUBSCRIPTIONS
TENNIS  GOLF  RUGBY  SQUASH
$238   $340   $500   $350
05 COL 1 OCCURS 4 STEP 7 PIC $$$$9
SOURCES TENNIS GOLF RUGBY SQUASH.
```

You can also use OCCURS with a relative COLUMN to provide the gap, in which case the STEP phrase is unnecessary.

D. Repeating the Same VALUE

You can combine a single VALUE with an OCCURS clause or a multiple COLUMNS clause, in which case the VALUE is simply repeated:

```
($)     ($)     ($)     ($)
238     340     500     350
05  COL 2   OCCURS 4 STEP 8 VALUE "($)".
(or 05  COLS 2, 10, 18, 26 VALUE "($)".)
```

A single-operand SOURCE field can be similarly repeated, although the occasions for doing so are rarer.

E. Repeating LINE

The LINE clause also has a multiple form. You may also combine an OCCURS clause with a single-operand LINE clause. (In the latter case, if you use STEP, as you must if the LINE is absolute, it refers to the vertical distance.) If you use a SOURCE, the entire table of SOURCE items must be "read into memory" first. Within the repeating LINE, you may have multiple VALUES and SOURCES clauses. This enables you to improve clarity by stacking your heading values in one place:

```
NEW    OLD    MEMBERS
MEMBERS MEMBERS LEAVING
03  LINES 2, 3.
05  COL 1 VALUES " NEW"
          "MEMBERS".
05  COL 10 VALUES " OLD"
              "MEMBERS".
05  COL 19 VALUES " MEMBERS"
                "LEAVING".
```

(You don’t have to code the literals vertically like this, but it does help the eye.)

F. Variable Number of Repetitions

If the number of repetitions is variable, you should use the OCCURS clause’s keyword TO and DEPENDING ON phrase, whose operand can be any data-name or arithmetic expression. Report writer will then dynamically calculate the actual number of repeats.
present on each occasion. It is valid for there to be no occurrences, so your minimum
can be zero. Any "unused" repeats are treated as **ABSENT**:

<table>
<thead>
<tr>
<th>FAMILY MEMBERSHIPS</th>
<th>AMOUNT DUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES PETER MARY IAN SARAH</td>
<td>$240</td>
</tr>
<tr>
<td>SMITH ALAN DEBBIE</td>
<td>$120</td>
</tr>
<tr>
<td>ROBERTS SUSAN TOM IONA</td>
<td>$180</td>
</tr>
</tbody>
</table>

The same method can be used for **LINEs**. If a body group has a **variable** number of
lines and they are all **relative**, report writer will take into account **only those lines**
actually present when applying its page-fit test.

**G. SOURCE Items in a Table**

If you need to output **SOURCE** items that are held in a table, report writer will
automatically **vary** an internal data-name which you can then use as a subscript. You
can specify a **FROM** value for the starting point and a **BY** value for the increment for
your subscript, but these are assumed to be **1** if you omit them:

<table>
<thead>
<tr>
<th>SPORTS CLUB SUBSCRIPTIONS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TENNIS GOLF RUGBY SQUASH</td>
<td>$238 $340 $500 $350</td>
</tr>
</tbody>
</table>

**You choose your own data-name for the VARYING clause, but it must **not** be defined
anywhere as a data item in your program. You can reuse the **same data-name** many
times in the REPORT SECTION, except where the VARYING clauses are nested.**
You may combine VARYING with a multiple COLUMNS or LINES clause, as well as with an OCCURS clause, and you may output results in more than one dimension. In the next example, the SPORT fields are printed in reverse order:

<table>
<thead>
<tr>
<th>SPORTS CLUB 4-YEAR SUBSCRIPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUASH</td>
</tr>
<tr>
<td>1996 $180</td>
</tr>
<tr>
<td>1997 $196</td>
</tr>
<tr>
<td>1998 $223</td>
</tr>
<tr>
<td>1999 $238</td>
</tr>
</tbody>
</table>

03 LINE OCCURS 4 VARYING YEAR-NO.
05 COL 2 PIC 9(4) SOURCE 1988 + YEAR-NO.
05 COL 8 OCCURS 4 STEP 7 VARYING SPORT-NO FROM 4 BY -1 PIC $$$$9 SOURCE SPORT (YEAR-NO, SPORT-NO).

H. Repeating Whole Groups Horizontally

The REPEATED clause enables you to place whole groups side-by-side. On each GENERATE, report writer will place the group in an internal buffer, until the last of each set arrives, whereupon the whole set will be printed side-by-side. You should define only the left-hand group.

<table>
<thead>
<tr>
<th>CRICKET FIXTURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ST TEAM VS OLD C.T.'S</td>
</tr>
<tr>
<td>ON 21ST APRIL</td>
</tr>
<tr>
<td>AWAY</td>
</tr>
<tr>
<td>2ND TEAM VS S.RICHMOND</td>
</tr>
<tr>
<td>ON 21ST APRIL</td>
</tr>
<tr>
<td>HOME</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1ST TEAM VS OLD C.T.'S</td>
</tr>
<tr>
<td>ON 28TH APRIL</td>
</tr>
<tr>
<td>HOME</td>
</tr>
<tr>
<td>2ND TEAM VS S.RICHMOND</td>
</tr>
<tr>
<td>ON 28TH APRIL</td>
</tr>
<tr>
<td>AWAY</td>
</tr>
</tbody>
</table>

01 CRICKET-FIXTURE TYPE DE REPEATED 2 TIMES EVERY 30 COLS.
03 LINE + 3.
05 COL 4 VALUE "1ST" WHEN REPEATED-COUNTER = 1
05 COL 8 VALUE "TEAM VS".
05 COL + 2 PIC X(10) SOURCE OPPONENTS-NAME.
03 LINE + 1 ... etc ...
03 LINE + 1 ... etc ...

If a different DETAIL group is GENERATED - say SOCCER-FIXTURE - or if your program issues a TERMINATE, and there are still left-hand groups in the buffer, these buffered groups are output first, padded out with blank entries on the right where necessary.
I. Different Levels Using the same CONTROL FOOTING

You will have noticed from some of the preceding examples that a lower CONTROL FOOTING and a higher CONTROL FOOTING often have a very similar layout and you may wish you could code a single report group and use it for any number of control levels. You can do this simply by listing more than one control in the TYPE clause, for example TYPE CF FOR REPORT, YEAR, MONTH or just CF FOR ALL. Any SUM totals are then automatically rolled forward up to each higher level. If any CONTROL FOOTING has a different layout from the others, you can use PRESENT WHEN CONTROL IS YEAR, PRESENT WHEN CONTROL IS MONTH, and so on to vary it.

1.3.6 More about Totalling

There are many other ways to use the SUM clause to produce totals. As well as totalling from one group to another, you may form totals within the same group. Here’s how you might enhance our four-yearly table with row and column totals. (Absorb this example slowly.)

<table>
<thead>
<tr>
<th>SPORTS CLUB 4-YEAR SUBSCRIPTIONS</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUASH</td>
<td>RUGBY</td>
<td>GOLF</td>
<td>TENNIS</td>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>$180</td>
<td>$300</td>
<td>$445</td>
<td>$290</td>
<td>$1215</td>
</tr>
<tr>
<td>1997</td>
<td>$196</td>
<td>$280</td>
<td>$440</td>
<td>$310</td>
<td>$1226</td>
</tr>
<tr>
<td>1998</td>
<td>$223</td>
<td>$320</td>
<td>$450</td>
<td>$320</td>
<td>$1313</td>
</tr>
<tr>
<td>1999</td>
<td>$238</td>
<td>$340</td>
<td>$500</td>
<td>$350</td>
<td>$1428</td>
</tr>
<tr>
<td>TOTALS</td>
<td>$827</td>
<td>$1240</td>
<td>$1835</td>
<td>$1270</td>
<td>$5182</td>
</tr>
</tbody>
</table>

(Totals may also be specified at the top or on the left.)

03 LINE OCCURS 4 VARYING YEAR-NO.
   05 COL 2 PIC 9(4) SOURCE 1995 + YEAR-NO.
   05 R-VAL COL 8 OCCURS 4 STEP 7
      VARYING SPORT-NO FROM 4 BY -1
      PIC $$99 SOURCE SPORT (YEAR-NO SPORT-NO).
   05 COL 37 PIC $(5)9 SUM OF R-VAL.
03 LINE COLS 8 15 22 29 37 VALUE "-----".
03 LINE.
   05 T-VAL COL 8 OCCURS 4 STEP 7 PIC $$99 SUM OF R-VAL.
   05 COL 37 PIC $(5)9 SUM OF T-VAL.

Report writer totals repeating values automatically along the horizontal or vertical axes. Notice that you should not place any subscripts after the data-name that is the operand of the SUM clause.

A SUM may be combined in an entry with a multiple COLUMN (or LINE) clause to give you a series of totals of another repeating entry with the same number of repetitions, as you see in the last line of this example:
CLUB OUTGOINGS IN 1999

<table>
<thead>
<tr>
<th>MONTH</th>
<th>BUILDINGS</th>
<th>INTERIOR</th>
<th>WAGES</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN</td>
<td>$80</td>
<td>$445</td>
<td>$2290</td>
<td>$121</td>
</tr>
<tr>
<td>FEB</td>
<td>$170</td>
<td>$350</td>
<td>$440</td>
<td>$2260</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DEC</td>
<td>$190</td>
<td>$440</td>
<td>$2260</td>
<td>$1130</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>TOTALS</td>
<td>$3120</td>
<td>$1240</td>
<td>$13250</td>
<td>$34930</td>
</tr>
</tbody>
</table>

03 LINE OCCURS 12 VARYING MONTH.
05 COL 2 PIC XXX SOURCE WS-MONTH-NAME (MONTH).
05 R-OUTGOINGS COLS 10 20 29 37 PIC $(5)9 SOURCES BUILDINGS INTERIOR WAGES TAX.

... Line.
05 COL 2 VALUE "TOTALS".
05 COLS 9 19 28 36 PIC $(6)9 SUM OF R-OUTGOINGS.

The total line may also be in a different group from the repeating line. If so, you might then remove the OCCURS 12 on the first LINE entry and GENERATE the group containing it 12 times.

As well as totalling a field using SUM, you may count the occurrences using the COUNT clause. COUNT simply adds 1 each time instead of the value of the field. You may COUNT the number of times any REPORT SECTION item appears, including LINEs or whole groups. All multiple occurrences contribute to the COUNT.

You may use SUM and COUNT as terms of a SOURCE expression. Be sure to enclose each term in parentheses. For example, to find the average amount of the subscription of our four sports above, you may write:

01 MAIN-GROUP TYPE DE.
05 R-SUBS COL 1 PIC $$$$9 SOURCE SPORT-SUBSCRIPTION.

... Type CF.
05 COL 1 PIC $$$$9 SOURCE IS (SUM OF R-SUBS) / (COUNT OF R-SUBS) ROUNDED.

(As usual, the words SOURCE IS are optional.) If the divisor (the COUNT term above) happens to be zero, report writer will detect the error, unless you write OVERFLOW PROCEDURE IS OMITTED in your RD statement. The action taken depends on what, if anything, you coded in the OVERFLOW PROCEDURE clause. (By default, report writer will detect the error and write an error message on your terminal or job log, leaving the field blank.)
You may also total numeric fields directly from other sections in your DATA DIVISION. (With the older ANS COBOL report writer this method was necessary to obtain totals. You coded the name of the FILE, WORKING-STORAGE, or LINKAGE SECTION item as an operand of the SUM clause in the lowest-level CONTROL FOOTING.) With such external items, you may use subscripts, and you may also SUM an arithmetic expression; for instance:

```
05 COL 1 PIC ZZZ9 SUM OF (WS-INCOME - WS-TAX).
```

If the item does not already appear as a SOURCE, this is the only method of totalling it. So this technique is useful where you require totals of a field but do not want to show the individual values that were added to produce the total. Its main disadvantage is that it may not be clear to the reader of your program exactly when the values are added into the total. See the remainder of this publication for a discussion of the relevant rules.

### 1.3.7 More about Conditional Entries

You have already seen how a single COBOL condition may be used to decide whether to output a report field. Multiple-choice entries are used when you have several possible contents for a field. Just write a series of SOURCE or VALUE clauses, each followed by `PRESENT WHEN` condition. (The keyword `PRESENT` is often omitted in a multiple-choice entry.) The period does not come until the end. Report writer examines all the conditions in sequence until it finds the first that is true and then uses the VALUE or SOURCE associated with that true condition. `WHEN OTHER` can be used to indicate "when none of the given conditions is true". (Compare the use of `ELSE` in elementary COBOL.) Study the following example:

<table>
<thead>
<tr>
<th>TITLE/NAME</th>
<th>PAY</th>
<th>AMOUNT DUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR. CODER</td>
<td>M</td>
<td>$10</td>
</tr>
<tr>
<td>MISS PROGRAMMER</td>
<td>Y</td>
<td>$120</td>
</tr>
<tr>
<td>ANALYST</td>
<td>Y</td>
<td>$160</td>
</tr>
</tbody>
</table>

```
03 LINE.
  05 COL 1 VALUE "MR. " WHEN TITL = 1
  VALUE "MRS. " WHEN TITL = 2
  VALUE "MISS " WHEN TITL = 3
  VALUE "DR. " WHEN TITL = 4.

*NB: One period after last item!

  05 COL +1 PIC X(12) SOURCE SURNAME.
  05 COL 24 VALUE "M" WHEN YEARLY-FLAG = 0
  VALUE "Y" WHEN OTHER.

  05 COL 40 PIC $$$$9 SOURCE (SUBSCRIPTION / 12) WHEN YEARLY-FLAG = 0
  SOURCE SUBSCRIPTION WHEN OTHER.
```

Note that the third person in our list, ANALYST, has no title because there is no `WHEN OTHER` ("catch-all") in the choice of titles.
You may produce many useful effects with the PRESENT WHEN clause by causing fields or lines, relative or absolute, to appear or disappear at certain times. If a relative entry (COLUMN + ... or LINE + ...) follows an entry that may or may not be PRESENT, its position is variable:

<table>
<thead>
<tr>
<th>UNPAID SUBSCRIPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESTER ( CRICKET SQUASH ): $250</td>
</tr>
<tr>
<td>CODER ( RUGBY ): $100</td>
</tr>
<tr>
<td>ANALYST ( TENNIS SQUASH RUGBY ): $450</td>
</tr>
</tbody>
</table>

03 LINE.
05 COL 1 PIC X(10) SOURCE SURNAME.
05 COL 12 VALUE "( ".
05 COL +1 VALUE "CRICKET " PRESENT WHEN CRICKET-FLAG = 1.
05 COL +1 VALUE "TENNIS " PRESENT WHEN TENNIS-FLAG = 1.
05 COL +1 VALUE "SQUASH " PRESENT WHEN SQUASH-FLAG = 1.
05 COL +1 VALUE "RUGBY " PRESENT WHEN RUGBY-FLAG = 1.
05 COL +1 VALUE ")".
05 COL +2 PIC $$$9 SOURCE UNPAID-SUBS.

The ABSENT WHEN clause has the same effect as PRESENT WHEN except that you write the negative condition. Other conditional clauses are PRESENT AFTER (previously known as GROUP INDICATE) and ABSENT AFTER. Instead of checking a standard COBOL condition, these clauses test whether there has been a page advance or a control break since the group was last produced. You may write PRESENT AFTER NEW PAGE, PRESENT AFTER NEW control-id, or PRESENT AFTER NEW control-id OR PAGE:

<table>
<thead>
<tr>
<th>SURVEY OF MEMBERSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1997</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1998</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Without the PRESENT AFTER clause, the YEAR would appear on each line.

<table>
<thead>
<tr>
<th>SURVEY OF MEMBERSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1999</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
1.4 **Some Shorter Forms**

COBOL Report Writer offers you several ways to shorten the amount of code you write. You have already seen several, such as shortening **COLUMN** to **COL**. Of course, the shorter forms may not always be clearer, and you may decide not to adopt them all. Here are some of them:

1. **The keywords** **TYPE**, **SOURCE**, **VALUE**, and **PRESENT** may be omitted. This reduces your coding effort at a cost of making your program less readable to a maintenance programmer unfamiliar with report writer.

2. If you do **not** code a **TYPE** clause in a level-01 entry, **TYPE DETAIL** is implied.

3. You may write **LINE** and **COLUMN** (or **COL**) in the same entry, provided that there is only one item in the **LINE**. So you could code:

   ```
   03 LINE + 1  COL 20  VALUE "GOLF".
   ```

   instead of:

   ```
   03 LINE + 1.
   05 COL 20  VALUE "GOLF".
   ```

   If there is second item in the line, this second method is the only way.

4. You may code the **LINE** clause in the level-01 entry, provided that there is only one **LINE** in the report group. So you could code:

   ```
   01 ACCOUNT-ENTRY TYPE DE  LINE + 1.
   ```

   instead of:

   ```
   01 ACCOUNT-ENTRY TYPE DE.
   05 LINE + 1.
   ```

   If there is another **LINE** in the report group, this second method is the only way.
1.5 Other Features

1.5.1 Variable-Length Fields

If any of your report fields are to take up a variable number of columns, use the left-shift (or “squeeze”) symbols `<` and `>` in the PICTURE. The examples below show the effect of these symbols:

```
MEMBERS AND CHILDREN'S AGES

CODER: MANDY(7), TOM(5).
TESTER: ALAN(11), HILARY(9), JASON(8).
ANALYST: ANGELO(8).
```

```
03 LINE.
  05 COL 1   PIC <X(12)> SOURCE SURNAME.
  05 COL + 1 VALUE ": ".
  05 OCCURS 1 TO 9 DEPENDING ON NUMBER-OF-CHILDREN
      VARYING R-CHILD-SUB.
  07 COL + 1 PIC <X(8)>   SOURCE FORENAME (R-CHILD-SUB).
  07 COL + 1 VALUE "(".
  07 COL + 1 PIC <9>9    SOURCE AGE (R-CHILD-SUB).
  07 COL + 1 VALUE ")".
  07 COL + 1 VALUE ", " WHEN R-CHILD-SUB < NUMBER-OF-CHILDREN
      VALUE ". " WHEN OTHER.
```

The reason why `PIC <9>9` was coded rather than `PIC <99>` against the child's age is to prevent a value of zero from causing the field to vanish completely. In the other cases, the closing `>` symbol is optional.

Now imagine this same code with all the "<" and ">" symbols removed from the PICTURES. This is what would appear:

```
MEMBERS AND CHILDREN'S AGES

CODER       : MANDY   (07), TOM     (05).
TESTER      : ALAN    (11), HILARY  (09), JASON   (08).
ANALYST     : ANGELO  (08).
```

1.5.2 Insertion Characters

As well as by using standard PICTURE symbols such as "/", "0" and "B", you can place any additional characters into your report field by placing them within "quotes" (or 'apostrophes') within the PICTURE. For example, to print a percentage:

```
PIC ZZZ9.99"%" SOURCE 100 * COST / TOTAL ROUNDED
```
1.5.3 COLUMN CENTER and RIGHT

You can specify the center or the right-hand column as an anchor point, rather than just the left-hand column. To do so, write COLUMN CENTER or COLUMN RIGHT. (CENTRE is an alternative spelling.) In the case of COLUMN CENTER, if your field has an even number of characters, the odd character goes on the right. This feature saves you time when you are working with fields of different lengths, in different lines, that should appear centered or right-aligned in a "stack". It also simply saves you the effort of counting out the length of a field in order to center it. See the following cases, all of which produce the same result:

Expenditure
▲ ▲ ▲
COL 15   COL RIGHT 25
COL CENTER 20

If your field is variable-length, report writer first takes the actual size of the field before it positions it. In this way a name, title, etc. can be centered or right-aligned:

JOHN CODER
12 WALLINGTON ROAD
EGHAM

03 LINE OCCURS 1 TO 5 DEPENDING ON NO-OF-ADDR-LINES VARYING R-ADDR-LINE.
05 COL CENTER 20 PIC <X(32) SOURCE ADDR-LINE (R-ADDR-LINE).

1.5.4 NEXTGROUP Clause

Use this clause when you want to create extra space between report groups or when you need to ensure that a particular report group is the last on the page, perhaps the CONTROL FOOTING of a major control. With new Report Writer, this clause is necessary only with body groups. It has the useful property that, if there is a higher-level control break, the lower-level CONTROL FOOTING group does not affect the higher-level one, so that, if there is room, they normally remain together on the same page.

Write the clause in your 01-level entry for the group. The form NEXTGROUP + integer will create integer extra blank lines following the group, provided it is not the last on the page. The form NEXTGROUP NEXT PAGE causes your group to be the last on its page.

1.5.5 GROUP LIMIT Clause

You may not want some particular report groups to appear below a certain line on the page. For example, a CONTROL HEADING would seem out of place if it were last on the page. Just code GROUP LIMIT IS integer in the 01-level entry of your group. Integer will then be the bottom line number allowed for the last line of the group. See immediately below for an example.
1.5.6 **CONTROL HEADING at Top of Every Page**

Many report layouts have CONTROL HEADING groups that have to appear at the top of each page as well as after a control break. If this is required, just write the words OR PAGE after the control-name in the TYPE clause of your CH group. The following diagram shows this effect, and also illustrates the GROUP LIMIT clause that we discussed above (see 1.5.5 GROUP LIMIT Clause).

Because of the GROUP LIMIT, the CONTROL HEADING will not appear after line 57.

A CONTROL HEADING re-appears because of the new page even though no control break occurred.

---

**RD** CLUB-EXPENDITURE

**PAGE LIMIT 60 FIRST BODY GROUP 3 LINE LIMIT 132**

**CONTROL IS SPORT.**

```plaintext
01 TYPE CH FOR SPORT OR PAGE

GROUP LIMIT 58.

03 LINE + 2.

05 COL 1 VALUE "SPORT:".

05 COL + 2 PIC X(8) SOURCE SPORT.

05 COL + 2 VALUE "(CONT.)" ABSENT AFTER NEW SPORT.

03 LINE VALUE "=====".
```
1.5.7 **MULTIPLE PAGE Groups**

If you have a large vertical table to print, perhaps a summary with one line for each value encountered, you may be concerned that it will not always fit on one page. Perhaps there are usually less than 60 items but you have to allow for anything up to 1000 items! To handle this, code the clause **MULTIPLE PAGE** on your 01-level. Report writer will then automatically do a page advance whenever the page is full (printing PAGE FOOTING and PAGE HEADING as usual). Thus your code would be:

```cobol
01  SUMMARY-PAGES  TYPE DETAIL  MULTIPLE PAGE.
  03  LINE  OCCURS 0 TO 1000 DEPENDING ON NO-OF-ITEMS.
  ... etc.
```

This feature also handles more complex layouts, perhaps a multi-page personnel profile.

1.5.8 **Line WRAP**

You may sometimes define a number of relative COLUMN entries in one line and wonder whether they will all fit in the same line. If not, report writer will automatically wrap your data round onto a continuation line, but only if you code a **WRAP** clause. You can specify the last column before the wrap, the starting column for the continuation and the line advance required. As an example, you may have a series of possible error messages:

```cobol
03  LINE + 3  WITH WRAP AFTER COL 120 TO COL 82 STEP 2.
  05  COL 1       PIC X(80)    SOURCE INPUT-RECORD.
  05  COL + 2     "ACCOUNT NUMBER INVALID"  PRESENT WHEN ...
  05  COL + 2     "AMOUNT NOT NUMERIC"      PRESENT WHEN ...
  05  COL + 2     "DATES IN REVERSE ORDER"  PRESENT WHEN ...
  etc.
```

1.5.9 **FUNCTIONs**

The **FUNCTION** clause is used when you need to produce a specially formatted or converted report field that cannot be produced by **SOURCE**, **SUM**, or **VALUE**. Each FUNCTION corresponds to a pre-written routine that is either a built-in part of the report writer software or written by a person at your location. Examples of built-in FUNCTIONs are:

- **DATE** This outputs today’s date, or any given date, in the order: Day-Month-Year.
- **MDATE** This produces the same output as **DATE**, but in the order: Month-Day-Year.
- **TIME** This gives the current time.
Here is an example of how to use MDATE. Let's suppose that today is May 7th, 1999. Then if you write:

```
PIC 99/99/99 FUNCTION MDATE, you will obtain: 05/07/99
```

```
PIC <X(9)B<99,B9(4) FUNCTION MDATE, you will obtain: MAY 7, 1999
```

Information about developing your own functions will be found later (see 5.2 Developing User-Written Functions).

### 1.5.10 Special Print Attributes (Styles)

Nowadays all large system printers and smaller-scale printers and terminals have the ability to produce special effects which we hardly ever make any use of in COBOL applications. The **STYLE clause** enables you to make full use of any special effects that are available without affecting your program's portability. Supposing that you wish to highlight any "negative profits". Write:

```
05 COL 21 PIC -(8)9 SOURCE PROFIT
STYLE IS HIGHLIGHT WHEN PROFIT < 0.
```

You will now not need to change your program when moving between, say, a personal system, a mainframe with a laser printer, and a mainframe with an old impact printer, except possibly to change the **TYPE** clause in the **SELECT...ASSIGN** if it is not preset as the default. Also, the STYLE clause has no effect on the COLUMN clauses or any other part of your source program.

### 1.5.11 Independent Report File Handlers

Normally, your report's outputs are directed to a standard print file, as though you had written the program in elementary COBOL using **WRITE AFTER ADVANCING...** statements. An **Independent Report File Handler** is a pre-written routine to which all the output for a report file is directed. It may manipulate and output the data in any way the designer chooses. Your program can be made to invoke the file handler each time it has a line of report data, instead of implicitly executing a **WRITE...AFTER ADVANCING**. Each file handler is identified by a unique "mnemonic-name" of up to four characters. You cause your report program to use a file handler by coding your SELECT clause for the report file in the following way:

```
SELECT print-file ASSIGN TO assignment-name
    MODE IS mnemonic-name.
```

The file handler may require you to define a **CODE clause** in your RD statement. This clause is used to pass additional information to the file handler. Apart from this, no other change need be made to your program.
1.5.12 Multiple Reports

Your program may need to produce several different physical reports. Of course, you may define as many report files as you like, and each may be associated with as many Report Descriptions as you wish. But what if several of the reports have a similar appearance? You will not want to duplicate the code for all the Report Group Descriptions. Instead, you may define the report just once and effectively assign it to several files (although only one FD entry is coded). Just add the following clause to your SELECT clause:

```
Duplicated integer times
```

with the integer set to the maximum number of distinct reports you need.

The DUPLICATED clause causes the special register REPORT-NUMBER to be set up. You can MOVE any value into REPORT-NUMBER from 1 to your maximum number. This causes report writer to channel subsequent output to the corresponding report file. Each report is logically separate. Of course, the contents of each report are different because your program is writing to only one of the set at any given time. The layouts need not all be identical, since you are quite free to vary them conditionally in the usual way. (For example, REPORT-NUMBER could be used as a subscript or within the condition of a PRESENT WHEN clause.)

Only one FD entry is required for all the physical files associated with the multiple report. Similarly, only one OPEN and one CLOSE are required to open and close all its files. More details will be found later (see 5.1 Multiple Reports).

1.5.13 Using the Page Buffer

Some layouts are so irregular that you may wish that you could build up the page in any order like a news-sheet editor. The Page Buffer facility enables you to do this. Just add to your SELECT clause:

```
WITH PAGE BUFFER
```

Now you can tackle a layout such as the following:

```
NEW MEMBER DETAILS

NAME AND ADDRESS     SPORTS PLAYED

ANDREW ANALYST       TENNIS
21 MITCHAM ROAD      SQUASH
PUTNEY               SWIMMING
LONDON SW6

01 NAME-ADDRESS-GROUP TYPE DE.

01 SPORTS-GROUP TYPE DE.
...
```
The report groups in boxes have been defined separately. Normally you would not be able to place them alongside each other. (The REPEATED clause is not appropriate as NAME-ADDRESS-GROUP and SPORTS-GROUP are instances of different groups, not instances of the same group.) By using the Page Buffer you may now write in the PROCEDURE DIVISION of your program:

```
SET PAGE STATUS TO HOLD
GENERATE NAME-ADDRESS-GROUP
SET LINE TO FIRST DETAIL
SET PAGE STATUS TO RELEASE
GENERATE SPORTS-GROUP
```

You may store the groups on the page in any order. It is also possible to change the left/right positioning of groups by means of the SET COLUMN statement. There are several other variants of SET PAGE and SET LINE (see 4.4 Report Writer SET statements).

1.6 Further Study

The remainder of this volume cover the topics of this Tutorial in more detail. Since each part is organized in alphabetical sequence, it is not advisable to read them straight through, and the following order of topics is suggested for a first reading:

Part 2:
- Report Files, REPORT SECTION and RD:
  - PAGE LIMIT clause;
  - CONTROL clause.

  The rest of this part may be left to a second reading.

Part 3:
- Introducing Report Groups:
  - TYPE clause;
  - LINE clause;
  - COLUMN clause;
  - SOURCE clause, VALUE clause;
  - OCCURS clause, VARYING clause;
  - SUM clause;
  - PRESENT WHEN clause, PRESENT AFTER clause;
  - FUNCTION clause.

  The rest of this part may be left to a second reading.

Part 4:
- Report Writer Verbs: Overview:

  INITIATE statement, GENERATE statement, TERMINATE statement, excluding at first reading the “GENERATE Processing Cycle” and “TERMINATE Processing Cycle”.

  The rest of this part may be left to a second reading.

Part 5 may also be left to a second reading.
This part contains full information about the COBOL Report Writer elements that can appear in the ENVIRONMENT DIVISION and FILE SECTION of your program, and then, in alphabetic order, the clauses that may be used in an RD entry.

If you are migrating older programs written using OS/VS or DOS/VS COBOL's built-in Report Writer, you should refer to the Compatibility paragraph at the end of each section, which points out any new Report Writer features that these compilers do not accept.

Although most of the examples that follow use UPPER-CASE text, you may also use lower-case characters in any of the keywords and operands.
2.1 Report Files and RD: Keyword Table

The following table lists the major keywords relevant to COBOL Report Writer that may appear in the ENVIRONMENT DIVISION, FILE SECTION, and the RD entry, with a summary of their purposes. The third and fourth columns tell you whether or not the item is provided by IBM's OS/VS and DOS/VS COBOL and, if so, whether COBOL Report Writer extends the facilities.

If you wish to remain compatible with OS/VS or DOS/VS COBOL, you should avoid the new keywords and the extensions to the old ones. You will find additional information on this subject in the compatibility paragraph at the end of each section.
## Report Files and RD: Keyword Table

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Purpose</th>
<th>OS/VS DOS/VS COBOL?</th>
<th>Extensions to OS/VS and DOS/VS COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT</strong> (in ENVI-RONMENT DIVISION)</td>
<td>Associates file with external medium</td>
<td>yes</td>
<td>▪ MODE clause to direct output to Independent Report File Handler&lt;br&gt;▪ DUPLICATED clause for multiple report files&lt;br&gt;▪ WITH PAGE BUFFER for holding page contents before printing&lt;br&gt;▪ WITH RANDOM PAGE for writing to a cursor-controlled device&lt;br&gt;▪ FIRST PAGE NO ADVANCING to suppress initial page advance&lt;br&gt;▪ TYPE clause to select type of output device</td>
</tr>
<tr>
<td><strong>REPORT IS / REPORTS ARE</strong> (in FD)</td>
<td>Associates report with file</td>
<td>yes</td>
<td>▪ ALL phrase</td>
</tr>
<tr>
<td><strong>STYLE</strong> (in FD)</td>
<td>Invokes a special printer facility for the file or report</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>LINE LIMIT</strong></td>
<td>Gives maximum report line width</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>GLOBAL</strong></td>
<td>Makes report available to contained programs</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>PAGE LIMIT</strong></td>
<td>Allocates regions of page for different group TYPES</td>
<td>yes</td>
<td>▪ DE=DETAIL&lt;br&gt;▪ FIRST BODY GROUP=FIRST DETAIL&lt;br&gt;▪ LAST DE OR CH=LAST DETAIL&lt;br&gt;▪ LAST CF=LAST BODY GROUP=FOOTING&lt;br&gt;▪ phrases are now positionally independent sub-clauses&lt;br&gt;▪ LIMIT, LINE(S) not required&lt;br&gt;▪ FIRST DETAIL/LAST DETAIL not required even when PAGE HEADING/FOOTING coded</td>
</tr>
<tr>
<td><strong>CONTROL</strong></td>
<td>Specifies field(s) whose change of contents triggers a control break</td>
<td>yes</td>
<td>▪ REPORT=FINAL&lt;br&gt;▪ controls may overlap</td>
</tr>
</tbody>
</table>
| CODE | Attaches non-print data to report records | yes | - value may be of any length  
|-------|------------------------------------------|-----|----------------------|
| OVERFLOW | Specifies action to be taken when expression or SUM overflows | no | - CODE IS identifier format  
| ALLOW | Selects ANS-85 or ANS-68 rules for SUMming | no | - CODE IS literal format |

## 2.2 Report Files

COBOL Report Writer produces output records and writes them automatically to COBOL report files when your program executes a GENERATE statement (see 4.2) or TERMINATE statement (see 4.6). The report files are described very much like any other output sequential files. Each must have a SELECT...ASSIGN clause in the ENVIRONMENT DIVISION, and an FD in the FILE SECTION. They are accessed in the PROCEDURE DIVISION through the OPEN and CLOSE statements.

If the output is to be written to a special medium, or the program is to run in any special environment, or special treatment is to be given to the report data, the MODE clause is used. This directs report writer to use a specific file handler instead of writing output records in the standard way.

Your program may contain any number of report files and, if required, any number of other files. As usual, you may code your SELECT...ASSIGN clauses and your FD entries in any order.

### 2.2.1 a. SELECT...ASSIGN clause:

```markdown
Format

```SELECT``` file-name

```ASSIGN``` TO assignment-name

RESERVE integer-1

AREA AREAS

(Continued over)```
### Format (continued)

- **MODE IS** `BATCH` `mnemonic-name`
  - `-USING` parameter
- **DUPLECTED** `integer-2` `TIMES` `WITH PAGE BUFFER` `WITH RANDOM PAGE`
- **TYPE IS** `DEFERRED NONE`
  - `device-name` `DEFERRED`
- **FIRST PAGE NO ADVANCING** `other standard clauses`

### b. FD entry:

- **FD file-name** `EXTERNAL` `GLOBAL`
  - `IS` `REPORT IS report-name`
  - `REPORTS ARE all`
  - STYLE IS `NORMAL`
  - standard `BLOCK CONTAINS` clause
  - standard `RECORD CONTAINS` clause
  - RECORDING `MODE IS` `mode`
  - `other standard clauses`
  - record-description-entry
2.2.2 Select and FD: Coding Rules

1. You may code any other clauses after SELECT and any other clauses except LINAGE in the FD entry that may be appropriate for an output sequential file. In particular, a FILE STATUS clause may be used to return the status of your report file. The order of clauses is not significant.

2. Each report-name is a name of up to 30 characters, formed according to the usual rules for COBOL names. You might choose names that describes the output produced by the report, such as REPORTS ARE MONTHLY-SALES, END-OF-YEAR-TOTALS.

3. Each report-name must be the same as the report-name following an RD in your REPORT SECTION. A report-name may be DBCS. A report-name may appear only once in an FD entry. However, it may appear in more than one FD provided that any INITIATE for the report-name has the UPON file-name phrase (see 4.3 INITIATE statement) and provided that all the corresponding SELECT statements for these files differ only in their file-name, ASSIGN clause and MODE clause (if present).

4. If every report is to be written to the same file, you may write REPORTS ARE ALL. ALL must be the only operand and REPORTS ARE ALL must be the only REPORT(S) clause in the program.

5. A RECORD CONTAINS clause, or a BLOCK CONTAINS clause with the CHARACTERS option, is required if the identifier form of the CODE clause is used in any RD associated with the file. In all other cases, it is optional.

6. You should not normally specify a record-description-entry after the FD entry, because report writer relieves you of the need to code any WRITE statements for the report files. Your program may WRITE records to a report file independently of report writer, provided that there is no MODE clause in the corresponding SELECT and no CODE clause in the RD, and in this case you will of course need to specify at least one 01-level record description following the FD entry.

An explicit WRITE to the report file may be necessary in rare instances, such as when a downstream program will read your report file and it requires a header or trailer which must not have a carriage control character in its first byte. (Otherwise, a REPORT HEADING or REPORT FOOTING could be used for this purpose: even if there are non-DISPLAY fields to be written, they could be handled using a SOURCE referencing them as a large group field.) An Independent Report File Handler may also be used to manipulate the output for this purpose (see 5.3 Independent Report File Handlers).

7. If you do code a record description after the FD entry, and you wish to obtain fixed-length records, you should code a RECORD CONTAINS clause, even if you have also specified RECORDING MODE IS F. The integer of the RECORD CONTAINS clause should agree with the size of your record and must allow for the carriage control character if the NOADV option is in effect.

8. The MODE clause is used to indicate that each line of the report is to be passed to an Independent Report File Handler, instead of being written directly to a print file. The mnemonic-name consists of up to four alphanumeric characters. No check is made on the availability of the file handler until execution time. The
file handler may be either the basic file handler PRNT, a user-written, or a built-in file handler. File handlers extend the uses of report writer beyond output to "batch" files. They have two chief uses: (a) they allow the output to be sent to any kind of new physical device without changes to the program and (b) they allow a "back-end" software routine to perform any additional processing on the output. File handlers are described in a later section (see 5.3 Independent Report File Handlers) where the supplied file handlers are also listed.

9. The DUPLICATED, WITH RANDOM PAGE, and WITH PAGE BUFFER features cannot be implemented at run time by direct output and require a file handler to be present (although their processing is handled entirely by the run time system and not by the file handler itself). So if any of these clauses is present, but no MODE clause has been coded, the precompiler will assume an implicit MODE PRNT clause to be present. The same assumption is made if any report associated with the file has a CODE clause (except when all such reports have a CODE clause and they are all of the same length), see 2.5 CODE clause.

10. If the MODE clause is present, or is assumed implicitly for the reasons given in the preceding paragraph, the following restrictions apply:

a. No record descriptions may follow the FD entry for that file, as you cannot WRITE directly to a file that is processed by an Independent Report File Handler, see 5.3 Independent Report File Handlers.

b. The EXTERNAL and GLOBAL attributes have no effect. (Note that a report may still be GLOBAL, even though its corresponding file is not. Use of the MODL file handler also allows a report file to be treated as global (see 5.3.2 Supplied File Handlers).

c. The clauses RESERVE integer-1 AREA(S), PADDING CHARACTER, RECORD DELIMITER, and PASSWORD of the SELECT...ASSIGN clause and the clauses BLOCK CONTAINS integer RECORDS, LABEL RECORD(S) IS/ARE data-name, RECORD IS VARYING,..., CODE-SET, and VALUE OF... of the FD are not processed by the file handler and are treated as documentary only.

d. No USE AFTER STANDARD ERROR/EXCEPTION PROCEDURE Declarative section should be coded for the file.

e. The CANCEL and STOP RUN statements cannot be relied on to CLOSE files implicitly, as allowed under ANS 85 for regular files.

11. The device-name of the TYPE clause gives the make and model of the output device, or some other symbolic name. The TYPE clause enables the precompiler, or the run time system, to select the correct sequence of control characters to produce the desired special effect on the target device. (See 3.22 STYLE clause.) Apart from the reserved name NONE (meaning that no particular device is to be assumed) and TEST (a specially reserved name), there is a set of special character values associated with each device-name and each of the special effects available from the device. The physical values of these characters and the method by which they are inserted into the output is
highly machine- and device-dependent, but quite transparent to the program. Device-names are described in Installation and Operation.

DEFERRED means that any styles used are to be interpreted at run time, rather than stored explicitly when the program is precompiled. This enables the same program to operate with a number of different output devices without re-compilation.

If DEFERRED alone is given, the program will determine the target device at run time from the operating environment. It is then assumed that there may be an implicit style at the FD and the RD levels at run time (unless STYLE NONE is specified) and provision is made for them.

DEFERRED is also implied if PAGE BUFFER is coded, since the page buffer routine must know which characters are printable and which are control characters. DEFERRED may also be forced by any particular STYLE if the implementation decides that the routines required to effect it cannot be included at precompilation time.

If TYPE is not coded, the precompiler will assume a default device-name, chosen by the user at customization time, which may be absent, i.e. NONE.

If the device-name is NONE, any STYLE clause applying to this file, other than NORMAL, will be rejected, whether it is in the corresponding FD, any report assigned to the file, or in a report group description.

2.2.3 FILE-CONTROL and FD: Operation

1. If you specify more than one report-name in the same REPORTS ARE clause, you will be able to generate report data either consecutively or concurrently for the same file. (REPORTS and REPORTS ARE are interchangeable, however many report-names follow.) Tips on creating more than one report concurrently will be found later (see 5.1.4 Concurrent Reports).

2. The USING phrase indicates that the file handler is to be passed the parameters you specify in addition to the parameters normally passed automatically to the file handler on each call. The additional parameters will be first in the list of parameters passed. Only user-written file handlers may employ additional parameters, and their associated documentation should specify the exact number and size of the additional parameters required because, unless these are correct, unpredictable results may occur. Each parameter may be an identifier or literal or any other item that would normally be allowed in the USING phrase of a CALL statement, including their additional keywords such as LENGTH OF, ADDRESS OF, BY CONTENT, or BY REFERENCE.

3. The DUPLICATED clause indicates that integer-2 copies of the report writer Report Control Areas are to be created. For example, if you code DUPLICATED 4 TIMES, four copies of PAGE-COUNTER, LINE-COUNTER, control-break areas, total fields, and other internal registers or locations used by report writer will be set up under that report-name. Each copy controls a totally separate report.
passed to a different physical file (although only one FD entry is needed). This enables you to produce several separate reports that share an identical or similar layout without having to re-code several similar Report Descriptions.

4. The WITH PAGE BUFFER clause indicates that the Page Buffer facility is to be available to the file handler. This enables you to use the SET PAGE TO HOLD / RELEASE, SET LINE, and SET COLUMN statements (see 4.4 Report Writer SET statements) to build up your page in random fashion. A full explanation of Independent Report File Handlers and all related clauses is given in a later part (see 5.3 Independent Report File Handlers).

5. The WITH RANDOM PAGE clause indicates that the SET LINE and SET COLUMN statements may be used (see 4.4 Report Writer SET statements) to build up your page in random fashion. This clause is used when the output device is one which outputs data page by page rather than line by line (such as a visual display, or laser or page printer) and can change its “current position” to anywhere on the page. It is similar to WITH PAGE BUFFER, except that the buffer is in the device itself rather than in the program.

6. If you require normal output to a standard file, you may write MODE IS BATCH. This prevents any use of a file handler. MODE IS BATCH cannot be used if a DUPLICATED or WITH PAGE BUFFER clause has been coded.

7. If you do not code a RECORD CONTAINS clause or a BLOCK CONTAINS clause with the CHARACTERS option, report writer will calculate the logical record length for the report file from the longest actual line found in all the Report Descriptions associated with the file (rounded up to a multiple of 4). The length of the CODE field, where appropriate, and carriage control character are also added.

8. If you do code a RECORD CONTAINS clause (or, in its absence, a BLOCK CONTAINS clause with the CHARACTERS option), the integer specified will be used as the logical record length for the report file. The same integer, after subtracting the length of the carriage control character (if the NOADV option is in effect) and the CODE field, if appropriate, is also used to calculate a provisional value for the maximum line width for any report associated with the file (up to the default maximum established on customization), in case you omit the LINE LIMIT clause in an RD entry.

9. If you write RECORDING MODE IS V for a standard batch file (one not produced by a file handler), Report Writer will write variable-length records to your report file, truncating them, where possible, immediately after the last field in the line. This means that in most reports records will be considerably shorter, even after allowing for the additional record descriptor bytes that precede each record. If you also coded an optional RECORD CONTAINS clause with the format RECORD CONTAINS lower-integer TO higher-integer CHARACTERS, the lower-integer is used as a minimum length for all report records written to the file. Since QSAM normally requires at least four bytes per record (plus the carriage control byte), you should write RECORD CONTAINS 4 TO maximum-integer CHARACTERS or, if NOADV is in effect, then RECORD CONTAINS 5 TO maximum-integer CHARACTERS.

10. If you specify a RECORDING MODE clause for a file that uses an Independent Report File Handler, the recording mode you specify will be placed in report
writer's File Control Area for the report file, and the file handler may choose whether to act on it or to ignore it. The records passed to a file handler are always variable-length, irrespective of the RECORDING MODE. The file handler may process them in this form or output them as fixed-length records. The built-in PRNT file handler ignores the RECORDING MODE and uses the record format specified, or implied, at run time.

11. You may specify EXTERNAL or GLOBAL for a file that has a REPORT(S) clause. It is not necessary for a report file to be GLOBAL in order for it to have a GLOBAL report associated with it.

12. If you write FIRST PAGE NO ADVANCING, the usual form feed is not issued at the start of the first page after execution of the OPEN for the file. Instead, the program assumes that the paper is already positioned on line 1 of the page. This feature is useful for printing on pre-numbered forms when you do not want the first page to be wasted. (You may also eliminate all form feeds using the MODE NOPF (see 5.3.2 Supplied File Handlers).) FIRST PAGE WITH ADVANCING (the normal default) is provided for symmetry.

2.2.4 Compatibility

The MODE, DUPLICATED, WITH PAGE BUFFER, FIRST PAGE NO ADVANCING and STYLE clauses, and the concept of an Independent Report File Handler are unique to new Report Writer.
2.3 **REPORT SECTION** and RD

Each of your Report Descriptions is placed in the **REPORT SECTION**. The ANS standards, OS/VS COBOL, and DOS/VS COBOL all require this to be the last SECTION of the DATA DIVISION, positioned immediately before **PROCEDURE DIVISION**. However, the precompiler allows REPORT SECTION to appear **anywhere** in the DATA DIVISION after FILE SECTION.

Just as the FILE SECTION consists of a series of **FD** entries, each with record descriptions headed by an **01**-level entry to follow, so the REPORT SECTION consists of **RD** entries, each followed by **Report Group Descriptions** headed by an **01**-level entry.

Your program may produce **any number** of reports, each with its own Report Description. Each Report Description consists of an RD entry followed by **any number** of Report Group Descriptions (although there are restrictions according to **TYPE clause**; see 3.24). The order in which they are coded is immaterial. If several of the reports have strong similarities, you should define the report only once and make use of the **DUPLEXED** clause.

2.3.1

```
Format

REPORT SECTION. report-description-entry

where report-description-entry is defined as follows:

Format

RD report-name IS GLOBAL

ALLOW SOURCE SUM CORR clause

CODE clause

CONTROL clause

LINE LIMIT clause

PAGE LIMIT clause

(continued over)
```
2.3.2 REPORT SECTION and RD: Coding Rules

1. All other DATA DIVISION sections, if present, should precede REPORT SECTION.

2. Each report-name must be unique and must be the same as one of the report-names specified in the REPORT clause of one or more FD entries. (If a report-name appears in more than one FD, the UPON phrase of INITIATE is required, see 4.3 INITIATE statement.) This correspondence is used to determine where the output is to be written. Each report-name introduced in the FD must match a report-name of an RD.

3. The clauses of the RD may be written in any order; the order is not significant.

4. Each RD entry is followed by at least one Report Group Description. These define all the report groups (sets of one or more report lines) that may be produced in the report. They are fully described in the next part (see 3.24 TYPE clause).

5. If GLOBAL is specified, the report is available to any program contained in the current program, in the following senses:
   a. An INITIATE, GENERATE, or TERMINATE for the GLOBAL report-name may be issued from within a contained program, provided that the contained program itself does not have a locally-defined report of the same name. The contained program need not contain a REPORT SECTION.
   b. A GENERATE for any of the DETAIL report groups of the GLOBAL report may be issued from within a contained program, provided that the contained program itself does not have a locally-defined report and DETAIL group with the same names. If a GLOBAL report and a locally-defined report have different names but share DETAIL groups with the same name, these may be distinguished as usual by means of the IN/OF report-name qualifier.
   c. The special registers PAGE-COUNTER, LINE-COUNTER, LINE-LIMIT, and CODE-VALUE of the GLOBAL report, together with any sum-counters, may be accessed as GLOBAL items. Other report fields are not globally accessible.
The **CONTROL** and **SOURCE** fields, or any other data items used during the processing of the **GLOBAL** report, are those that are accessible to the containing - not the contained - program. Thus, **SOURCE** items must normally also be **GLOBAL** if you wish to set up values in them and print them from within a contained program.

If a contained program contains a **GLOBAL** report with an identical report-name, this will override the scope of original **GLOBAL** report until the end of the contained program. Similarly, a **DETAIL** group in a **GLOBAL** report of a contained program may override an identically-named one in the containing program.

The program in which the **GLOBAL** report is defined must receive control at least once before the report can be accessed, even though it need not itself contain any procedural statements referring to the **GLOBAL** report.

6. The **STYLE** clause causes one or more **styles** to take effect for the report as a whole. Usually this means that a certain control sequence will be sent to the printer just after the **INITIATE** is executed and another control sequence just before the **TERMINATE** is executed. If no **STYLE** clause is coded, an implicit style for the report may take effect, if this has been defined for the output device. **STYLE NORMAL** ensures that no style takes effect at the report level. Full details are given in the next part (see 3.22 **STYLE clause**).

The Formats and Rules for the other clauses in the RD are given in the sections that follow.

### 2.3.3 Compatibility

1. The following features are provided by new Report Writer only:
   - The **GLOBAL** phrase, and access to **GLOBAL** reports,
   - Use in ANS-85 contained or batched programs,
   - The **LINE LIMIT** clause,
   - The **ALLOW** clause,
   - The **OVERFLOW** and **SUM OVERFLOW** clauses,
   - The **STYLE** clause.

2. New Report Writer allows several FD’s to be associated with a given report-name, provided that the **UPON** phrase is used with an **INITIATE** for the report-name. It does not write to more than one file simultaneously. If you wish to continue to write to two files simultaneously, this may be achieved by means of a **file handler** that performs a **WRITE** to each file. See 5.1 **Multiple Reports**.

See the end of each section for compatibility notes on the other clauses.
2.4 ALLOW clause

This clause enables you to select whether the ANS-68 or the ANS-85 standard rules should be used for the formation of totals.

2.4.1 Format

```
| ALLOW | NO | SOURCE | SUM | CORR |
```

2.4.2 ALLOW Clause: Coding Rules

1. The COBOL Report Writer software, as supplied, assumes:

```
ALLOW SOURCE NO SUM CORR
```

because of the setting of the OSVS precompiler option. This default may be altered permanently by customization, or temporarily by changing the setting of the OSVS option in the JCL. You will need to code this clause only if you need to override the normal default, or if your program is to be portable and it is important to document which standard you are assuming.

2.4.3 ALLOW Clause: Operation

1. ALLOW SOURCE SUM CORR causes SOURCE SUM correlation to take effect throughout the report. The correlation between SOURCE items in a DETAIL and SUM clauses in a CONTROL FOOTING group is the main distinguishing feature of the ANS-68 standard. It is the only important case where the same code may give different results under the ’68 and the ’85 standards. OS/VS and DOS/VS COBOL’s built-in Report Writer uses the ANS-68 standard, and this is why the supplied version has the option OSVS set on, implying SOURCE SUM correlation. It will be important for you to understand the effect of SOURCE SUM correlation if your report has:

   - more than one DETAIL group, and
   - a SUM clause referring to a data item that is defined in a section of your DATA DIVISION (other than the REPORT SECTION).

   For full details, see 3.23 SUM clause.

2. If you code the word NO the effect is reversed.

2.4.4 Compatibility

The ALLOW clause is provided by new Report Writer only. OS/VS and DOS/VS COBOL always act as though ALLOW SOURCE SUM CORR were assumed.
2.5 **CODE clause**

This clause can be used to **prefix non-printable fields** to the report records. Such information is typically of use to de-spooling software and special device handlers.

### 2.5.1 Format

```
+-------------------+-------------------+
| CODE IS literal   |
| WITH CODE mnemonic-name |
| identifier        |
+-------------------+-------------------+
```

2.5.2 **CODE Clause: Coding Rules**

1. The forms `CODE IS` and `WITH CODE` are synonymous.

2. The `CODE` clause is **not** permitted if the associated FD entry is followed by a record description entry. This is because it would be illogical to WRITE independently to the file if there is also a CODE. See 2.2.2 **Select and FD: Coding Rules** and the rest of this section for further details.

3. The literal, if coded, must be a non-numeric literal.

4. If an identifier operand is used, it must represent a group field or a non-edited alphanumeric elementary field. The associated FD entry for the file must then have either a `BLOCK CONTAINS` clause with the `CHARACTERS` option or a `RECORD CONTAINS` clause, or both.

5. If a `mnemonic-name` operand is coded, there must be an entry in `SPECIAL-NAMES` of the form:

   ```
   literal IS mnemonic-name
   ```

   where `literal` is non-numeric. The value of `literal` is then used as the CODE value.

   If you require compatibility with ANS-68 report writer, **one** character is the norm. For ANS-85 compatibility, you should code a two-character literal.

6. If your report file description has a `RECORD` or `BLOCK CONTAINS` integer `CHARACTERS` clause and there is no `MODE` clause after `SELECT`, the size of the CODE, plus the maximum line width, must not be greater than the number of `CHARACTERS` specified.

7. Not all RD entries associated with the same report file need have a CODE clause and all the CODE clauses need not specify an operand of the same length. However, if the report file does not use a user-written file-handler that might ascribe some meaning to different lengths of CODE, it may be impossible to recognize the CODE at the front of each records and decide how long it is.
2.5.3 CODE Clause: Operation

1. The presence of a CODE clause implicitly establishes the special register CODE-VALUE in the Report Control Area. You may alter the contents of CODE-VALUE at any time. If there are several CODE clauses in different RDs within your program, you must qualify CODE-VALUE by the report-name.

2. If you do not specify a MODE clause after the SELECT for the corresponding report file, the value of the CODE is prefixed to every record written by report writer to the report file. The CODE is placed immediately before the carriage control character.

| CODE | ccc | print data ...

If the length of the CODE is not the same for every report being written to the file, or if some of the reports have no CODE, then a MODE PRNT clause is assumed in default, causing the built-in PRNT file handler to be invoked.

3. If you do specify a MODE clause after the SELECT, CODE-VALUE will be passed to the Independent Report File Handler in the Report Control Area. Built-in file-handlers (PRNT, MODL, NOPF) treat the CODE in the authodox way just described, but a user-written file handler may interpret CODE-VALUE in any desired way. Hence, your own user-written file handler may use the CODE clause for the passing of any additional information that is required by the file handler but does not necessarily appear in the report line itself. See 5.3 Independent Report File Handlers for more information.

4. If a literal or mnemonic-name is used, the size of CODE-VALUE is the length of literal, and CODE-VALUE is preset to the value of literal.

5. If an identifier is used, the length of CODE-VALUE is the (maximum) record length given in the RECORD CONTAINS clause, minus the LINE LIMIT. The current value of identifier is stored in CODE-VALUE at the start of the processing for each DETAIL or CONTROL HEADING for the report. This does not occur if the current group is a CONTROL FOOTING, so as to ensure that only pre-control-break values will be used. In the following example (assuming ADV is in effect) the size of CODE-VALUE is 140 minus 132 = 8 bytes, and the field WC-ACCOUNT-REF is moved to CODE-VALUE at the start of each non-CF body group:

```
FD REPORT-FILE
  RECORD CONTAINS 140 CHARACTERS
  REPORT IS MAIN-ACCOUNTS.
...
RD MAIN-ACCOUNTS
  LINE LIMIT IS 132
  CODE IS WS-ACCOUNT-REF ...
```

2.5.4 Compatibility

1. OS/VS and DOS/VS COBOL allow only the format: WITH CODE mnemonic-name. The corresponding literal defined in SPECIAL-NAMES must be one character.
2.6 **CONTROL clause**

This clause should be coded in your RD if your report has additional lines, such as total lines and subheadings, that are to be produced upon a change of value in one or more "key" fields (known as control fields or simply controls).

2.6.1 Format

```
CONTROL IS
  CONTROLS ARE
  REPORT
  FINAL
  control-id

control-id
```

2.6.2 **CONTROL Clause: Coding Rules**

1. As the format shows, you may code either the special keyword REPORT (or its equivalent, FINAL), or a list of identifiers (control-ids), or both. Commas are optional but helpful separators here, but you should code at least one space or new line between the operands. At least one operand must be coded.

2. REPORT, if present, must appear first in the list of control-ids. You may omit REPORT even if you refer to it in the Report Description. FINAL is an alternative name for REPORT.

3. Each control-id must be REPORT/FINAL or the name of an unedited data item in the DATA DIVISION of your program. It must not be a special register in the REPORT SECTION, such as PAGE-COUNTER. You may include qualifiers and subscripts if necessary. A PICTURE of a control-id should not have a "V" (implied decimal point) symbol.

4. You cannot use the same control-id more than once in the same CONTROL clause (unless a redefinition is used), but you can use the same control-id in different RDs.

5. If the OSVS option is in effect, control-ids may be required to be either group items or unedited alphanumeric or numeric DISPLAY items with a maximum size. Thus edited items and items with a USAGE of COMPUTATIONAL or INDEX are prohibited. Details will be found in Installation and Operation. If you would like to use such an item as a control without restrictions on its format, you can simply REDEFINE it or use a group level item containing only the item in question:

```
05 MONTH-X.
07 MONTH PIC 99 COMP.
05 next-item ...
RD ... CONTROL IS MONTH-X.
```
6. It is acceptable for your control fields to overlap. The following usage is therefore allowed:

```
03 ACCOUNT-CODE.
  05 BRANCH          PIC 99.
  05 FILLER          PIC XX.
  ... CONTROLS ARE BRANCH, ACCOUNT-CODE ...
```

7. Coding the CONTROL clause enables you to include some additional elements in your report description, namely:

- You can specify a CONTROL HEADING and/or a CONTROL FOOTING group for each control-id, if needed. (See 3.24 TYPE clause.)
- You can code PRESENT/ABSENT AFTER clauses (formerly known as GROUP INDICATE) with any of the control-ids as operand to cause report fields, lines etc. to appear or disappear after a change in its value. (See 3.17 PRESENT AFTER clause.)
- You can defer the resetting (zeroing) of a total field until after a change in a higher control. (See 3.23 SUM clause.)

### 2.6.3 CONTROL Clause: Operation

1. You code a CONTROL clause when your report has a structure based on changes in the value of one or more "key" or control fields, whose names you list in the CONTROL(S) clause. Report writer does not sort your data (to do that you could use COBOL SORT) but, assuming that your data is sequenced according to the control fields, report writer can perform certain actions automatically, such as the production of a CONTROL FOOTING and a CONTROL HEADING group when its contents change. It is also possible for a single CONTROL FOOTING group to be used for more than one level of control. (See 3.24 TYPE clause.)

In the following diagram, there are two levels of control. Two CONTROL HEADING groups and one CONTROL FOOTING have been coded. (There is no CONTROL FOOTING FOR YEAR.) The "boxes" around each of these groups shows their extent.

We have used some abbreviations in the following code - for example omitting the SOURCE and TYPE keywords - to save space. To shorten it further, you could abbreviate CONTROL HEADING FOR YEAR and CONTROL FOOTING FOR YEAR as CH YEAR and CF YEAR.
The preceding diagram illustrates the following important points:

a. Your CONTROL HEADING and CONTROL FOOTING groups are coded as separate 01-level report groups.

b. You can lay out CH and CF groups exactly as you like, just as you would for a DETAIL: report writer imposes no pre-defined format on any groups.

c. If you need a different CONTROL HEADING at more than one level (YEAR and MONTH in our example), you must code a new group for each level. This means that all groups may have different layouts. In this example, this applies also to the CONTROL FOOTING groups. (However, you can if you wish use the same group description for both levels by coding TYPE CF FOR YEAR, MONTH.)

d. Report writer produces your CONTROL HEADING group at the start of each new value of the control. Similarly, it produces your CONTROL FOOTING group at the end of each new value of the control.
CONTROL FOOTING groups are produced using the control values that existed before the control break. (See next item below for a fuller description of this.)

Both the CONTROL HEADING and the CONTROL FOOTING groups are optional for each control-id. You may code just a CONTROL HEADING group, or just a CONTROL FOOTING group, or neither.

2. The reserved word REPORT (or FINAL) is a special case representing the highest possible control. It is not a data-name. Include this as the first of your set of controls if you need special action to be taken once only at the beginning and end of the report; for example, if you require grand totals to be produced for the entire report.

3. Report writer keeps an internal copy of the pre-break contents of each control so that it may detect changes in the controls, known as control breaks. Ignoring the special case REPORT or FINAL for the moment, whenever your program issues a GENERATE statement, the CONTROL clause causes report writer to compare the contents of each control with its contents when the previous GENERATE was executed. The first control is examined first, then the second and so on. If no changes are found in any of the controls, no special action is taken. As soon as a change is found in a control, no further controls are examined. A break in a higher control always implies a break in all the lower controls, whether their contents have actually changed or not. (Obviously, 1991's JANUARY is a different month from 1992's JANUARY.) If you have more than one control, they must therefore have a hierarchy. Here are some examples:

<table>
<thead>
<tr>
<th>Structure of your data:</th>
<th>Format of the CONTROL clause:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONTH within YEAR</td>
<td>CONTROLS ARE YEAR, MONTH</td>
</tr>
<tr>
<td>CITY within COUNTY within STATE</td>
<td>CONTROLS ARE STATE, COUNTY, CITY</td>
</tr>
</tbody>
</table>

Your control-names must exist somewhere as data-names in the program outside the REPORT SECTION. For example, if in the last case the data-names in your file layout are actually written CUST-STATE, CUST-COUNTY, CUST-CITY, then your CONTROL clause would have to be written: CONTROLS ARE CUST-STATE, CUST-COUNTY, CUST-CITY.

4. When a control break is detected, if your report has CONTROL FOOTING groups, each control field is first saved in a temporary holding area and is then overwritten with the contents it had before the break for the duration of the production of the CONTROL FOOTINGs. This means that your program will use the before-the-break contents of any CONTROL field (for example in a SOURCE, or a PRESENT WHEN) in the following TYPES of group:

a. a CONTROL FOOTING;

b. a PAGE HEADING or PAGE FOOTING, when the page advance was caused by a CONTROL FOOTING;

and in the following situations:
c. when it is used as a SOURCE or SUM operand, either as it is or as a subscript or qualifier, or as part of an expression;
d. when it is used as part of a condition;
e. when it appears in a parameter to a FUNCTION;
f. when it is referenced implicitly, that is, via a redefinition, or via a group field or subordinate field or an intersecting field.

Only control fields exhibit their before-the-break values when referenced at CONTROL FOOTING time. To obtain the before-the-break value of a field other than a control field, you should use a Declarative procedure to save its current value (see 4.7.3 USE BEFORE REPORTING Directive: Operation).

After the lowest-level CONTROL FOOTING has been produced, and before any CONTROL HEADING or DETAIL groups are output, the current contents of all controls are restored. For a full description of the steps, see 4.2 GENERATE statement.

5. Report writer will not detect a control break until your program issues a GENERATE. If a control field in your input data changes several times but no GENERATE is issued during that time, no control breaks will be detected.

6. Your CONTROL identifiers need not be chosen just from ready-made locations in your input files or database. You may also "manufacture" them in WORKING-STORAGE. As a simple example, you may wish to print subtotals by quarter, although your main input gives just the months:

   RD ...  
   CONTROL IS W-QUARTER ...  
   ...  
   COMPUTE W-QUARTER = (F-MONTH-NO + 2) / 3

7. You cannot define more than one CONTROL HEADING or CONTROL FOOTING for a given control-id in your report. However, cases sometimes occur when you would like two CONTROL FOOTING report groups for the same level of control. You may achieve this referring to the same control field under a different name, as in the following case, where we must have two groups because the second part begins on a new page:

   05 W-ACCT-NO-1    PIC X(6).
   05 W-ACCT-NO-2    REDEFINES W-ACCT-NO-1    PIC X(6).
   * REPORT SECTION.
   RD ...  
   CONTROLS ARE W-ACCT-NO-1 W-ACCT-NO-2 ...  
   ...  
   01 GRP-A    TYPE CF FOR W-ACCT-NO-2.
   03 LINE + 3.   ...  
   ...  
   01 GRP-B    TYPE CF FOR W-ACCT-NO-1.
   03 LINE NEXT PAGE.   ...
Here, the two controls W-ACCT-NO-1 and W-ACCT-NO-2 are physically the same field. Consequently, there will be a break in the higher control whenever there is a break in the lower control, and the two CONTROL FOOTING groups, GRP-A and GRP-B, will always appear together in that order.

8. Report writer will consider a control break to have taken place if there is any change in the bit-pattern of the control field. For example, if the field is packed decimal (COMPUTATIONAL-3), a value of (hex) 123C and (hex) 123F will be considered different, even though they both represent the same numeric value. If this property is undesirable, your program should MOVE such a field to a DISPLAY field and use the DISPLAY field as the control field.

9. If your program has several Report Descriptions, each report is processed independently of the others. You can decide separately for each report whether it will have a CONTROL clause and which controls to specify. When you issue a GENERATE for a report that has controls, report writer examines the controls for that report only, ignoring all the others.


In some report structures, there may appear to be controls which are in parallel rather than hierarchical arrangement. For example:

```
Department #
  \
Student Id.  Teacher Id.
```

The organization of records in your file might be:

Department A:
- STUDENT record #1
- STUDENT record #2
- STUDENT record #3
  ... etc ...
- TEACHER record #1
- TEACHER record #2
- TEACHER record #3
  ... etc ...
Department B:
  ... etc ...

As you see, there is no hierarchical relationship between STUDENTS and TEACHERS. You might wish to print a CONTROL FOOTING group for the STUDENT records in each Department and a CONTROL FOOTING group of quite different appearance for the STUDENT records. To achieve this you must regard the two different CONTROL FOOTING groups as different versions of the same CONTROL FOOTING and use a **PRESENT WHEN clause** to distinguish them (see 3.18). Make the STUDENT-TEACHER indicator an extra lower control. Here is a skeleton solution:
11. There may be other purposes for specifying an item as a control. You might include it for the following reasons:

   a. To trigger a **PRESENT AFTER clause** (or a GROUP INDICATE clause), or **The RESET Phrase** of the SUM clause.

   b. To force a control break even though a lower control has not changed. You might want to output just *monthly* totals over several years' data. If you then declare MONTH as a control you must include YEAR too as a higher control, because it is quite possible for JANUARY 1991 to be followed immediately by JANUARY 1992 if you happen to have no data for FEBRUARY to DECEMBER 1991. (Note: in this example, you could also solve the problem by having just one control, YEAR-MONTH, if they are contiguous.)

   c. Because you may want to use the field as a SOURCE at CONTROL FOOTING time and you want to obtain the **previous** value of the field. (See item 4 above.) For example, you might have both CUSTOMER-NUMBER and CUSTOMER-NAME. By making CUSTOMER-NAME a control field following CUSTOMER-NUMBER (the "true" control field), you can be sure that you will see only the **pre-break** values of CUSTOMER-NAME at Control Footing time.

   d. For documentary purposes. The lowest-level controls need not be used at all in the program.

   e. If the program's SPECIAL-NAME paragraph contains an **ALPHABET** clause, you may need to use the **NOXCAL** option to ensure that the specified collating sequence is used. See **Installation and Operation**.

### 2.6.4 Compatibility

1. The use of REPORT as an alternative to FINAL is unique to new Report Writer.

2. Only new Report Writer allows control fields to **overlap**.

3. Only new Report Writer forbids the use of a COMPUTATIONAL item as a CONTROL field under certain circumstances. New Report Writer regards two instances of a COMP-3 control to be different if their sign is hex C in one case and hex F in the other, even though all the remaining digits may be equal.

4. Only new Report Writer checks that all the **control-ids** in a given CONTROL clause are different.
2.7 **LINE LIMIT clause**

This clause indicates the maximum number of columns likely to be required for the longest line of your report. It enables report writer to check that all of your report fields appear within the limits of the report line, and to warn you if there is any danger of data being lost beyond the right-hand extremity of the lines.

2.7.1 **Format**

```
| LINE LIMIT IS | integer | identifier |
```

2.7.2 **LINE LIMIT Clause: Coding Rules**

1. The value coded gives the maximum line width, in other words the greatest number of print columns required for your report. You may simply enter the column width of your printer, for example: `LINE LIMIT IS 132` or, if your report is clearly designed to take up less than this number of columns, use that value instead. Do not allow for the carriage control character.

2. The identifier form of the clause is used if you wish the width of your report line to assume different values at different times. This form of the clause takes effect only when you use either the **REPEATED clause** (see 3.19), or the **WRAP clause** (see 3.28). The identifier must be an unedited numeric field.

3. If the FD entry for the corresponding report file contains a **BLOCK** or **RECORD CONTAINS integer CHARACTERS** clause (other than **BLOCK CONTAINS 0 CHARACTERS**), the value of integer, after allowing for the carriage control character (if the **NOADV** option is in effect), and the size of any CODE field, must not be less than the LINE LIMIT, or its default value (see item 3 in the section below).

2.7.3 **LINE LIMIT Clause: Operation**

1. If any report field extends beyond the maximum line width given in your LINE LIMIT clause, report writer will signal a fault, either at compile time or, if that is not foreseeable, at run time.

2. If you use the identifier form of the clause, report writer evaluates its contents dynamically at **INITIATE** time and uses that as the value for the clause. For the purpose of checking the validity of COLUMN numbers, it will use the default maximum value described below. The value set up by the identifier is used at run time for the following purposes:

   a. to vary the number of **REPEATED** groups that may be placed side-by-side (see 3.19 **REPEATED clause**),

   b. as one means of adjusting the right margin when the **WRAP clause** is used to produce line wrap round (see 3.28 **WRAP clause**).
c. to check for (illegal) line overflow in variable-position report fields when the WRAP clause is not used.

3. If you omit the LINE LIMIT clause, report writer will assume a default value of the **maximum line width**. This is set to 256 in the report writer software as supplied but this default may be changed by customization to any lesser value (see Installation and Operation).

4. The LINE LIMIT need not be the same as logical record length of the report file. The latter is established from the computed maximum length of the lines of the report, or from the RECORD or BLOCK CONTAINS clauses if present (see 2.2.3 FILE-CONTROL and FD: Operation).

5. An internal special register with the reserved name LINE-LIMIT is established in the Report Control Area, containing the value specified in the LINE LIMIT clause, or its default value.

**2.7.4 Compatibility**

1. The LINE LIMIT clause is unique to new Report Writer. OS/VS and DOS/VS COBOL do not perform checks on the feasibility of COLUMN numbers.
2.8 OVERFLOW clauses

The OVERFLOW clause tells report writer how to protect your program from arithmetic errors, such as zero divide, that would cause an elementary COBOL program to fail. The SUM OVERFLOW clause tells report writer what action to take if a total field overflows - an event which is more likely to happen than with other report fields. It may be difficult to estimate the number of digits needed for totals, since this will depend on the number and content of items to be accumulated into the totals. Use these clauses if you need to change the standard action.

2.8.1 Format a

```
►► OVERFLOW PROCEDURE IS
  OMITTED
  STANDARD
  REPLACE BY literal-1
  STOP literal-2
```

2.8.2 Format b

```
►► SUM OVERFLOW PROCEDURE IS
  OMITTED
  STANDARD
  REPLACE BY literal-1
  STOP literal-2
```

2.8.2 OVERFLOW Clause: Coding Rules

1. The OVERFLOW (format a) and SUM OVERFLOW clauses (format b) are distinct clauses and you may choose a different option for each.

2. If your program contains no SUM clauses, the SUM OVERFLOW clause is not required. Similarly, if your program has no clauses of the form SOURCE arithmetic-expression, the OVERFLOW clause is not required. In either case, the clause may nevertheless be coded and it then has no effect.

3. Use the OMITTED option if your report uses arithmetic expressions or has a SUM clause respectively but there is no likelihood of a size error.

4. Use the REPLACE BY option if your report may be sensitive to improbable values in the user's data and you would like to show on the report exactly where errors have occurred. REPLACE BY can be followed by either a numeric or a non-numeric literal-1, whatever the PICTURE of your report field.

5. Use the STOP option only if SUM or arithmetic overflow is extremely unlikely but potentially damaging and you are content for your program to execute an "emergency" COBOL STOP in such a case.
2.8.3 OVERFLOW Clause: Operation

1. The OVERFLOW clause takes effect if your program contains clauses of the form SOURCE arithmetic-expression. On each occasion that the expression is evaluated a check may be made in case the result is too large for the report field. Also, if any expression involves a division step there could be a zero divide error, such as FIELD-A / FIELD-B when FIELD-B contains zero.

2. The SUM OVERFLOW clause takes effect if your program contains a SUM clause. On each addition, a check may be performed for size error. This clause does not affect any of the other functions of the SUM clause, such as the resetting (zeroing) of the totals. The default in each case is STANDARD (see below).

3. If you choose the OMITTED option the effect is as follows:

   OVERFLOW: Report writer will not perform any checks for arithmetic overflow. This will save a small overhead on the evaluation of expressions. If a size error occurs, then at best your report field will have some high-order digits truncated. If a zero divide error occurs, your program will fail at run time.

   SUM OVERFLOW: Report writer will not perform any checks for SUM overflow. This will save a small overhead on totalling. If a size error occurs, at least one top digit will be truncated and lost from the total field.

4. If you choose the STANDARD option, the effect is as follows:

   OVERFLOW: Report writer will always check for size error, or zero divide, when it evaluates each SOURCE expression. If this happens, your report field will be blank and a run time error 10 will be indicated.

   SUM OVERFLOW: Report writer will check for size error on each addition into a total field. If this happens, a run time error 11 will be indicated. No adding will take place into your total field and you will obtain the total up to the point just after the last valid addition.

5. If you code the REPLACE BY option, the effect is as follows:

   OVERFLOW: Report writer will check for a size error or zero divide and, if this occurs, it will place your specified literal-1 in the SOURCE report field instead of the erroneous value.

   SUM OVERFLOW: Report writer will check for a size error and, if this occurs, it will place your specified literal-1 in the SUM report field instead of the erroneous value.

   In either case, if you choose a numeric literal-1, this value will be stored according to the rules of the MOVE statement. If you choose a non-numeric literal-1, the literal will be stored, as for a MOVE, in your report field, treated as though it were redefined as an unedited alphanumeric field (PIC X...). For example, if overflow occurs and your report field is defined as:

   05 COL 20 PIC ZZZ9.99 SOURCE NUM-ORDERED * UNIT-PRICE.
and your RD contains the clause: **OVERFLOW PROCEDURE IS REPLACE BY ZERO**, the following will appear:

```
0.00
```

and if your RD contains the clause: **OVERFLOW PROCEDURE IS REPLACE BY ALL “?”**, the following will appear:

```
???????
```

6. If you code the **STOP** option, report writer will execute a COBOL **STOP literal-2** as soon as an error is detected.

**2.8.4 Compatibility**

The OVERFLOW and SUM OVERFLOW clauses are unique to new Report Writer. The **SUM OVERFLOW IS OMITTED** option emulates the effect of OS/VS and DOS/VS COBOL’s built-in Report Writer.
2.9 PAGE LIMIT clause

The PAGE LIMIT clause should be coded if your report is to be divided into pages. It also allows you to sub-divide the page into regions for the headings, main data, and footings.

2.9.1 Format

```
HEADING IS integer-1
FIRST DETAIL IS integer-2
BODY GROUP
LAST DETAIL OR CONTROL HEADING
FOOTING
PAGE LIMIT IS integer-6
LIMITS ARE LINES
```

2.9.2 PAGE LIMIT Clause: Coding Rules

1. The format above gives you choices of keywords, and in each case the different keywords have the same meaning. Traditionally, the sub-clauses were referred to as the HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING phrases. We also usually refer to the whole clause as the PAGE LIMIT clause, even though the word LIMIT is optional.

2. Each of the sub-clauses is optional, but none of the first four sub-clauses may be present without the PAGE LIMIT sub-clause. You may code the sub-clauses in any order, and they may appear anywhere in the RD statement. The order shown follows a natural progression from the top to the bottom of the page and is therefore recommended for maximum lucidity.

3. The values of integer-1, integer-2, integer-3, and integer-4 (if you code their sub-clauses), and integer-6 represent the start and finish points of various regions of your page, working down from the top to the bottom. Ensure that the regions start and finish in the order shown in the diagram below (see 2.9.3). Any two integers may be equal. All integers must lie between 1 and 9999 inclusive.
4. If you use the identifier form of the LAST DETAIL sub-clause, the identifier used must be an unedited numeric field and its value at every generation of your report must lie between the FIRST DETAIL and LAST CONTROL FOOTING positions, inclusive.

5. By using the +integer-5 form of the LAST CONTROL FOOTING sub-clause, you specify the extra lines to be made available to CONTROL FOOTING groups. Ensure that you cannot exceed the PAGE LIMIT; that is, the LAST DETAIL value (identifier-1 or integer-3) + the LAST CF offset (integer-5) must be not greater than the PAGE LIMIT (integer-6).

6. The FIRST four PAGE LIMIT sub-clauses may all be omitted. Here are some guidelines on their use:
   
a. **HEADING** is never required. However, if your report has a PAGE HEADING that begins with a relative LINE, you may use HEADING as an anchor point for the start of that group.

b. **FIRST DETAIL** should be coded if you have a PAGE HEADING group, especially one that might vary in depth, and you want the body of the page to follow at a fixed position underneath it.

c. **LAST DETAIL** should be coded if you have a PAGE FOOTING group and want the body of the page to end short of the line preceding it or if you want to use LAST DETAIL in conjunction with LAST CF as described in the next paragraph. Use the LAST DETAIL identifier form if you want to vary the logical page depth dynamically.

d. **LAST CF** (or LAST CONTROL FOOTING, or FOOTING) should be coded if you have CONTROL FOOTING groups and want to leave some space before the PAGE FOOTING begins. (If the OSVS option is not in effect, this may be provided automatically - see item 7 in the next section below.)

7. If you omit the PAGE LIMIT clause, your report will consist of one continuous stream of output without page breaks. Your Report Group Descriptions will then not be able to contain absolute LINE (see 3.10 LINE clause), or NEXT GROUP (see 3.13 NEXT GROUP clause), PAGE HEADING and FOOTING groups, or any form of any clause that makes use of the keyword PAGE.
2.9.3 PAGE LIMIT Clause: Operation

1. The PAGE LIMIT clause enables report writer to assign regions to your page. The following diagram shows how the various regions are mapped onto your page:

   Regions of the Page

   +-------------------------------------------------------------+
   | ///////////////////////////////////////////////////////////// |
   | PAGE                                                      |
   | HEADING                                                   |
   | DETAIL and CONTROL HEADING groups                         |
   |     ▼ CONTROL FOOTING groups                              |
   |     ▼ body of page                                       |
   |     ▼ ▼ ▼                                               |
   |     ▼ ▼ ▼                                               |
   |     ▼ ▼ ▼                                               |
   |     ▼ ▼ ▼                                               |
   |     ▼ ▼ ▼                                               |
   | PAGE                                                      |
   | FOOTING                                                   |
   | ///////////////////////////////////////////////////////////// |

   Each of your groups will be checked to fit into its appropriate region. (The REPORT HEADING and REPORT FOOTING groups are special cases.) The fitting of groups on the page is described in detail in the next part (see 3.10 LINE clause).

2. If you code a HEADING sub-clause, its value will be used in the case where you have a PAGE HEADING or a REPORT HEADING group whose first LINE clause is relative. Those groups will then be positioned relative to the value of HEADING minus 1. (Compare the different rule for the positioning of a relative first LINE clause in a body group at FIRST DETAIL, see 3.10 LINE clause.)

3. The region between FIRST DETAIL and LAST CONTROL FOOTING inclusive is the body of the page. Apart from the optional REPORT HEADING and REPORT FOOTING groups, which may appear anywhere on the page, only body groups (CONTROL HEADING, DETAIL, and CONTROL FOOTING) will appear in this region. However, if a PAGE HEADING group encroaches into the FIRST DETAIL position, a diagnostic message (096) will be issued and the first body group will appear immediately after the PAGE HEADING group (as though the FIRST DETAIL were absent - see below).
4. CONTROL HEADING and DETAIL groups are not allowed to appear below the LAST DETAIL position. If LAST DETAIL is above LAST CONTROL FOOTING, your CONTROL FOOTING groups will thereby have extra space available to them. This extra space reduces the likelihood of the displeasing effect that results when they are forced to the top of a page. (See 3.10 LINE clause for more details.) You can imply this spacing by making LAST DETAIL fall short of PAGE LIMIT (or short of the line before the PAGE FOOTING if you have one). An alternative way to indicate this extra space is to code: LAST CONTROL FOOTING IS +integer-5.

5. If you code the identifier form of the LAST DETAIL sub-clause, report writer will take the contents of the identifier at the start of each GENERATE and use that as the value for the sub-clause.

6. If you use the relative form of the LAST CONTROL FOOTING sub-clause (with +), the number of lines you specify will be added to the LAST DETAIL value to give the LAST CONTROL FOOTING value. For example: LAST CONTROL FOOTING +3 specifies that 3 extra lines are to be available during the page fit for CONTROL FOOTING groups, regardless of any variations in an identifier operand of LAST DETAIL.

7. If you omit any of the first four optional sub-clauses, and report writer needs their values, it will infer default values according to the following rules:

no HEADING: = 1; that is, the top of the logical page.

no FIRST DETAIL:
   a. If there is no PAGE HEADING, the value of HEADING is the default; hence, if HEADING is allowed to default to 1, the body of the report will begin at the top of the page.
   b. If there is a PAGE HEADING, then the line immediately following the PAGE HEADING group, that is, the body of the report, will start immediately after the PAGE HEADING. If your PAGE HEADING varies in size, you may deliberately omit FIRST DETAIL, and the body of your page will adjust itself to take up all the space available.

no LAST DETAIL:
   a. If there is a LAST CONTROL FOOTING (other than the + form), LAST DETAIL takes the same value, that is, all your body groups will be allowed to come down to the LAST CONTROL FOOTING position.
   b. If there is a LAST CONTROL FOOTING with the + integer-5 form, then (first line of the PAGE FOOTING) - 1 - integer-5 is used; that is, report writer will use as much of the page as possible, allowing for any PAGE FOOTING, and leave integer-5 extra lines for the CONTROL FOOTING group(s).
   c. If there is no LAST CONTROL FOOTING, then the same default is used as for LAST CONTROL FOOTING (see next).
no LASTCONTROL FOOTING:

a. If the OSVS option is in effect and there is a LAST DETAIL, this value is also used for the LAST CONTROL FOOTING. In all other cases:

b. If there is no PAGE FOOTING, then the PAGE LIMIT is used; that is, your CONTROL FOOTING groups will be allowed to come down to the bottom of the page. (If you use the identifier form of LAST DETAIL, there is an exception to this rule: LAST CONTROL FOOTING will be the same as LAST DETAIL.)

c. If there is a PAGE FOOTING group, then the last line before the PAGE FOOTING is used. (If the PAGE FOOTING is a relative group, this is calculated by placing the last line of the PAGE FOOTING at the PAGE LIMIT.) Note that this default action differs from ANS COBOL, where LAST CONTROL FOOTING defaults to LAST DETAIL, if you specify it. This extension has the advantage that the LAST CONTROL FOOTING sub-clause can be omitted in most cases without misalignment of CONTROL FOOTING groups or wastage of space at the bottom of the page.

8. The following examples show some possible forms of this clause:

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. PAGE LIMIT 60.</td>
<td>This form is <strong>valid for all situations</strong> because of the defaults. All your body groups will fit between your PAGE HEADING (or line 1 in its absence) and your PAGE FOOTING (or line 60 in its absence).</td>
</tr>
<tr>
<td>b. FIRST DE 5 LAST CF +3 PAGE LIMIT 60.</td>
<td>This reserves lines 1-4 for the PAGE HEADING and, if your report has no PAGE HEADING, they will be left blank. Your CONTROL FOOTING groups may come down to the line before any PAGE FOOTING, or line 60 if there is no PAGE FOOTING, but CONTROL HEADING and DETAIL groups must end 3 lines before that point.</td>
</tr>
<tr>
<td>c. LAST DETAIL WS-PAGE-SIZE PAGE LIMIT 60.</td>
<td>Here the value of WS-PAGE-SIZE will be used as the lower limit for <strong>all</strong> body groups (since the identifier form of LAST DETAIL causes LAST CONTROL FOOTING to default to LAST DETAIL instead of PAGE LIMIT).</td>
</tr>
</tbody>
</table>
2.9.4 Compatibility

1. The alternative spellings DE for DETAIL, FIRST BODY GROUP for FIRST DETAIL, LAST DETAIL OR CONTROL HEADING / CH, LAST CONTROL FOOTING / CF, and LAST BODY GROUP are unique to new Report Writer.

2. The concept that each keyword may introduce a clause in its own right is unique to new Report Writer. OS/VS and DOS/VS COBOL require the keyword PAGE to appear first and do not allow different phrases of the PAGE LIMIT clause to be separated by another clause.

3. OS/VS and DOS/VS COBOL require the keyword LINE or LINES.

4. The defaults assumed by OS/VS and DOS/VS COBOL are not sufficient to allow omission of the clauses in most cases as they are with new Report Writer. Where new Report Writer's defaults are different from those of OS/VS COBOL and DOS/VS, no undetectable incompatibility will result, because in these cases the different defaults assumed by OS/VS and DOS/VS COBOL cause compilation errors.
This part describes in detail every aspect of the Report Group Descriptions that follow your RD entry in the REPORT SECTION. After the next section, the sections are in alphabetical order for easy reference.

If you are migrating older programs written using OS/VS or DOS/VS COBOL’s built-in Report Writer, you should refer to the Compatibility paragraph at the end of each section, which points out any new Report Writer features that these compilers do not accept.
3.1 Introducing Report Groups

3.1.1 What is a Report Group?

A report group is an uninterrupted block of report lines, from zero up to any number. With the exception of MULTIPLE PAGE groups (see 3.12 MULTIPLE PAGE clause), the lines in a report group are always together on the same page and are generated in a single operation. Of the seven TYPES, all but the DETAIL groups are produced automatically. Your program issues at least one GENERATE statement for each DETAIL group (or for the report as a whole), and any other groups that you have defined are automatically generated in the correct positions relative to the DETAILS, depending on their TYPE.

Your RD entry may be followed by any number of Report Group Descriptions, but there is a limit to the number each TYPE other than DETAIL. Each Report Group Description begins with a 01 level-number entry in the A-margin.

More information about report groups follows or may be found later in this part (see 3.24 TYPE clause).

3.1.2 Report Groups: Keyword Table

The following table lists the major report writer keywords that may appear in a Report Group Description, with a summary of their purposes. The third and fourth columns tell you whether or not the item is provided by IBM's OS/VS and DOS/VS COBOL and, if so, whether COBOL Report Writer extends the facilities. The clauses may be found, listed in alphabetical order by keyword, following this section.

If you wish to remain compatible with OS/VS or DOS/VS COBOL, you should avoid the new keywords and the extensions to the old ones, possibly by using the option described in Installation and Operation to restrict your use of extended features. You will find additional information on this subject in the Compatibility paragraph at the end of each section.
## Report Groups: Keyword Table

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Purpose</th>
<th>OS/VS DOS/VS COBOL?</th>
<th>Extensions to OS/VS and DOS/VS COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TYPE</strong></td>
<td>Indicates whether group is produced automatically, (PH,PF,CH,CF,RH, RF) or GENERATED explicitly (DE)</td>
<td>yes</td>
<td>▪ TYPE keyword optional&lt;br&gt;▪ optional FOR/ON with CH/CF&lt;br&gt;▪ CH FOR control OR PAGE form&lt;br&gt;▪ no TYPE = TYPE DETAIL&lt;br&gt;▪ multiple CONTROL FOOTING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NEXT GROUP</strong></td>
<td>Provides extra vertical space between groups</td>
<td>yes</td>
<td>▪ PLUS can be written +&lt;br&gt;▪ optional words BODY and DE OR CH&lt;br&gt;▪ optional ON before NEXT PAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GROUP LIMIT</strong></td>
<td>Gives lowest permissible position for body group</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>MULTIPLE PAGE</strong></td>
<td>Allows a group to span several pages</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>REPEATED</strong></td>
<td>Repeats body groups side-by-side across page</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>Specifies vertical position</td>
<td>yes</td>
<td>▪ PLUS can be written +&lt;br&gt;▪ multiple form&lt;br&gt;▪ LINE alone = LINE PLUS 1&lt;br&gt;▪ LINE PLUS ZERO = LINE PLUS 0&lt;br&gt;▪ LINE without subordinate COLUMNS gives blank line&lt;br&gt;▪ absolute may follow relative if group starts with absolute&lt;br&gt;▪ RH/RF or body groups may occupy several pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OCCURS</strong></td>
<td>Indicates repeating item</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>VARYING</strong></td>
<td>Defines internal counter for use as subscript, etc.</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>PRESENT/ABSENT WHEN</strong></td>
<td>Gives condition for printing or skipping item</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Keyword</td>
<td>Purpose</td>
<td>OS/VS DOS/VS COBOL?</td>
<td>Extensions to OS/VS and DOS/VS COBOL</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>PRESENT/ABSENT AFTER</td>
<td>Specifies internal condition for printing/skipping</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>GROUP INDICATE</td>
<td>Simple form of PRESENT AFTER</td>
<td>yes</td>
<td>° may be used at group level</td>
</tr>
<tr>
<td>BLANK WHEN ZERO</td>
<td>Causes zero value to be spaces</td>
<td>yes</td>
<td>° allowed at group level</td>
</tr>
<tr>
<td>JUSTIFIED</td>
<td>Changes alignment rules for alpha-numeric fields</td>
<td>yes</td>
<td>° allowed at group level</td>
</tr>
<tr>
<td>SIGN</td>
<td>Changes output convention for &quot;S&quot; PICTURE symbol</td>
<td>no</td>
<td>° LEADING literal TRAILING literal format for user-specified &quot;signs&quot;</td>
</tr>
<tr>
<td>WRAP</td>
<td>Allows data to &quot;wrap round&quot; onto continuation lines</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>STYLE</td>
<td>Invokes a special printer property</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>USAGE</td>
<td>Documentary</td>
<td>yes</td>
<td>° DISPLAY-1 for DBCS items</td>
</tr>
<tr>
<td>COLUMN</td>
<td>Specifies horizontal position</td>
<td>yes</td>
<td>° may be shortened to COL</td>
</tr>
<tr>
<td>PICTURE</td>
<td>Gives format in which field is to be presented</td>
<td>yes</td>
<td>° relative form (+ or PLUS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° CENTER and RIGHT options</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° multiple format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° allowed alone as dummy entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° left-shift symbols &lt;...&gt; for variable left alignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° optional when VALUE &quot;literal&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° general insertion characters</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Specifies field whose contents are to appear in report item</td>
<td>yes</td>
<td>° SOURCE keyword optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° arithmetic-expression format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° ROUNDED phrase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° SUM or COUNT term may be used as operand</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° multiple format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>° multiple-choice format</td>
</tr>
<tr>
<td>Keyword</td>
<td>Purpose</td>
<td>OS/VS DOS/VS COBOL?</td>
<td>Extensions to OS/VS and DOS/VS COBOL</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>VALUE</td>
<td>Specifies fixed value for report item</td>
<td>yes</td>
<td>□ VALUE keyword optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ multiple format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ assumes default PICTURE</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>Specifies run time routine to provide contents of item</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>Indicates totaling of specified item(s)</td>
<td>yes</td>
<td>□ optional word OF after SUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ arithmetic-expression format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ may be used as term in SOURCE expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ allowed in non-CF groups</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ may refer to SOURCE/VALUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ ROUNDED phrase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ ANS-74/85 method (no SOURCE SUM correlation) available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ automatic check for overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ &gt; 1 SUM clause in entry ok</td>
</tr>
<tr>
<td>COUNT</td>
<td>Counts appearances of item(s)</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>PAGE-/LINE-COUNTER</td>
<td>Special registers for page number &amp; vertical position</td>
<td>yes</td>
<td>□ need not be qualified in REPORT SECTION/DECLARATIVES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>□ adding to LINE-COUNTER creates gap on page</td>
</tr>
<tr>
<td>COLUMN-COUNTER</td>
<td>Special register for horizontal position</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
3.2 Coding Report Group Descriptions

A Report Group Description is a REPORT SECTION data structure beginning with an 01-level entry and including any number of lower entries. It may consist of the 01-level report group entry only. You may code any number of Report Group Descriptions after your RD entry.

3.2.1

where report-group-entry is defined as follows:

- level-number
- data-name
- TYPE clause

- NEXT GROUP clause
- GROUP LIMIT clause
- REPEATED clause

- LINE clause
- COLUMN clause
- PICTURE clause

- SOURCE clause
- VALUE clause
- ROUNDED

- SUM clause
- COUNT clause
- RESET phrase
- FUNCTION clause

- PRESENT/ABSENT WHEN clause
- PRESENT/ABSENT AFTER clause
- GROUP INDICATE clause
- multiple-choice (see below)

- WITH
- WRAP clause
- MULTIPLE PAGE clause

- OCCURS clause
- VARYING clause
- SIGN clause

(Continued over)
3.2.2 Report Groups: Coding Rules

1. Level-numbers are used, as in any DATA DIVISION record, to establish a hierarchy of group levels above the elementary level. The level-number at the start of each report group must be 01. (It may be written without the leading 0, but we shall still refer to it as the 01-level.) Choose your lower level-numbers exactly as you would for any other COBOL logical record. (Note that OS/VS and DOS/VS COBOL Report Writer uses the TYPE, LINE, and COLUMN clauses themselves, rather than the level-numbers, to establish hierarchy.)

2. You need not place data-names on your report group entries except in the following cases, where they are required:
   a. On the 01-level entry of a DETAIL group, or any other group that may be referred to in the USE BEFORE REPORTING header of a Declarative SECTION (see 4.7 USE BEFORE REPORTING directive). When a data-name is used at the 01-level, it gives a name to the entire report group and is called the report-group-name or group-name. The group-name must be unique within an RD, but you may re-use the name in another RD.
   b. On a numeric entry referred to in a SUM clause or term. (See 3.23 SUM clause.)
   c. On any entry at any level that is referred to in a COUNT clause or term. (See 3.6 COUNT clause.)

In all other cases it is preferable to omit data-names since an unnecessary data-name may mislead the program-reader into thinking that it is referred to in one of the ways above. It may also impede the precompiler's optimization.

Data-names may be DBCS.

3. The following general-purpose DATA DIVISION clauses can be used in a report group. The first four may be used at both group and elementary levels (not the
01-level in the case of OCCURS). **PICTURE** and **VALUE** are allowed only at the elementary level.

- **BLANK WHEN ZERO**
- **JUSTIFIED**
- **OCCURS**
- **SIGN**
- **PIC/PICTURE**
- **VALUE/VALUES**

but the following **cannot** be used in a report group:

- any **USAGE** other than **DISPLAY**, such as **COMPUTATIONAL**
- **REDEFINES**
- **SYNCHRONIZED/SYNC**
- **RENAMES**
- **88-level** entries

In addition to the general DATA DIVISION clauses, report groups may contain special-purpose clauses that describe the position and contents of fields. The first four clauses, **TYPE**, **NEXT GROUP**, **GROUP LIMIT**, and **REPEATED**, can appear only at the 01-level. The special-purpose clauses are:

- **TYPE** indicates the type of group
- **NEXT GROUP** creates extra space between groups
- **GROUP LIMIT** gives a special lower limit for the group
- **REPEATED** places groups side-by-side
- **LINE** gives the vertical position
- **MULTIPLE PAGE** enables a report group to span several pages
- **COLUMN** gives the horizontal position
- **SOURCE** used to capture a data field by name
- **SUM** used to produce a total
- **COUNT** used to produce a count
- **FUNCTION** invokes a special formatting routine
- **WRAP** allows data to continue on a new line
- **VARYING** provides counters for a repeating entry
- **STYLE** produces special printer effects
- **PRESENT/ABSENT WHEN/AFTER** controls whether or not an item, line or group is output
- **GROUP INDICATE** an older form of **PRESENT AFTER**

4. The clauses **PICTURE, COLUMN, SOURCE, VALUE, SUM, COUNT, and FUNCTION** can appear only at the elementary level.

5. For every entry with a **COLUMN** clause, there must be a **LINE** clause either at the same level as or at a higher level. If a given report line contains only one elementary field (and provided the entry is not a multiple-choice entry, see 3.18 **PRESENT WHEN clause**), you may combine the **LINE** and the **COLUMN** clauses in the same entry, for instance:
There is no limit to the number of elementary fields a report line may contain. If the report line contains several elementary fields, you must code all the COLUMN entries within the line at a lower level than the LINE entry, for example:

```
03 LINE 5.
  05 COL 60         VALUE "QUARTERLY REPORT".
  05 COL 90         VALUE "1997".
```

Not all the COLUMN clauses within a LINE need be at the same level, since some or all elementary entries might be contained within non-LINE group entries; for example:

```
03 LINE 5.
  05 COL 60         VALUE "QUARTERLY REPORT".
  05 PRESENT WHEN YEAR NOT = SPACES.
    07 COL 90     VALUE "YEAR:".
    07 COL +2     PIC X(4)     SOURCE YEAR.
```

6. A LINE clause must not be subordinate to another LINE clause. (If this rule is violated, the nested LINE entries will be treated as though they were defined at the same level as the first.)

7. If a given report group contains several lines, you must give all the LINE entries in that group a level-number other than 01. If the group contains only a single report line, you may code the LINE clause in the 01-level entry; for instance:

```
01 ORDER-LINE   TYPE DE   LINE + 1.
  05 ...
```

8. You may choose only one of five clauses: SOURCE VALUE SUM COUNT and FUNCTION to supply the contents of an elementary item automatically, bearing in mind that (a) SOURCE or VALUE can be repeated several times in a multiple-choice entry (see 3.18.5 The Multiple-Choice Form), and (b) SUM (see 3.23 SUM clause) or COUNT (see 3.6 COUNT clause) can be used as special operators in terms in an arithmetic expression (see 3.21 SOURCE clause).

9. If there is no SOURCE, VALUE, SUM/COUNT, or FUNCTION clause in an elementary item, the following conditions must be met:

   a. The entry must have a data-name (following the level-number).

   b. The entry must have a COLUMN clause.

   c. If the COLUMN clause is relative, the item must have a fixed horizontal position (that is, it must follow an item with a fixed end-column, unless it is first in the line).

   d. The item must have a PICTURE with none of the features that are unique to the REPORT SECTION (that is, "<" or ">" symbols or general insertion characters). Report writer will then expect you to fill in the report field independently with some COBOL procedural statement.
As an example, in the following case

```
05 R-PRIOR-CUST-NO COL 20 PIC 9(7).
```

the only way this item can receive a value is through a COBOL MOVE, ADD, ACCEPT, or other procedural statement, which is outside the scope of report writer and is left to you. Report writer will ensure that the report field is not overwritten by any other field, so any value stored in the item R-PRIOR-CUST-NO will remain there and will appear whenever the enclosing group is produced.

Note that you should not attempt to change the value of a group field, such as a report line, in this way.

10. You may code the clauses of any report group entry in any order, except in the case of a multiple-choice entry, where each SOURCE VALUE or FUNCTION operand is immediately followed by WHEN condition. Details of this combination are given later (see 3.17 PRESENT AFTER clause).

11. However, you may code the SUM clause more than once in the same entry. You may also code a SOURCE VALUE or FUNCTION clause and its associated PRESENT/ABSENT WHEN/AFTER clause more than once in a multiple-choice entry. All other clauses may appear no more than once in an entry.

### 3.2.3 Compatibility

1. The COUNT, GROUP LIMIT, VARYING, PRESENT/ABSENT WHEN/AFTER, FUNCTION, REPEATED, MULTIPLE PAGE, STYLE, and WRAP clauses are provided by new Report Writer only.

2. Only new Report Writer allows the SIGN clause in the REPORT SECTION.

3. Only new Report Writer allows the BLANK WHEN ZERO and JUSTIFIED clauses in a group entry.

4. OS/VS and DOS/VS COBOL do not use level-numbers, apart from 01-level, to establish hierarchy and rely instead on the keywords TYPE, LINE, and COLUMN.

5. OS/VS and DOS/VS COBOL allow just three hierarchic levels within the Report Group descriptions.

6. OS/VS and DOS/VS COBOL do not permit REPORT SECTION entries to be the subject of non-Report Writer procedural statements.
3.3 BLANK WHEN ZERO clause

This causes a field with zero contents to be blanked out.

3.3.1 Format

```
     BLANK    WHEN    ZERO
```

3.3.2 BLANK WHEN ZERO Clause: Coding Rules

If you code BLANK WHEN ZERO in an elementary entry, the entry must have a numeric PICTURE. If you code it in a group level entry, it applies to all the numeric elementary entries within the group. (This clause is usually referred to as the BLANK WHEN ZERO clause, even though the word WHEN is optional.)

3.3.3 BLANK WHEN ZERO Clause: Operation

BLANK WHEN ZERO causes a numeric field to be replaced entirely by spaces if its value is zero. For example:

```
| 1,099.50 | 19.27 | 20.90 | 160.00 |
```

```
05 COL 1   PIC Z,ZZZ.99   BLANK WHEN ZERO
OCCURS 4 STEP 10 VARYING RW-X SOURCE WS-VAL (RW-X).
```

You may use BLANK WHEN ZERO even if your field is not in a fixed position. It is also permitted with a variable-length field (PICTURE symbol "<"), in which case a zero value in a variable part has a length of zero.

3.3.4 Compatibility

Only new Report Writer allows this clause at the group as well as the elementary level.
3.4 COLUMN clause

The COLUMN clause specifies the horizontal positioning of a field in the report line.

3.4.1 Format

```
<table>
<thead>
<tr>
<th>COLUMN</th>
<th>NUMBER</th>
<th>IS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL</td>
<td>NUMBERS</td>
<td>ARE</td>
</tr>
<tr>
<td>COLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLUS</td>
<td>integer-1</td>
<td></td>
</tr>
<tr>
<td>+ integer-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3.4.2 COLUMN Clause: Coding Rules

1. COLUMN with no operands is shorthand for COLUMN +1.
2. For every entry with a COLUMN clause, there must be a LINE clause at the same or a higher level. So you may write:

```
05 LINE 5  COL 20  VALUE "TITLE PAGE".
```

provided that there are no other entries within LINE 5. If the line has two or more fields, you must then create a new level, as with:

```
05 LINE 6.
07 COL 5  VALUE "REPORT XYZ".
07 COL 72  VALUE "ANNUAL RETURNS".
```

If you code a COLUMN entry at the same level as the preceding LINE:

```
05 LINE 6 COL 5  VALUE "REPORT XYZ".
05 COL 72  VALUE "ANNUAL RETURNS".
```

then Report Writer will diagnose this as invalid but will allow it as stated.

3. COLUMN may be the only clause in an entry. The result is a blank field whose only purpose is to shift the current horizontal position (COLUMN-COUNTER) to the right. A blank field (absolute or relative) occupies one column's width in addition to the usual spacing for the COLUMN. It is therefore equivalent to coding a VALUE consisting of a single space alongside the COLUMN clause (although doing so would be less efficient). For example, COL + 4 coded alone in an entry shifts the current horizontal position four (not three) columns right. (See the discussion in the next section.)
4. The size of your item is calculated from the PICTURE clause or the size of the VALUE "literal", and is used in combination with your COLUMN clause to check that: (a) the line width is not exceeded (see 2.7 LINE LIMIT clause and, if you are using automatic line wrap, see 3.28 WRAP clause), and (b) no two items overlap (unless they both carry a PRESENT AFTER clause). If two or more items overlap in whole or in part, report writer will diagnose this as invalid but will allow the COLUMN positions and field sizes as coded. The later-coded entry will then overwrite the earlier-coded entry at run time by the number of overlapping columns and data will be lost. No run time error occurs.

5. Within each LINE, any absolute COLUMN numbers should be in ascending order, after the evaluation of any PRESENT WHEN clause (see 3.18) and any PRESENT AFTER clause (see 3.17). If this rule is broken, report writer will issue a Warning (message 250) but will allow the COLUMN positions as coded.

6. The RIGHT and CENTER phrases cannot be coded with the + (relative) form of this clause.

7. You may write "+" with or without a space on either side.

3.4.3 COLUMN Clause: Operation

1. The COLUMN clause positions your elementary field horizontally. Here is a list of the options:
   a. COLUMN + integer-1. This is the relative form. It indicates that the horizontal position within the line is to be moved integer column positions from the last character of the preceding field to the first character of this field. Note that the "gap" before the field will be one less than the value of the integer. For example:
      COLUMN +1 = "no gap"
      COLUMN +2 = "one space before field", and so on.
      If you use COLUMN + integer in the first field of a line, it is treated as though you had written the absolute form, COLUMN integer-1, since the initial value of the horizontal position is zero.
   b. COLUMN integer-2. This is the absolute form. It indicates that the left-hand column of the field must appear on that fixed position within the line. Remember that the first column in the line is COLUMN 1. LINE LIMIT is the highest possible value of integer.
   c. COLUMNS integer-1 integer-2 ... This is the multiple form of the clause. It reduces your coding effort by including several operands in the same clause. The relative multiple form, COLUMNS + integer, + integer ..., is also allowed and you may combine both forms in the same clause. Multiple COLUMNS are described in more detail below (see 3.4.4 Multiple COLUMNS Clause).
   d. COLUMN RIGHT and COLUMN CENTER are used if you wish to specify the right-hand or center position of the field as alternative anchoring points, to save you the effort of "counting out" the length of the field to establish its left-hand column when you already know its center point or right-hand column. COLUMN LEFT, the normal alignment, is provided for syntactic completeness.
If you specify COLUMN CENTER and the field is an even number of column positions wide, the extra character position goes to the right of the central column. The following example shows the allowable alternatives for a six-character field starting in column 1:

\[
\begin{array}{|c|c|c|}
\hline
\text{COLUMN 1} & \text{COLUMN RIGHT 6} & \text{COLUMN CENTER 3} \\
\hline
\end{array}
\]

You will get a ragged left-hand side in the case of COLUMN RIGHT and centralization with ragged left and right, in the case of COLUMN CENTER.

By successively generating the following entry on different lines with different values:

```
05 COL RIGHT 15 PIC 9(6) SOURCE POPULATION-COUNT.
```

you will obtain:

```
   1
  234
56789
8765432
```

↑ column right 15

The following, generated several times with a different CITY:

```
05 COL CENTER 20

"LONDON" WHEN CITY = "L"
"BATH" WHEN CITY = "B"
"GLASGOW" WHEN CITY = "G"
"KINGSTON UPON HULL" WHEN CITY = "H".
```

gives you, for example,

```
LONDON
BATH
GLASGOW
KINGSTON UPON HULL
```

↑ column center 20

Note that RIGHT and CENTER may also be used with the multiple format of the COLUMN clause discussed in 3.4.4 Multiple COLUMNS Clause.

The CENTER or RIGHT option is required if you need to center or right-align a variable-length field - (see "<" and ">" symbols) for full details. The output in the box above could also have been produced using a SOURCE item thus:

```
05 COL CENTER 20 PIC <X(20) SOURCE CITY-NAME.
```
2. If an elementary item has **no COLUMN** clause then, if the OSVS precompiler option is in effect or a data-name is present, the item will **not** appear in the report. It is then termed an **unprintable item**. Unprintable items are used chiefly for summing in the following cases:

   a. For **Subtotalling and SOURCE SUM Correlation** (see 3.23.5). A **SOURCE clause** may be written as an unprintable item because a **SUM** of a certain data item is required but its individual values are not:

   ```
   01 DUMMY-DETAIL TYPE DE.
   05 PIC 9(6) SOURCE IS WS-PAY. *> unprintable
   01 TYPE CF ...
   03 LINE ...
   05 COL 22 PIC Z(6)9 SUM WS-PAY.
   ```

   b. For **rolling forward** of certain values into totals. This is **report writer** principal way of forming totals. This time the unprintable item has a data-name and the data-name is summed:

   ```
   01 DUMMY-DETAIL TYPE DE.
   05 R-PAY PIC 9(6) SOURCE IS WS-PAY.
   01 TYPE CF ...
   03 LINE ...
   05 COL 22 PIC Z(6)9 SUM OF R-PAY.
   ```

   c. For forming totals that are not printed **directly** but used **indirectly**:

   ```
   01 TYPE CF ...
   05 R-PAY PIC 9(7) SUM OF WS-PAY.
   05 R-TAX PIC 9(7) SUM OF WS-TAX.
   05 COL 22 PIC Z(6)9 SOURCE R-PAY – R-TAX.
   ```

   Further examples may be found in **SUM Clause**.

3. If the precompiler option **OSVS** is **not** in effect, and any elementary entry beneath the **LINE level** has **no COLUMN clause**, then **COLUMN + 1 is assumed** for the entry, **provided that the level-number is not followed by a data-name**. Thus, you could omit both **COLUMN clauses** in the following fragment:

   ```
   05 LINE 1.
   07 COL 1 VALUE "REPORT ".
   07 COL 8 PIC XXX SOURCE REPORT-IDENT.
   ```

### 3.4.4 Multiple COLUMNS Clause

You may use the multiple form of the COLUMN clause by placing several **integer** or **+ integer** terms after the keyword. This reduces the effort needed to code several adjacent entries that have a similar format. Note the following points:

1. A multiple COLUMNS clause is functionally equivalent to an ordinary COLUMN clause used in conjunction with an **OCCURS clause** (see 3.14); for example:

   a. You may use **VARYING** to vary a counter that is used as a subscript in a **SOURCE clause**.
b. You may use a simple (single-operand) `VALUE SOURCE SUM`, or `FUNCTION` clause to place the same value repeatedly in each instance of the field.

c. You may use a multiple `VALUE` or `SOURCE` clause to place a different value in each instance of the field.

d. You may place a data-name at the start of the entry and `SUM` the data-name in another entry to produce a total of the instances or occurrences collectively or individually. (To form individual totals, there must be another multiple or occurring COLUMN clause in the entry with the SUM clause.)

2. Unlike the method of repetition using the OCCURS clause, the intervals between the entries defined by a multiple COLUMNS clause need not be regular.

3. Your multiple COLUMNS clause will be syntactically correct if it would be correct when written as a series of separate COLUMN entries.

4. Here are some examples of the multiple COLUMNS clause:

   *This places the literal in all 3 column positions:
   ```
   05   COLS 9 21 36     VALUE "-------".
   *
   *This gives you WVAL (1) WVAL (2) and WVAL (3) in the 3 right-hand positions:
   ```
   ```
   05   COLUMNS RIGHT 21 31 42     PIC ZZZZ9 
   VARYING R-SUB           SOURCE WVAL (R-SUB).
   ```
   ```
   *This illustrates the combining of absolute and relative positions:
   ```
   05    COLUMN NUMBERS ARE 6 +3 +3 ...
   ```

5. The following diagram and corresponding code illustrates the usefulness of the multiple COLUMNS clause:

   **SPORTS EQUIPMENT COMPANY: WAGES SUMMARY**

<table>
<thead>
<tr>
<th>AREA</th>
<th>BASIC PAY</th>
<th>OVERTIME</th>
<th>COMMISSION</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORTH</td>
<td>$1,420,000</td>
<td>$600,000</td>
<td>$150,500</td>
<td>$2,170,500</td>
</tr>
<tr>
<td>EAST</td>
<td>$2,100,000</td>
<td>$850,000</td>
<td>$220,000</td>
<td>$3,170,000</td>
</tr>
</tbody>
</table>

   ```
   01   TYPE PH.
   03  LINE 1  COL CENTER 28 "SPORTS EQUIPMENT COMPANY: WAGES SUMMARY".
   03  LINE 2  COLS RIGHT 4 19 33 47 62 VALUES
<table>
<thead>
<tr>
<th>&quot;AREA&quot;</th>
<th>&quot;BASIC PAY&quot;</th>
<th>&quot;OVERTIME&quot;</th>
<th>&quot;COMMISSION&quot;</th>
<th>&quot;TOTAL&quot;</th>
</tr>
</thead>
</table>
   03  LINE 4  OCCURS 4          STEP 1     VARYING AREA-NO.
   05   COL 1  "NORTH" | "EAST" | "SOUTH" | "WEST".
   05   R-PAY  COLS RIGHT 19 33 47 BASIC-PAY OVERTIME SOURCES
   | PIC $$$, $$ $$, $$$ | TAX. |
   05   COL RIGHT 62  PIC $, $$ $$, $$ $$ SUM OF R-PAY.
   ```

3.4 – COLUMN
3.4.5 Compatibility

Only new Report Writer has the following features:

- Abbreviation of keyword as COL,
- Relative form (+ or PLUS),
- CENTER and RIGHT options,
- Multiple format,
- Blank entry when COLUMN clause present alone.
3.5 COLUMN-COUNTER

COLUMN-COUNTER is a special register that, at any given instant, contains the value of the rightmost column number of the most recently processed report field.

3.5.1 Format

```

>>>COLUMN-COUNTER<<<
```

3.5.2 Use of COLUMN-COUNTER

1. COLUMN-COUNTER indicates the current horizontal position within a line. At each stage during the processing of a line, it contains the most recent rightmost column number on which data was placed in the line. It may also be incremented by an "empty" (unprintable) entry containing only a COLUMN clause. Regardless of the number of reports and report lines it contains, each program has just one COLUMN-COUNTER.

2. Report writer uses COLUMN-COUNTER as a place-keeper and internal subscript while assembling variable-length or variable-position fields in a report line. For the sake of efficiency, it does not initialize or update COLUMN-COUNTER in a report line consisting of fixed-length, fixed-position fields, unless you explicitly refer to it in an item in the line.

3. You may use COLUMN-COUNTER in the condition of a PRESENT WHEN clause (see 3.18), or as the operand of a SOURCE clause (see 3.21), or as a parameter to a FUNCTION (3.7 FUNCTION clause). It cannot be used in your PROCEDURE DIVISION, because the generating of each line is an indivisible operation.

4. COLUMN-COUNTER may be used to simplify the conditions that would otherwise be required to string together a series of conditional fields:

```
05  COL 30.
05  COL + 1  "A"  PRESENT WHEN A-FLAG = "Y".
05  PRESENT WHEN B-FLAG = "Y".
07  COL + 1  ","  PRESENT WHEN COLUMN-COUNTER > 30.
07  COL + 1  "B".
```

5. If your current group is a REPEATED group (see 3.19 REPEATED clause), or if you use the SET COLUMN statement of the Page Buffer feature (see 4.4 Report Writer SET statements), you should bear in mind that COLUMN-COUNTER is relative to the left-hand boundary of the group. So it does not include the extra left-hand margin used to offset your REPEATED groups or the extra margin you create with SET COLUMN.

6. COLUMN-COUNTER should not be used in a condition where the field is to be totalled using the SUM clause, because COLUMN-COUNTER is not updated when totalling is performed at the start of processing for the group.

3.5.3 Compatibility

COLUMN-COUNTER is unique to new Report Writer.
3.6 COUNT clause

The COUNT clause counts the number of appearances of any specified report field or group item.

3.6.1 Format

```
COUNT OF report-section-data-name
  \[ \text{RESET ON control-id} \text{ ROUNDED} \]
```

3.6.2 COUNT Clause: Coding Rules

1. Each data-name must be the name of any REPORT SECTION entry other than an RD. It may even be the data-name of a group entry that contains the COUNT clause.

2. You may place the COUNT clause in a different report group from the item counted (rolling forward) or in the same group (cross-footing). As with the SUM clause, if the COUNT clause appears in a multiple CONTROL FOOTING, all but the lowest level total is formed by rolling forward the next-lower total.

3. Unlike the situation with the SUM clause, the item counted need not be a numeric field.

4. Like the SUM clause, you may use the COUNT clause in two ways:
   a. As a clause in its own right. You write the clause in place of a SOURCE, VALUE, or FUNCTION clause. For example:

   ```
   05 COL 21 PIC ZZZ9 COUNT OF R-CUSTOMER-NAME.
   ```

   b. As a term in an expression used as a SOURCE operand; for example:

   ```
   05 COL 21 PIC ZZ9 SOURCE IS (COUNT OF R-GAIN) - (COUNT OF R-LOSS).
   ```

5. You may combine SUM and COUNT terms in the same expression. For example, the following will give you the average value of all the instances of a numeric field:

   ```
   05 COL 32 PIC ZZZ99
   SOURCE (SUM OF PURCHASE) / (COUNT OF PURCHASE) ROUNDED.
   ```
3.6.3 COUNT Clause: Operation

1. This clause gives a count of the number of times the item referenced has appeared. In other words, whenever the item appears in the Report, 1 is added to the count. You cannot count items that are outside the REPORT SECTION, such as WORKING-STORAGE items.

2. The item referenced may be at any level; for example:
   a. A 01-level entry: you will obtain the number of appearances of a particular group.
   b. A LINE entry: you will count the number of appearances of that particular LINE.
   c. A COLUMN entry: you will obtain a count of appearances of that particular elementary item.

3. Assuming that you do not use the RESET phrase, when your COUNT field has been output the count returns to zero. Hence, you will always obtain the count of the number of appearances of the item since the last time the count was output.

4. You may use the RESETON phrase to delay the resetting of the count to zero until a higher-level control break occurs, in exactly the same way as you can with the SUM clause.

5. If the item counted is one that repeats several times because of an OCCURS or REPEATED clause, the count will include each repetition. An OCCURS ... DEPENDING will count just the number of items actually output. A PRESENT clause (or the equivalent) is also taken into account, so that items "not present" do not contribute to the count.

6. You may count more than one item by writing more than one data-name as an operand in the clause. The counts are then simply consolidated internally.

   (The COUNT clause is really a special variant of the SUM clause, so other points of interest can be found under 3.23 SUM clause.)

3.6.4 Compatibility

Only new Report Writer provides the COUNT clause.
3.7 FUNCTION clause

The FUNCTION clause invokes a special routine to reformat or convert the data before it appears in your report field.

3.7.1 Format

```
FUNCTION IS function-name

( parameter )
```

3.7.2 FUNCTION Clause: Coding Rules

1. Function-name must be alphanumeric and must be either a standard supplied function name or a user-written one. It may begin with a numeric character. A full list of standard function names is given below, and you or any other person may add new ones to the list at any time.

2. The FUNCTION clause may be used only at the elementary level, in place of SOURCE, SUM, or VALUE. It is permissible for the entry to have no COLUMN or PICTURE clause, i.e. the entry may be a dummy. You may use it in a multiple-choice entry, if so desired.

3. The number of parameters you need, if any, and their formats are stated below (see 3.7.4 Built-in Functions) for each built-in function. User-written functions should also have a description that tells you this information. Some functions take no parameters while others allow either a fixed or a variable number.

4. Each parameter may be any identifier. It may also be an integer, literal, or arithmetic expression, for example:

```
FUNCTION DAYSIN (SALES-DATE + 365).
```

in which case the parameter is given to the Function routine as a one-word signed binary field. You cannot use SUM or COUNT terms in the expression. Neither can you use LENGTH OF, ADDRESS OF, BY CONTENT, or BY REFERENCE, as part of a parameter.

5. The FUNCTION keyword may also be used in the format FUNCTION IS function-name (...) in a SOURCE clause or in a condition in a PRESENT WHEN clause. In such cases function-name is a COBOL intrinsic function and is not part of the FUNCTION clause described here.
3.7.3 FUNCTION Clause: Operation

1. The function routine behaves like a subroutine whose parameters consist of the parameters specified in the FUNCTION clause, if any, together with information about the size of the report field, if any, and the report field's contents, both of which are automatically supplied by report writer. Details on the linkage to function routines are given later (see 5.2 Developing User-Written Functions).

2. If you code an edited PICTURE (numeric or alphanumeric) with the FUNCTION clause, report writer invokes the function routine using an internal intermediate field that has the unedited equivalent of your PICTURE. This is obtained by removing all insertion characters, including any decimal point. Also any "Z" or "*" symbol or floating sign or currency symbols are converted to an equivalent number of "9" symbols. The "<" symbol is also removed. For example, if your entry is as follows:

| 05 COL 21   | PIC $$,$$9.<99 FUNCTION MONEY (WS-VAL). |

Report writer will provide the function routine with an intermediate field with a PICTURE of 9999V99 as a parameter for receiving the output value. Report writer then executes a MOVE from the intermediate field to your original report field. This means that all functions, including user-written ones, can operate with edited report fields, without function routine developers needing to allow for the many possible edited formats.

3. One of the parameters report writer passes to the function routine is the size in bytes of the report field, if any, to be output. If the PICTURE is edited, this size is the size of the intermediate field. Using the case given in the previous paragraph, for instance, the function routine will be told that the size is 6. The function routine may decide to produce different formats for the output report field, depending on the number of bytes implied by this size. It may also simply truncate the output value (on the left or the right) down to the number of characters required. This can give you freedom to choose from many possible PICTURES to use with a function. For example, if MDATE is used, and today's date is 5th January, 1997:

- PICTURE 9(6) will give you: 010597
- PICTURE 99/99/99 will give you: 01/05/97
- PICTURE XXXB99B9(4) will give you: JAN 05 1997
- PICTURE <X(9)B<99B9999 will give you: JANUARY 05 1997

Functions cannot detect the kind of PICTURE symbols used. For example PICTURE XXBXXBXX produces the same result as PICTURE 99B99B999.

4. The clause SOURCE IS CURRENT-DATE behaves operationally like an implicit “Function”, since it invokes a run time routine.
3.7.4 Built-In Functions

The following is a list of the built-in functions.

**CTIME** (Clock Time) is similar to **TIME** (see below) except that it uses the 12-hour clock and prints either AM or PM (or blanks at midnight (00.00) and noon (12.00)). The (unedited) PICTURE should therefore allow for two additional non-numeric characters. For example, **PIC 99'99BXX** could result in **11:30 PM**.

**DATE** (European Date) returns a date in any one of a number of display formats in the order day/month/year. The date does not change after the first invocation of the function.

Number of parameters: 0 or 1.

If no parameter is supplied, the date will be the current date at the start of the run. If one parameter is supplied, it must be either a 7-digit date held in a **PIC S9(7) COMP-3** (packed) format in the form **ccyyddds** (**s** = sign), or a 5-digit date held in a **PIC S9(5) COMP-3** (packed) format in the form **yyddds** (in which case the current century is assumed).

Report-field (PIC) lengths (excluding editing symbols):
- 5 - yydd
- 6 - ddmmyy
- 7 - dddMMyy (MM is first 3 characters of month name)
- 8 - ddmccyy (cc is century)
- 9 - dddMMccyy
- 13 - dddM(9)yy (M(9) is 9-character month name)
- 15 - dddM(9)ccyy

**DAY** (Day-of-Week) returns the alphabetic day-of-the-week represented by the given date.

Number of parameters: 0 or 1.

If no parameter supplied, the current date is used. If one parameter is supplied, it may be either a 7-digit date held in a **PIC S9(7) COMP-3** (packed) format in the form **ccyyddds** (**s** = sign), or a 5-digit date held in a **PIC S9(5) COMP-3** (packed) format in the form **yyddds** (when the current century is assumed), or a two- or four-byte (PIC S9(9) COMP or PIC S9(4) COMP) location containing a value from 1 (= Monday) to 7 (= Sunday), or a one-byte (PIC 9 DISPLAY) location containing a value from 1 to 7.

Report-field lengths (excluding editing symbols):
- 3 - first 3 letters of day (MON, TUE, WED, THU, FRI, SAT, SUN)
- 9 - full name of day

**DAYSIN** (Days Elapsed Since Base Date) converts a binary number of days since January 1st, 1601 to a date in the same format as for DATE. This function is identical to MDAYS, except that the date is returned in the order: day/month/year.
**MDATE** (US Date) returns a date in any one of a number of display formats in the order month/day/year. It is similar to the DATE function, except that the day and month fields are reversed.

Report-field lengths (excluding editing symbols):
- 5 - yyddd
- 6 - mmddyy
- 7 - MMMddyy (MM is first 3 characters of month name)
- 8 - mmddccyy (cc is century)
- 9 - MMMddccyy
- 13 - M(9)ddyy (M(9) is a 9-character month name)
- 15 - M(9)ddccyy

**MDAYS** (Days Elapsed - US Format) converts a binary number of days since the Codasyl/ANSI standard base date of January 1st, 1601 to a date in the same format as MDATE, that is, month/day/year. See MDATE above for more information on report field lengths.

Number of parameters: 1.

The parameter must be the number of days since January 1st 1601 inclusive, held as one fullword binary (such as PIC S9(9) COMP). This is the base date, 584,693 days since absolute year zero (ignoring calendar reforms). A zero value means December 31 1600. A negative value gives a date before December 31 1600. You may change the base date by re-compiling the source of the FUNCTION supplied with this product, in which case your new function should have a different name to avoid incompatibility.

**MONTH** (Month Name) returns an alphabetic month name.

Number of parameters: 0 or 1

If no parameter is supplied, the current month is returned. If a parameter is supplied, it should be either a 7-digit date held in a PIC S9(7) COMP-3 (packed) format in the form ccyyddds (s = sign), or a 5-digit date held in a PIC S9(5) COMP-3 (packed) format in the form yyddds (when the current century is assumed), or a two-byte (PIC 99 DISPLAY) location containing a value from 01 to 12, or a four-byte binary (PIC S9(9) COMP) location containing a value from 1 to 12.

Report-field lengths (excluding editing symbols):
- 3 - first three characters of month (JAN,FEB,MAR,etc.)
- 9 - full name of month
MOVE (Save Register) copies any report writer special register to a specified working location. This is a dummy function which requires no report field and may be coded without a COLUMN or PICTURE clause. (If the entry is not a dummy, the field will be space-filled.)

Number of parameters: 2.

Parameter 1 is any numeric special register, typically LINE-COUNTER or COLUMN-COUNTER. It may also be a user-defined data-name specified in a VARYING clause.

Parameter 2 is any half-word binary location (PIC S9(4) COMP) to receive the contents of parameter 1.

This FUNCTION is used to capture on the fly the contents of a register whose value is constantly changing. The following sample code prints up to 12 monthly payments to print side-by-side showing only those which are not zero and, by storing each value of the VARYING subscript R-MONTH in LAST-MONTH for those occurrences selected, it is able to state which month was the last to be printed.

```
03 LINE.
  05 OCCURS 12 VARYING R-MONTH PRESENT WHEN PAY (R-MONTH) > 0.
  07 COL + 3 PIC ZZZZ9 SOURCE PAY (R-MONTH).
  07 COL 90 "Last month with a payment was".
  05 COL + 2 PIC X(9) FUNC MONTH (LAST-MONTH).
```

**RDATE** (Real DATE) is similar to DATE without the optional parameter, except that the current date is always fetched. Compare RMDATE.

**RMDATE** (Real MDATE) is similar to MDATE without the optional parameter, except that, if the date changes during the run, because the time passes through midnight (00:00:00), the date is changed.

Report-field (PIC) lengths (excluding editing symbols):

- 5 - yyddd
- 6 - yymmd
- 7 - yyMMMdd (MMM is first 3 characters of month name)
- 8 - ccymmd (cc is century)
- 9 - ccyyMMmd
- 13 - yyM(9)dd (M(9) is 9-character month name)
- 15 - ccyyM(9)dd

**RYDATE** (Real YDATE) is similar to YDATE (see below) without the optional parameter, except that the current date is always fetched. Compare RMDATE.
**STATE** (US State) returns the name of one of the US states, plus "DC".

Number of parameters: 1

The parameter may be either a state number with PICTURE 99 (DISPLAY) ranging from 01 (ALABAMA) to 51 (WYOMING), in alphabetical order of their full names, or a two-character standard abbreviation e.g. "AL" or "WY".

The report-field length must be at least 14. If it is less than 20, then DISTRICT OF COLUMBIA is rendered as D.C.

**STATEF** (US State or Territory) is similar to **STATE** (see above) except that the five overseas territories are included, merged into the set of 51 domestic states, in alphabetical order of their full names.

**STIME** (Static Time) is similar to **TIME** (see below) except that the time is only fetched initially from the operating system and therefore does not change in value throughout the program.

**TIME** (Run Time) returns the current time in format **hhmmssstt**, where tt is hundredths of a second, if available, otherwise zeros. The value of the time may change on each invocation of the function.

Number of parameters: none

Report-field (PIC) lengths:

- 4  -  **hhmm**
- 6  -  **hhmmss**
- 8  -  **hhmmssstt**

As an example, the following example uses **DAY**, **DATE**, and **TIME**:

<table>
<thead>
<tr>
<th>05</th>
<th>COL 31</th>
<th>PIC &lt;X(9)</th>
<th>FUNC DAY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>COL +2</td>
<td>PIC &lt;99/&lt;99/9(4)</td>
<td>FUNC MDATE.</td>
</tr>
<tr>
<td>05</td>
<td>COL 51</td>
<td>PIC 99,99,99</td>
<td>FUNC TIME.</td>
</tr>
</tbody>
</table>

executed at 3 PM on March 7th, 1997 will result in:

```
FRIDAY 3/7/1997 15,00,00
```

**YDATE** (Date Reversed) returns a date in any one of a number of display formats in the order **year/month/day**. Apart from this order, it is similar to **DATE** and **MDATE**.

---

3.7 – FUNCTION 101
**ZIP**  
(Zip Code) prints a standard US or Canadian ZIP code.

Number of parameters: 1.

The single parameter must contain the ZIP code in the format $9(9)$ COMP-3. This is then output in the form nnnn-nnnn, but the final -nnnn is left blank if the last four digits are 9999.

### 3.7.5 Compatibility

Only new Report Writer provides FUNCTION as an independent clause.
3.8 GROUP LIMIT clause

This clause explicitly gives the lowest permissible vertical position for any line in a specific body group.

Format

►► GROUP LIMIT IS integer◄◄

3.8.1 GROUP LIMIT clause: Coding Rules

1. Your integer must not be greater than the lower limit (LAST DETAIL, LAST CONTROL FOOTING, or PAGE LIMIT) that would normally apply without the clause (see 2.9 PAGE LIMIT clause).

2. Code this clause only at the 01-level in a body group (DETAIL or CH/CF).

3. Your group's first LINE clause must be relative.

3.8.2 GROUP LIMIT clause: Operation

1. When doing the page-fit test, report writer will use your integer as the bottom line number, beneath which no part of the group may appear, instead of the usual end-of-region values (see PAGE LIMIT clause).

2. This clause is especially useful in a CONTROL HEADING, which might appear misplaced if it appeared near the bottom of the page like the DETAIL and CONTROL FOOTING groups.

3.8.3 Compatibility

The GROUP LIMIT clause is unique to new Report Writer.
3.9  **JUSTIFIED clause**

This clause causes a SOURCE, SUM, COUNT, VALUE, or FUNCTION clause whose operand is shorter or longer than the target report field to fill the report field with excess spaces, or perform truncation, on the left instead of the right.

3.9.1  **Format**

```
  ┌── Format ─────────────────────────────────────────────┐
  │                                                │
  │                                                      │
  │ ►►──┬── JUSTIFIED ─┬─ RIGHT──►◄                     │
  │                                                      │
  │                                                      │
  │ ────┴─── JUST ───────┘                                │
  │                                                      │
  │                                                      │
  │ ───────────────────────────────────────────────────────────────┘
```

3.9.2  **JUSTIFIED Clause: Coding Rules**

1. If you code JUSTIFIED RIGHT in an elementary entry, the entry must have an **alphanumeric** PICTURE.

2. This clause acts on elementary-level fields, but you may also code it in a group-level entry, including 01, where it applies to all the alphanumeric elementary entries in the group.

3.9.3  **JUSTIFIED Clause: Operation**

1. The JUSTIFIED clause is retained for compatibility with ANS COBOL and acts on an elementary field in the same way as in basic COBOL. It cannot be used for right-flushing variable-length fields, for which **COLUMN RIGHT** should be used. (See 3.4 **COLUMN clause**.) This is because JUSTIFIED does not consider the **contents** of the sending field.

2. The JUSTIFIED clause takes effect when your alphanumeric SOURCE field is of a different size from the PICTURE. The padding out with spaces or the truncation (if the PICTURE is smaller than the field) then takes place on the **left** instead of the right. In the following example, we want to output either MONTH-NUMBER (2 characters) in columns 12 and 13 or YEAR-NUMBER (4 characters) in columns 10 through 13:

   ```
   05   COL 10   PIC X(4) JUSTIFIED RIGHT
   SOURCE MONTH-NUMBER WHEN MONTH-IND = 1
   YEAR-NUMBER WHEN OTHER.
   ```

3.9.4  **Compatibility**

OS/VS and DOS/VS COBOL allow this clause only in elementary entries, but in all other respects new Report Writer and the older compilers treat it identically.
3.10 **LINE clause**

The LINE clause positions a line vertically on the page.

### 3.10.1 Format

```
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER IS</th>
<th>PLUS</th>
<th>integer-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NUMBERS ARE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LINES ARE</td>
<td>+</td>
<td>integer-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON NEXT PAGE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### 3.10.2 LINE Clause: Coding Rules

1. Here is a list of the alternative forms:

   a. **LINE + integer-1.** This is the relative form. **PLUS** may be written in place of +. It indicates that the line should advance integer lines from the previous position. **LINE + 0** or **LINE + ZERO** is allowed, indicating that no advance is to take place. The result is that the line overprints the previous line. You may write “+” with or without a space on either side.

   b. **LINE integer-2.** This is the absolute form. It indicates that the line will appear on that fixed position on the page. The first line on the logical page is LINE 1, corresponding to position reached on issuing a form feed. The highest line number allowed is given by the PAGE LIMIT operand. This absolute form is allowed only if you have a **PAGE LIMIT clause** in your RD.

   c. **LINE integer-2 ON NEXT PAGE.** This is similar to format b but the **ON NEXT PAGE** phrase forces a page advance to occur before the line is output, irrespective of whether the group containing it would fit on the current page.

   d. **LINE ON NEXT PAGE** forces the line to be output in the FIRST DETAIL position on the next page (for a body group) or the HEADING position (for a REPORT HEADING or REPORT FOOTING group).

   e. **LINES ARE integer-1 integer-2 ...** is the multiple form of the clause. It enables you to save coding effort by describing several lines in the same clause. You may also write **LINES + integer, +integer etc** (the relative form) and may combine both forms in the same clause. You may also write **ON NEXT PAGE** after the last operand, in which case this phrase applies to the first line in the set.

2. **LINE alone is shorthand for LINE + 1.**
3. The first LINE clause in each group determines whether the group as a whole is to be relative or absolute. If the first LINE clause is relative, the group is a relative group and all the other LINE clauses must be relative. So the entire group can be positioned anywhere on the page, within the permitted upper and lower limits for the group:

```
  ▲ ▲ ▲
relative report group                LINE + ...  LINE + ...
  ▼ ▼ ▼
```

If the first LINE clause is absolute, the group is an absolute group and any of the remaining LINE clauses may be absolute or relative. For example, the following two arrangements of LINE clauses within a group are equivalent:

<table>
<thead>
<tr>
<th>01 TYPE PH.</th>
<th>01 TYPE PH.</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 LINE 1. ...</td>
<td>03 LINE 1. ...</td>
</tr>
<tr>
<td>03 LINE 2. ...</td>
<td>03 LINE + 1. ...</td>
</tr>
<tr>
<td>03 LINE 4. ...</td>
<td>03 LINE + 2. ...</td>
</tr>
</tbody>
</table>

Using relative lines has the advantage that you can adjust the position of the entire group by changing just the first LINE number.

4. If a group has absolute LINE clauses, except where NEXT PAGE is used, the integers must be in increasing order. Except in the case of LINE + 0, no lines may overlap. (If you place PRESENT WHEN condition clauses on the LINE entries, then mutually exclusive lines may have the same line numbers. See 3.18 PRESENT WHEN clause.)

5. A LINE need not contain any COLUMN entries (whether actual or implied by default). The result is a blank line containing no data. For example:

<table>
<thead>
<tr>
<th>01 TYPE PH.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 LINE 2 VALUE &quot;** HEADING **&quot;.</td>
</tr>
<tr>
<td>05 LINE 4. * (or LINE + 2.)</td>
</tr>
</tbody>
</table>

This causes the PAGE HEADING to occupy lines 2 to 4, although data is written only on line 2.

6. One LINE clause must not be subordinate to another LINE clause. If this rule is broken, a warning message will be issued and the previous line will be terminated; for example:

<table>
<thead>
<tr>
<th>01 PAYMENT-LINES</th>
<th>LINE NEXT PAGE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 COLUMN 1</td>
<td></td>
</tr>
<tr>
<td>03 LINE PLUS 1.</td>
<td></td>
</tr>
<tr>
<td>05 COLUMN 1</td>
<td></td>
</tr>
</tbody>
</table>
7. The **ON NEXT PAGE** phrase may be coded only in the first LINE clause of a report group, except when the report group entry has a **MULTIPLE PAGE clause** (or when this is assumed, as in the case of REPORT HEADING and REPORT FOOTING) where there is no restriction. (See 3.12 **MULTIPLE PAGE clause**.) This permits such groups to occupy several pages.

3.10.3 **LINE Clause: Operation**

1. If a group is absolute, its first LINE clause indicates the starting position of the group (see rule 3 above). The next two paragraphs cover the relative case.

2. **Positioning of relative body groups (CH, DE, and CF)**
   
a. If the group is the **first** body group on the page, the first line of the group is positioned at the **FIRST DETAIL** position, irrespective of the integer of the LINE clause. If you did **not** code a FIRST DETAIL sub-clause in your RD, the first line will appear one line after the PAGE HEADING group, if there is one, or at the HEADING position if not. Subsequent lines in the group are positioned relative to each other.

b. If the group is **not** the first body group on the page, then the first line of the group is positioned relative to LINE-COUNTER, which normally contains the last line position of the preceding body group. (But note that a NEXT GROUP clause or an alteration to the value of LINE-COUNTER in the PROCEDURE DIVISION can affect this positioning.)

3. **Page-Fit Test for Body Groups**

When a report has a PAGE LIMIT clause, report writer performs a **page-fit test** before outputting any body group (CONTROL HEADING, DETAIL, or CONTROL FOOTING). This is to ensure that none of the lines of the group will be output unless all the lines of the group will fit in the region of the page reserved for its type of body group. (See 2.9.3 **PAGE LIMIT Clause: Operation**.) The type of page-fit test performed will depend on whether the group is absolute or relative. In all cases, remember that LINE-COUNTER contains the current vertical position, normally the position of the line last written before this group was generated.

   
a. **Absolute Page-Fit Test for Body Groups**

   If the first LINE is **absolute**, report writer takes the first LINE number as the starting position of the group. (If any of the absolute LINE clauses are conditional, then this "first line" need not be the first LINE number coded; if all absolute lines are **ABSENT**, the group will be treated as null or relative, as the case may be.) If LINE-COUNTER is positioned at a line **above** the starting line position of the group, report writer will **not** skip to a new page. In all other cases, a page advance takes place (see item 5 **below**) and the group is printed at the line specified on a fresh page. The next diagram illustrates this process:
b. Relative Page-Fit Test for Body Groups

When all the LINE clauses in the group are relative, report writer computes the total of all the integers of the LINE + clauses. This gives the total size of the group. If the group contains LINE entries that have a condition attached, as from a PRESENT WHEN/AFTER clause, or from an OCCURS ... DEPENDING clause, all these clauses are evaluated first, so that the actual number of lines about to be produced is known. The value LINE-COUNTER + total-size-of-group is the last line on which the group would appear if it were produced on the current page. If this last line position is beyond the lower permitted limit of the group, the group cannot fit on the page and a page advance takes place.

As shown by the diagram and rules under PAGE LIMIT clause, and GROUP LIMIT clause, the lower permitted limit of the group is given by:

- The report group's own GROUP LIMIT clause, if there is one, or
- The RD's LAST DETAIL, if the group is a DETAIL or CONTROL HEADING
- The RD's LAST CONTROL FOOTING if the group is a CONTROL FOOTING.

The LAST DETAIL and LAST CONTROL FOOTING sub-clauses need not be explicitly specified in your RD, as report writer will assume defaults for them, as outlined under PAGE LIMIT clause (see above).
The following diagram shows an example of a page-fit test.

**BODY-GROUP**

*total size 3 lines*

---

<table>
<thead>
<tr>
<th>RD</th>
<th>FIRST DETAIL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAST DETAIL 60</td>
</tr>
<tr>
<td></td>
<td>PAGE LIMIT 64</td>
</tr>
</tbody>
</table>

---

**... GENERATE BODY-GROUP ...**

<table>
<thead>
<tr>
<th>01</th>
<th>BODY-GROUP TYPE DE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>LINE + 2...</td>
</tr>
<tr>
<td>03</td>
<td>LINE + 1...</td>
</tr>
</tbody>
</table>

---

Initial LINE-COUNTER = 10;
10 + 3 = 13 = last LINE position;
13 not > 60, so **enough room**.

Later, on the same page:

**... GENERATE BODY-GROUP ...**

<table>
<thead>
<tr>
<th>01</th>
<th>BODY-GROUP TYPE DE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>LINE + 2...</td>
</tr>
<tr>
<td>03</td>
<td>LINE + 1...</td>
</tr>
</tbody>
</table>

---

Initial LINE-COUNTER = 58;
58 + 3 = 61 = last LINE position;
61 > 60, so **NOT enough room**, so **page advance takes place**.

← Second group produced at FIRST DETAIL position, line 3.

---

4. **Positioning of relative non-body groups (RH, PH, PF, and RF)**

a. **TYPE RH and PH**: relative to **HEADING minus 1**. (However, in the case of the first PH group when there is a RH group also on the first page, it is relative to the last line of the RH group.)

b. **TYPE PF**: relative to **LAST CONTROL FOOTING**.

c. **TYPE RF**: relative to **HEADING minus 1**. (However, in the case where the RF group does not begin on a new page, it is relative to the last line of the PF group, if there is one, or to LAST CONTROL FOOTING, if there is no PF group.)
5. **Page advance processing**

When report writer executes a page advance, it does the following:

- If you defined a **PAGE FOOTING**, this is output.
- **PAGE-COUNTER** is incremented by 1.
- An advance is made to the top of the next page. In "batch" printing a form feed is output, but with an Independent Report File Handler the action may vary (see 5.3 Independent Report File Handlers). **LINE-COUNTER** is then set to zero.
- If you defined a **PAGE HEADING**, this is output.

### 3.10.4 Multiple LINES Clause

By writing several integer or + integer terms in a LINE clause, you save time in defining a group of lines that all have a similar layout. Note the following points:

1. A multiple LINES clause is functionally equivalent to a **LINE** with an **OCCURS** clause; for example:
   - **a.** You may use a **VARYING** clause to vary an internal counter that may be used as a subscript in a **SOURCE** clause within the scope of the **LINE** clause.
   - **b.** A simple (single-operand) **VALUE**, **SOURCE**, **SUM**, or **FUNCTION** is repeated in every occurrence of the line.
   - **c.** You may use a multiple **VALUE** or **SOURCE** clause to place a different value in a report field in each occurrence of the multiple **LINE**.
   - **d.** You may place a **data-name** at the start of the entry and **SUM** it into another entry to produce a total of all the (multiple) entries.

2. However, in contrast to the **OCCURS** clause, the intervals between the lines defined by a multiple **LINES** clause need not be regular.

3. Your multiple **LINES** clause will be syntactically correct if it would be correct when written as a series of **LINE** clauses in separate entries.

4. Here are some examples of the multiple **LINES** clause:

   a. 03 LINES 1, 3, 4.
   b. 03 LINE NUMBERS ARE +2, +1, +1  COL 6 VALUE "----".
      Here the ON NEXT PAGE phrase applies to the first of the three lines:
   c. 03 LINES 1, +2, +2 ON NEXT PAGE.

5. The multiple **LINES** clause is useful in lines, such as headings, that have stacked text. Consider the following example in conjunction with the description under **Multiple SOURCES** and **Multiple VALUES**.
It may be coded as follows:

```
01 TYPE PH.
03 LINES 1 2 3.
   05 COL 1 VALUES "SALES"
         "THIS"
         "YEAR".
   05 COL 11 VALUES "" ""
         "LAST"
         "YEAR".
   05 COL 21 VALUES "GROSS"
         "PROFIT"
         " ".
   05 COL 35 VALUES "PAGE" " " " ".
* We use a blank "literal" or SOURCE NONE
* in any occurrence that is to be empty.
   05 COL 40 PIC ZZ9 SOURCES PAGE-COUNTER, NONE, NONE.
```

Of course, you do not have to code the multiple VALUES in a "stacked" format like this, but it is suggested to aid readability.

### 3.10.5 Compatibility

- Only new Report Writer provides the following features:
- Use of + as alternative to PLUS,
- The multiple format,
- LINE PLUS ZERO as alternative to LINE PLUS 0,
- Blank entry with LINE clause only to provide blank line,
- Absolute LINE allowed to follow relative LINE,
- Absolute LINES may advance to new pages within RH, RF, and body groups.
3.11 **LINE-COUNTER**

LINE-COUNTER is an internal special register that contains the most recent report line number on which data was produced, or which has been advanced to by a blank LINE clause or a NEXT GROUP clause. LINE-COUNTER may also be altered by an ordinary PROCEDURE DIVISION statement or implicitly by Report Writer SET statements.

### 3.11.1 Format

```

```

### 3.11.2 Uses of LINE-COUNTER

LINE-COUNTER is a special register giving the current vertical position on the page. After a GENERATE has been executed, LINE-COUNTER will be set to the line number of the last line occupied by the last group output. A NEXT GROUP may change it further (see NEXT GROUP clause). You can test the value of LINE-COUNTER to determine the vertical position on the current page.

Whenever you issue a GENERATE statement, report writer examines LINE-COUNTER to decide where it should output the lines of the group. It is set to zero by the INITIATE and by a page advance. Then, if your line has a LINE + integer clause, report writer uses \((\text{LINE-COUNTER} + \text{integer})\) as its target line. When the page-fit test is performed for a relative body group, report writer adds the size of your group to LINE-COUNTER to see whether the last line will be below the group’s lower limit.

You may alter the value of LINE-COUNTER in the PROCEDURE DIVISION, but take care! **Do not reduce** the value of LINE-COUNTER, or you are liable to cause too many lines to be written to your page so that it overflows its lower limit. You may **increase** its value safely. Some legitimate reasons for altering its value are:

1. To create an additional gap between the previous group and the next group:
   
   ```
   ADD size-of-gap TO LINE-COUNTER
   ```

   If your page is nearly full (so that LINE-COUNTER is nearly equal to the LAST DETAIL value) this will simply cause a page advance upon the next GENERATE instead of a gap. If your ADD statement sets LINE-COUNTER below LAST DETAIL, no harm is done. You should note that the NEXTGROUP clause performs this function more elegantly.

2. To cause a page advance, that is, to force report writer to place no more body groups on this page, code:

   ```
   MOVE page-limit TO LINE-COUNTER
   ```
Again, coding the NEXT GROUP NEXT PAGE clause on the previous group also performs this function better. (See 3.13 NEXT GROUP clause.)

3. If your REPORT SECTION has several Report Descriptions, each will have its own distinct LINE-COUNTER. In the main-line PROCEDURE DIVISION, you will therefore need to qualify LINE-COUNTER by the name of the report. (See 3.15 PAGE-COUNTER, and 5.1 Multiple Reports.)

3.11.3 Compatibility

1. OS/VS and DOS/VS COBOL also allow LINE-COUNTER to be altered explicitly, but this does not create a gap.

2. OS/VS and DOS/VS COBOL Report Writer require LINE-COUNTER to be qualified wherever it is used if your program contains more than one RD.
3.12 **MULTIPLE PAGE clause**

This clause allows a report group to span more than one page.

### 3.12.1 Format a

```
▶ MULTIPLE PAGE ◄
```

### 3.12.2 Format b

```
►► NO MULTIPLE PAGE ◄◄
```

### 3.12.2 MULTIPLE PAGE Clause: Coding Rules

1. This clause can be coded in any type of group other than PAGE HEADING and PAGE FOOTING. The RD must have a PAGE LIMIT clause.

2. Either MULTIPLE PAGE or NO MULTIPLE PAGE may be coded at the 01 level. Alternatively, NO MULTIPLE PAGE may be coded on a group of one or more LINE entries, provided that there is a MULTIPLE PAGE at the 01 level. No other nesting is allowed.

3. By default, NO MULTIPLE PAGE is assumed for all body groups and MULTIPLE PAGE for REPORT HEADING and REPORT FOOTING groups.

4. The MULTIPLE PAGE clause is not allowed in a group that has a REPEATED clause.

### 3.12.3 MULTIPLE PAGE Clause: Operation

1. The MULTIPLE PAGE clause enables a single report group to occupy any number of consecutive pages. It allows you to code NEXT PAGE on as many LINE clauses as you wish throughout the group (rather than just on the first). It also allows you to define as many relative lines as you wish, irrespective of the size of the page. Thus you can print very large tables using a single GENERATE.

2. When the group is a body group with MULTIPLE PAGE, no page fit-test is performed for the group as a whole. Instead, each report line is subjected to an individual page-fit test. A LINE clause with NEXT PAGE also causes a page advance to take place. Loosely speaking then, a multiple page group may begin at any point and cause a page advance at any point.

3. When a page advance is required, a PAGE FOOTING and PAGE HEADING group are printed as usual if defined. This is the only case where groups may interrupt another group. If the line that caused the page advance is relative, it is placed in the FIRST DETAIL position, disregarding the integer of its LINE clause. In the following layout, note how the "ROBINSON" group (in blue) spans three pages:
<table>
<thead>
<tr>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>JONES</td>
<td>ROBINSON</td>
</tr>
<tr>
<td>RUGBY</td>
<td>BADMINTON</td>
<td>BADMINTON</td>
</tr>
<tr>
<td>TENNIS</td>
<td></td>
<td>VOLLEYBALL</td>
</tr>
<tr>
<td>BASKETBALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FISHING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOCCER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TENNIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUGBY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRICKET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWIMMING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PING PONG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUDO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICE HOCKEY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| (4)        | (5)        | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>THOMSON</td>
<td>JOHNSON</td>
<td></td>
</tr>
<tr>
<td>JUDO</td>
<td>SNOOKER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
01 SPORTS-LIST TYPE DE MULTIPLE PAGE.
  03  LINE COL 1 PIC X(20) SOURCE W-SURNAME.
  03  LINE OCCURS 1 TO 100 VARYING R-SPORT
       COL 3 PIC X(20)   SOURCE W-SPORT (R-SPORT).

4. NO MULTIPLE PAGE can be used to prevent the lines in its scope from being split
   over a page boundary. If an OCCURS clause is also present, this rule applies
   separately to each occurrence. In the following example we want each set of
   address lines to remain together on the page:

01 PREVIOUS-ADDRESSES MULTIPLE PAGE.
  03 OCCURS 0 TO 20 TIMES DEPENDING ON NUMBER-OF-PREV-ADDRESSES
       VARYING R-ADDR-NO
       NO MULTIPLE PAGE.
  05 LINE OCCURS 0 TO 6 TIMES VARYING R-LINE-NO
       ABSENT WHEN W-ADDR-LINE (R-ADDR-NO R-LINE-NO) = SPACES.
  07 COL 1 PIC X(64)   SOURCE R-LINE (R-ADDR-NO R-LINE-NO).

5. At the 01 level, NO MULTIPLE PAGE merely documents the usual situation that a
   group cannot span pages.

3.12.4 Compatibility

The MULTIPLE PAGE clause is unique to new Report Writer.
3.13  **NEXTGROUP clause**

This clause is used to create additional vertical space after a group, without causing page overflow if there is no room on the page for the extra blank lines.

### Format

```
NEXT BODY DETAIL OR CONTROL HEADING GROUP IS

PLUS integer-1

integer-2 ON NEXT PAGE
```

### NEXTGROUP Clause: Coding Rules

1. This clause must be written only at the 01-level.

2. You can use the clause in any body group (DETAIL or CH/CF) and also in a REPORTHEADING (provided that a PAGE HEADING is present) as well as in a PAGE FOOTING (provided that a REPORT FOOTING is present).

3. If your report has no PAGE LIMIT clause, you can use only the `+ integer-1` form.

4. The optional words BODY and DE OR CH document the effect of the NEXT GROUP clause in the context of the type of group in which it is coded, but they do not have any actual effect on the clause. You can write NEXT BODY GROUP only in a DETAIL or CONTROL HEADING group, and NEXT DE OR CH GROUP only in a CONTROL FOOTING group.

5. DE OR CH may also be written: DETAIL OR CONTROL HEADING, CH OR DE, or CONTROL HEADING OR DETAIL.

### Effect of NEXTGROUP on Body Groups

1. Subject to certain constraints, the NEXT GROUP clause causes report writer to increase the value of LINE-COUNTER after all the lines in the body group have been produced, so that the next body group appears on a new page or after an additional vertical gap.

2. If you write NEXTGROUP `+ integer-1`, report writer will add integer-1 to LINE-COUNTER, but using the LAST CONTROL FOOTING value as a maximum limit. This creates an extra "gap" of that many lines between this group and the next, provided that there is room for both to appear on the same page. The following diagram shows how to use the clause to create more space after a CONTROL FOOTING:
<table>
<thead>
<tr>
<th>City</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Houston</td>
<td>1200</td>
</tr>
<tr>
<td>Dallas</td>
<td>500</td>
</tr>
<tr>
<td>Austin</td>
<td>1400</td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td>Total</td>
<td><strong>3100</strong></td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td>Miami</td>
<td>1500</td>
</tr>
<tr>
<td>Orlando</td>
<td>2300</td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td>Total</td>
<td><strong>5800</strong></td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td>Atlanta</td>
<td>1100</td>
</tr>
</tbody>
</table>

---

**01 CF FOR STATE NEXT GROUP PLUS 3.**

Notice that there is no "gap" after the TOTAL GEORGIA group, because it already extends to the LAST CONTROL FOOTING position.

You might suppose that you could also get extra blank lines by writing a blank LINE entry (containing no COLUMN entries or just a VALUE of SPACES) at the end of the group. But this is not a satisfactory replacement for the NEXT GROUP clause, because the blank LINE is treated as a part of the group and will affect the page-fit test. For example, if you had written

**03 LINE + 2.**

or

03 **LINE + 2.**
05 **COL 1 VALUE " "**.

at the end of the CF group instead of the NEXT GROUP clause in the above example, the TOTAL GEORGIA group would have appeared at the top of the next page, because its total depth would now be 5 lines instead of 3. Likewise, a LINE + 2 at the front of the DETAIL group would produce unwanted blank lines ahead of every DETAIL group.

3. The **NEXT GROUP integer-2** (absolute) form may force the next body group to be produced relative to the given line number. Report writer first examines the LINE-COUNTER. If this is less than integer-2 (in other words, if that position has not yet been reached), LINE-COUNTER is set equal to integer-2. Otherwise, report writer sets LINE-COUNTER equal to LAST CONTROL FOOTING to ensure that no more body groups will appear on this page and saves the value of integer-2 in the Saved Next Group Integer location. Before the next body group, if any, is produced, report writer sets LINE-COUNTER equal to the saved integer. It is inadvisable to use this form of the clause unless your next body group will always be a relative group.

4. If you write **NEXTGROUP NEXTPAGE** report writer will set LINE-COUNTER to the LAST CONTROL FOOTING value. This forces report writer to leave the rest of the page blank and begin the next body group, if any, on the next page. (LAST CONTROL FOOTING is the lowest position on which any body group can appear as described under 2.9 PAGE LIMIT clause).

To begin on a fresh page after a control break, you may place NEXT GROUP NEXT PAGE in the CONTROL FOOTING for that control level, as the following example shows:

---

**3.13 – NEXT GROUP**
If you write a NEXT GROUP clause in a CONTROL FOOTING group, report writer checks the level of the control break being processed before putting the clause into effect. If the level of the break is higher than the level of this group, the NEXT GROUP clause is ignored. This means that a CONTROL FOOTING’s NEXT GROUP clause can never affect the position of the next CONTROL FOOTING. As the following example shows, the CONTROL FOOTING FOR REPORT (the grand total) will not be forced onto a new page. To document this fact, the optional words DE OR CH are provided in the syntax.

If you should want the CONTROL FOOTING FOR REPORT to appear on a new page in this case, you should instead code ON NEXTPAGE in its first LINE clause.
6. You can code a **dummy** report group containing only a NEXT GROUP clause, as in the example below. When the group is processed, no output takes place, but LINE-COUNTER will be set equal to LAST CONTROL FOOTING solely because of the NEXT GROUP NEXT PAGE clause. This may be useful when you want to begin a new page on change of, say, BRANCH, but there is no subtotal that would normally justify coding a CONTROL FOOTING group. Just write:

```
01 CF FOR BRANCH NEXT GROUP NEXT PAGE.
```

To obtain a gap of three lines at any time, as an alternative to altering LINE-COUNTER directly, code:

```
01 THREE-LINE-GAP DE NEXT GROUP + 3.
```

and in your Procedure Division, write: **GENERATE THREE-LINE-GAP.**

### 3.13.4 Effect of NEXT GROUP on Non-Body Groups

1. You can write a NEXT GROUP clause in a **REPORT HEADING** group, in which case the group affected will be your first **PAGE HEADING** group, which immediately follows the REPORT HEADING. NEXT GROUP NEXT PAGE indicates that the REPORT HEADING is to be by itself on a separate page, rather than on the first page above the PAGE HEADING, but this will be assumed anyway if the REPORT HEADING will not fit above the first PAGE HEADING.

2. You can also write NEXT GROUP in a **PAGE FOOTING** group, in which case it affects the **REPORT FOOTING**. This form is never necessary, as the first LINE clause used in the REPORT FOOTING is a better way to handle this case.

### 3.13.5 Compatibility

- Only new Report Writer provides the following features:
- The optional words BODY and DE OR CH.
- Optional word ON before NEXT PAGE.
3.14 OCCURS clause

The OCCURS clause is used to show regular repetition of a field or a group of fields in a horizontal dimension, and a line or group of lines in a vertical dimension.

3.14.1 Format

```
OCCURS integer-1 TO integer-2 TIMES

DEPENDING ON identifier

STEP integer-3

WIDTH

DEPTH

LINES

COLUMNS

COLS
```

3.14.2 OCCURS Clause: Coding Rules

1. The OCCURS clause must not be used at the 01-level.

2. You may use the OCCURS clause at any of these levels:

   a. **Below the LINE level in an elementary entry.** Your field will be repeated horizontally the number of times indicated by the operand:

   ```
   03 LINE.  
   05 COL + 2 "NO" OCCURS 5. 
   
   NO NO NO NO NO
   ```

   b. **Below the LINE level in a group entry.** The whole group of fields will be repeated horizontally the number of times indicated by the operand:

   ```
   03 LINE.  
   04 OCCURS 3.  
   05 COL + 3 "DATE".  
   05 COL + 2 "AMOUNT". 
   
   DATE AMOUNT DATE AMOUNT DATE AMOUNT
   ```
c. **At the LINE level.** The line will be repeated vertically the number of times indicated by the operand.

```
03 LINE OCCURS 4.
  05 COL 1   "NAME".
  05 COL 21  "ADDRESS".
```
yielding:

```
NAME                ADDRESS
NAME                ADDRESS
NAME                ADDRESS
NAME                ADDRESS
```


d. **Below the 01-level but above the LINE level.** The whole group of lines will be repeated vertically the number of times indicated by the operand.

```
02 OCCURS 3.
  03 LINE + 2  COL 1   "NAME".
  03 LINE + 1  COL 1   "ADDRESS".
```

```
NAME          ADDRESS
NAME          ADDRESS
NAME          ADDRESS
```

Each group of LINE + clauses takes effect number of times given in the OCCURS clause.

These four levels are known as axes. Any fields in your group can be repeated in this way on any of the four axes, and you may have repetition on up to **four axes** by nesting the OCCURS clauses. Each of your report groups may have any number of OCCURS clauses, provided that the nesting limit of four is never exceeded at any one time.

3. You may use **STEP, WIDTH, or DEPTH** to specify the distance between the start of successive repetitions. Its operand indicates the number of columns or lines between the start of one item and the start of its successor:

```
05 COL 1  "PRICE" OCCURS 4  WIDTH 8 COLUMNS.
```

```
PRICE   PRICE   PRICE   PRICE
< - 8 ->

3.14 – OCCURS 121
Integer-3 of the STEP phrase must be at least as large as the field itself. WIDTH and DEPTH are alternative keywords with the same meaning as STEP. WIDTH may be used only in the horizontal direction and DEPTH may be used only in the vertical direction. You can write the documentary words COLUMNS or LINES after integer-3 to make the direction explicit. The STEP, WIDTH, or DEPTH phrase may be coded anywhere within the entry.

4. A STEP phrase is expected if your repeating entry is absolute. (It is absolute if you write COLUMN integer or LINE integer or, in the case of a group field, if the first COLUMN or LINE in the group is absolute.) However, if you omit STEP, a minimum value will be assumed, equal to the (maximum) size of the report field, with a cautionary message.

5. If your field is relative, the STEP phrase is optional. If you omit it the distance between repetitions is the physical size of the field, which may vary at run time if there are “<” PICTURE symbols or if the field is a group field containing entries with a PRESENT WHEN clause.

6. As the examples above illustrate (item 2 a through d), the VALUE clause is allowed within an entry subject to an OCCURS clause.

7. You must code the DEPENDING ON phrase if the number of occurrences is variable. The expression can be any COBOL numeric identifier or arithmetic expression that has an integer value. The data items used in your expression can come from anywhere in your program. They need not appear elsewhere as report fields and need not reside in the same “record” as the SOURCE fields.

8. Integer-1 in the format above and the TO keyword are used only in conjunction with DEPENDING ON. Integer-1 must be less than integer-2 and may be zero. If you omit integer-1 TO and write OCCURS integer-2 TIMES DEPENDING ON, this is taken to mean OCCURS 0 TO integer-2 TIMES DEPENDING ON.

3.14.3 OCCURS Clause: Operation

1. You should use the OCCURS clause for source-code reduction when you need to obtain automatic repetition either horizontally or vertically within a report group. The effect is as though you had coded every entry individually.

2. You may use a single-operand VALUE to place the same contents in each repetition.

3. If you use a SOURCE clause within an entry subject to OCCURS, the same SOURCE operand will be used to set up each occurrence. However, you may allow the effective SOURCE operand to vary by use of the VARYING clause and making use of the VARYING data-name as its subscript or as a term in the SOURCE operand. (See 3.27 VARYING clause for full details and examples.)

4. You may also use a multiple SOURCES (see 3.21 SOURCE clause) or multiple VALUES (see 3.26.4 Multiple VALUES) to place different contents in each repetition. The following sample layout shows both methods of combining both SOURCE and VALUE clauses with OCCURS:
SPORTS EQUIPMENT COMPANY: WAGES SUMMARY

<table>
<thead>
<tr>
<th>AREA</th>
<th>BASIC PAY</th>
<th>OVERTIME</th>
<th>COMMISSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORTH</td>
<td>$1,420,000</td>
<td>$600,000</td>
<td>$150,500</td>
</tr>
<tr>
<td>EAST</td>
<td>$2,100,000</td>
<td>$850,000</td>
<td>$220,000</td>
</tr>
<tr>
<td>SOUTH</td>
<td>$1,870,000</td>
<td>$1,000,000</td>
<td>$350,000</td>
</tr>
<tr>
<td>WEST</td>
<td>$970,000</td>
<td>$250,000</td>
<td>$110,000</td>
</tr>
</tbody>
</table>

which may be coded as follows:

```
01 SUMMARY-PAGE.
  03 LINE 1     COL CENTER 25 "SPORTS EQUIPMENT COMPANY: WAGES SUMMARY".
  03 LINE 2     COLS 1 11 26 39 VALUES "AREA" "BASIC PAY" "OVERTIME" "COMMISSION".
  03 LINE 4     OCCURS 4  STEP 1 VARYING AREA-NO.
      05 COL 1     VALUES "NORTH" "EAST" "SOUTH" "WEST".
      05 COLS RIGHT 19 33 47 PIC $$$,$$$,$$$ SOURCES BASIC-PAY (AREA-NO), OVERTIME (AREA-NO), COMM (AREA-NO).
```

Notice how the series of **row-heading** items (NORTH, EAST, etc.) is handled: the enclosing OCCURS at the LINE level simply steps through each item in the series in turn.

5. If a **PRESENT WHEN** clause is coded in the same entry as an OCCURS clause, the PRESENT WHEN applies separately to each occurrence (see 3.18.3). If the STEP phrase is **not** used, an entry that is **ABSENT** will not take up any space, as in the case below where non-negative values are to be skipped:

```
01 LOSS-MONTHS   TYPE DE.
  03 LINE OCCURS 12 VARYING R-MONTH PRESENT WHEN PROFIT (R-MONTH) < 0.
      05 COL 1     PIC X(9) SOURCE MONTH-NAME (R-MONTH).
      05 COL 12    PIC $(6)9 SOURCE PROFIT (R-MONTH).
```

The effect is similar in the horizontal direction:

```
LOSS-MAKING MONTHS:
MARCH  $12000
JUNE   $1000
OCTOBER $23000
```

```
01 LOSS-LINE     TYPE DE.
  03 LINE.
      05 COL + 3 OCCURS 12 VARYING R-MONTH PRESENT WHEN PROFIT (R-MONTH) < 0 PIC XXX SOURCE MONTH-NAME (R-MONTH).
  03 LINE.
      05 COL + 2 OCCURS 12 VARYING R-MONTH PRESENT WHEN PROFIT (R-MONTH) < 0 PIC ZZZ9 SOURCE PROFIT (R-MONTH).
```
However, if a STEP phrase is used, the spacing - in whichever direction - is regular and even entries that are ABSENT take up their usual space:

<table>
<thead>
<tr>
<th>JAN</th>
<th>FEB</th>
<th>MAR</th>
<th>APR</th>
<th>MAY</th>
<th>JUN</th>
<th>JUL</th>
<th>AUG</th>
<th>SEP</th>
<th>OCT</th>
<th>NOV</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120</td>
<td>230</td>
<td></td>
</tr>
</tbody>
</table>

which may be coded as follows:

```cobol
01  LOSS-MONTHS   TYPE DE.
   03  LINE.
      05  COL +3   OCCURS 12  VARYING R-MONTH
                  PIC XXX  SOURCE MONTH-NAME (R-MONTH).
   03  LINE.
      05  COL 2    OCCURS 12  VARYING R-MONTH   STEP 5
                  PRESENT WHEN PROFIT (R-MONTH) < 0
                  PIC ZZZ9  SOURCE PROFIT (R-MONTH).
```

### 3.14.4 Use of OCCURS...DEPENDING ON

1. If your report group has an OCCURS clause with a **DEPENDING ON** phrase, report writer will evaluate the associated identifier or expression at run time on each separate occasion when the report group is about to be generated. If its value is outside the range of integer-1 TO integer-2 in your OCCURS clause, this is **not an error** condition, and report writer will assume the maximum number, integer-2.

2. Now that the actual number of repeats is known, report writer creates **only that number** of occurrences, treating your clause as though you had written a fixed OCCURS clause with that number as the integer. **This is an important notion.** Here are some of its consequences:

   a. If your OCCURS ... DEPENDING is in the **horizontal** direction and is followed by an **absolute** COLUMN clause, the space not filled by excess occurrences will be **blank**:

   ```cobol
   05  COL 1   VALUE "ONE"  OCCURS 5 DEPENDING ON COUNT-1 STEP 5.
   05  COL 27  VALUE "END".
   
   If COUNT-1 contains 3, the effect is as though you had written:
   
   ```cobol
   05  COL 1   VALUE "ONE"  OCCURS 3   STEP 5.
   05  COL 27  VALUE "END".
   ```

   b. If your OCCURS ... DEPENDING is in the **horizontal** direction and is followed by a **relative** COLUMN, it is positioned relative to the last actual occurrence:

   ```cobol
   05  COL 1   VALUE "TWO"  OCCURS 5 DEPENDING ON (A + B) STEP 6.
   05  COL +2  VALUE "END".
   ```
If $A + B$ equals 3, the result is as though you had written:

```plaintext
05 COL 1 VALUE "TWO" OCCURS 3 STEP 6.
05 COL +2 VALUE "END".
```

| TWO  | TWO  | TWO END |

If your OCCURS ... DEPENDING is in the **vertical** direction and is followed by an **absolute** LINE, excess occurrences will be **blank**:

```plaintext
03 LINE 2 OCCURS 1 TO 4 DEPENDING ON COUNT-1 STEP 1.
05 COL 1 VALUE "ADDRESS LINE".
03 LINE 6.
05 COL 1 VALUE "ZIPCODE".
```

If COUNT-1 contains 2, the result is as though your OCCURS 1 TO 4 clause above were simply coded as OCCURS 2:

| ADDRESS LINE         | << line 2 |
| ADDRESS LINE         | << line 3 |
| ZIPCODE              | << line 6 |

If your OCCURS ... DEPENDING is in the **vertical** direction and is followed by a **relative** LINE, no line positions will be occupied by the unused occurrences, and the line that follows will be relative to the last actual occurrence:

```plaintext
03 LINE + 1 OCCURS 4 DEPENDING ON COUNT-1.
05 COL 1 VALUE "ADDRESS LINE".
03 LINE + 1.
05 COL 1 VALUE "ZIPCODE".
```

| ADDRESS LINE         | << line + 1 |
| ADDRESS LINE         | << line + 1 |
| ZIPCODE              | << line + 1 |

The **page-fit test** allows only for the occurrences that are **actually present**. So, if the group as a whole is relative, you will have a "best fit". For instance, if you **GENERATE** the above group and only three lines are available to it, the group will fit on the page.

3. You may code several entries with OCCURS ... DEPENDING in the same report line and nest horizontal and vertical repetitions to produce many varied and useful effects. The DEPENDING expression is re-evaluated each time the item is about to be output, so you may **alter** the size and shape of your report fields **dynamically** and automatically. You may also use OCCURS ... DEPENDING in a horizontal or vertical **group** entry.

3.14 – OCCURS

125
3.14.5 Compatibility

The use of the OCCURS clause in the report section, and the optional phrases, are unique to new Report Writer.
3.15 PAGE-COUNTER

This special register contains the number of the current page.

3.15.1 Format

```
│►► PAGE-COUNTER │ IN │ report-name │ OFF │
```

3.15.2 Uses of PAGE-COUNTER

1. PAGE-COUNTER contains the number of the current page. It is set to 1 by the INITIATE statement and, if your report has a PAGE LIMIT clause, is incremented by 1 on each page advance after the initial one. So its value is 1 on the first page, 2 on the second, and so on. The incrementing takes place between the production of your PAGE FOOTING group, if you defined one, and your PAGE HEADING group, if you defined one. So accessing PAGE-COUNTER in a USE BEFORE REPORTING Declarative section for a PAGE HEADING, for example, yields the value that applies to the new page. The TERMINATE statement leaves PAGE-COUNTER unchanged.

2. You may treat PAGE-COUNTER as a numeric location (implicit PICTURE S9(9) COMP SYNC) in any SOURCE expression or condition. For example, to print the page number anywhere in the page, write:

```
PIC ZZZ9 SOURCE IS PAGE-COUNTER.
```

You can change the value of PAGE-COUNTER at any time. Just treat it as any numeric data item (except that report writer defines it automatically). As an example, if you want to restart your page numbering from 1, after each control break, code

```
MOVE ZERO TO PAGE-COUNTER
```

in a Declarative SECTION for the CONTROL HEADING, so that when report writer next does a page advance it will increment it from zero to one.

3. Like all of the special registers except COLUMN-COUNTER, a PAGE-COUNTER is maintained separately for each report because all your reports are completely independent of each other. This also means that you must occasionally qualify PAGE-COUNTER if you have more than one report in the program. You do this by using report-name as a qualifier, as shown in the format at the head of this section.

```
You need a qualifier only if there is ambiguity, that is, in the main-line PROCEDURE DIVISION. In the Report Group Descriptions and in a USE BEFORE REPORTING directive section report writer assumes that you mean "IN current-report". However, you may still use a qualifier even in those cases, for instance when you want to access the PAGE-COUNTER of a different report. More information on Multiple Reports will be found later (see 5.1).
```
4. If your report contains a REPORT HEADING that is on a page by itself, the value of PAGE-COUNTER for the first page of details will be 2 since page 1 is occupied by the REPORT HEADING. If you want PAGE-COUNTER to be 1 on the first page of details, you will need to code

```
MOVE ZERO TO PAGE-COUNTER
```

after the INITIATE statement.

### 3.15.3 Compatibility

OS/VS and DOS/VS COBOL require PAGE-COUNTER to be qualified wherever it is used if your program contains more than one RD.
3.16 **PICTURE clause**

A PICTURE clause is used, as in basic COBOL, to indicate the size and format for each report field.

### 3.16.1 Format

\[
\begin{array}{c}
\text{►►} \quad \text{PICTURE} \quad \text{IS character-string} \quad \text{◄◄} \\
\text{PIC}
\end{array}
\]

### 3.16.2 PICTURE Clause: Coding Rules

1. All the available PICTURE symbols may be used, including a currency symbol defined by a `CURRENCY SIGN` phrase, provided they are consistent with a DISPLAY field, and the rules for combining them are exactly as for basic COBOL. Here are some examples:

   - **PIC ZZZ9.99** reproduces the field with up to five integral places, of which four leading zeros can be changed to a space, followed by a decimal point and two fractional places.
   
   - **PIC -$(5)9** outputs a "-" sign if the field is negative, followed by from 1 to 6 digits, with leading zeros changed to a space, and a currency symbol placed immediately before the first digit.
   
   - **PIC XXBXXBXX** outputs a six-character field, with spaces inserted between characters 2 and 3, and characters 4 and 5.

2. If the item is DBCS (Double-Byte Character Set), the PICTURE may contain only the symbols `G` and `B` (representing a DBCS space). However, a PICTURE clause is not required with a DBCS literal.

3. Report writer allows the additional left-shift PICTURE symbols "\(<\)" and "\(\)" to indicate that all or part of your field is variable-length. These symbols may be used only in the REPORT SECTION. You may place the "\(<\)" symbol anywhere within the PICTURE, in any number of places, provided that it is followed by one of the following symbols: "X", "A", "9", "Z", or floating (that is, repeated) ".". (In other words, it must appear before a symbol that represents "data", as opposed to an "editing" symbol.) The "\(\)" symbol may optionally be used to terminate its scope. The effect of these symbols is described below (see 3.16.4).

4. Report writer also allows general insertion characters, in the REPORT SECTION only. These are indicated by writing the characters to be inserted in "quotes" (or 'apostrophes) anywhere in the PICTURE, for example: **PIC 99"."99"."99**. These insertion characters do not take a repetition factor (so that PIC "."(5)X is invalid and must be written PIC "....."X).
5. You may **omit** the PICTURE clause in an entry that has a `VALUE` clause, whether **nonnumeric** (when a PICTURE X(n) is assumed) or (unsigned) **numeric** (when a PICTURE S9(n) DISPLAY is assumed.) (A **SYMBOLIC CHARACTER** is treated as a one byte nonnumeric.) A PICTURE is required if you use **ALL "literal"**, or a **figurative constant** such as **QUOTE**. In the case of a multiple `VALUE` or multiple-choice entry, there may be several "literals" but still **no** PICTURE is required.

6. If you use a **SOURCE SUM/COUNT** or **FUNCTION** clause, the PICTURE is necessary, even if you want to display the field in exactly the same format in which it is stored.

7. If the **OSVS** precompiler option is in effect, PICTURE symbol "A" may be used even with a literal that is not entirely alphabetic.

8. In common with the rest of the DATA DIVISION, PICTURE is allowed only in an elementary entry.

### 3.16.3 PICTURE Clause: Operation

1. In the REPORT SECTION, the PICTURE clause plays the same role as it does in other SECTIONs. The rest of this section and the next describe the extensions which are unique to the REPORT SECTION.

2. **General insertion characters** may used to reduce the number of entries to be coded. For example:

   | 05 COL  1 | VALUE "(". |
   | 05 COL +1 | PIC 99     | SOURCE W-TODAY-YY. |
   | 05 COL +1 | VALUE ".". | |
   | 05 COL +1 | PIC 99     | SOURCE W-TODAY-MM. |
   | 05 COL +1 | VALUE ".". | |
   | 05 COL +1 | PIC 99     | SOURCE W-TODAY-DD. |
   | 05 COL +1 | VALUE ")". | |

   may be reduced to:

   | 05 COL  1 | PIC "("99"."99"."99")" | SOURCE W-TODAY. |

   and:

   | 05 COL  1 | PIC "("99"."99"."99")" | SOURCE W-TODAY. |

   may be reduced to:

   | 05 COL  20 | PIC <9(5)>9 | SOURCE LETZTE-ZAHLUNG. |
   | 05 COL +1 | VALUE "DM". | |

   may be reduced to:

   | 05 COL  20 | PIC <9(5)>9"DM" | SOURCE LETZTE-ZAHLUNG. |

   and:

   | 05 COL 100 | VALUE "Page:". |
   | 05 COL +2 | PIC ZZ9 | SOURCE PAGE-COUNTER. |

   may be reduced to:

   | 05 COL 100 | PIC "Page: ZZ9" | SOURCE PAGE-COUNTER. |
Note that an insertion character specified in this way never has any semantic significance. Thus "." is never treated as a decimal point for alignment purposes.

3.16.4 Variable-Length Fields ("<" and ">" Symbols)

1. If you code the "<" symbol somewhere in your PICTURE in the REPORT SECTION, report writer takes it as referring to the following symbol and repetitions of that symbol, until the end of the PICTURE or a change of symbol is found. (The use of parenthesis as a shorthand for repetition does not count as a change of symbol.) The "<" symbol itself does not contribute to the length of the field. In the following case:

   \textbf{PIC 9<99,999.<99}

   Report writer will divide the field into the following parts:
   \begin{align*}
   9 & \quad \text{(1 character fixed)} \\
   <99 & \quad \text{(0 to 2 characters variable)} \\
   ,999. & \quad \text{(all fixed)} \\
   <99 & \quad \text{(0 to 2 characters variable)}
   \end{align*}

   The variable parts of your field are represented by those parts of your PICTURE that follow a "<" symbol, up to a change of symbol. To mark where each variable part ends, you may code a ">" symbol, resulting, in the above case, in:

   \textbf{PIC 9<99>,999.<99>}

   By convention, PIC \texttt{<Z(n)} is taken to mean the same as the more conventional PIC \texttt{<9(n)}, and PIC \texttt{<$(n)} is understood to mean the same as PIC \texttt{$<9(n-1)}.

2. When your field is output, report writer first stores the value to be output in a working area whose PICTURE is the same as the PICTURE you coded but without the "<" symbol(s). It then examines each part of your field that corresponds to a variable part of the PICTURE. If the symbol after the "<" is "X" or "A", that part is non-numeric and report writer will delete trailing spaces (that is, it will not advance COLUMN-COUNTER over them). If the symbol after the "<" is "9" or "Z" or "-", that part is numeric and report writer will delete leading zeros by left-shifting the contents of the rest of the field. After a decimal point, either explicit (.) or implicit (V) followed by a "<" symbol, it is trailing zeros that are deleted.

3. When characters are deleted, any characters that follow them are pulled to the left over them. This gives a free format effect to variable fields that have several parts. When the field has been output, the horizontal pointer (COLUMN-COUNTER), will point to the last character stored. If your next entry has an absolute COLUMN, you will have a variable number of spaces from the end of this field to the fixed starting point of the next. If, however, your next entry has a relative COLUMN (COL +), the number of spaces between the fields will be fixed, but the starting position of the next field will vary.

4. You may use the "<" and ">" symbols to split any part of your field into two fragments, one variable and one fixed, resulting in a minimum size and a maximum size. For example: PIC \texttt{XX<XXX> means from 2 to 5 non-numeric characters while PIC <999>9 means from 1 to 4 numeric characters (a form that is more useful than PIC \texttt{<9(4)} because a value of all zeros is reproduced as a zero).
5. Here are some examples of the "<" symbol:

a. If PICTURE $9999.99 gives you the result: $0000.50

then PICTURE $9<99.<99 will give you the result: $0.5

b. If PICTURE 99/99/99 gives you the result: 21/09/00

then PICTURE <99/<99/<99 will give you the result: 21/9/

c. If  

05 COL 1   PICTURE X(6)   SOURCE TITLE.
05 COL + 2  PICTURE X(20)  SOURCE SURNAME.
05 COL + 1  "::".

gives you the result: MR. SMITH

then  

05 COL 1   PICTURE <X(6)   SOURCE TITLE.
05 COL + 2  PICTURE <X(20) SOURCE SURNAME.
05 COL + 1  "::".

will give you the result: MR. SMITH

6. Special action is taken with the "," (comma) and "." (decimal point) symbols. If you write a "<" and a series of "9" symbols before ",", and the numeric value that corresponds to them is zero, then the "," will also be deleted. Also, if you write "<" after a "." (decimal point) and all the numeric positions after the decimal point are zero, then the decimal point will also be deleted. For example, if your field is described as:

PICTURE <99,<999,<999.<99

then values of 00002345.10 and 00000001.00 will be reproduced as:

2,345.1 and: 1 respectively

7. Any "," or "." encountered in a numeric field after a "," symbol turns off the effect of the ",". If you want its effect to persist across such an insertion character, you must turn it on again by coding another "<" symbol; for example: PIC 9,<999>,<999>.99.

3.16.5 Compatibility

1. Only new Report Writer allows the '<' and '>' symbols and general insertion characters in quotes or apostrophes. In all other aspects of the PICTURE clause, both implementations are compatible.

2. The general insertion character feature may be incompatible with Codasyl (and hence some future standard) if quotes or apostrophes in a PICTURE are ascribed some different significance.
3.17 **PRESENT AFTER clause**

This clause is similar to the **PRESENT WHEN clause** (see 3.18), except that it tests a condition arising **internally** within report writer's automatic control-break and page-advance processing, rather than evaluating a general COBOL condition.

3.17.1

<table>
<thead>
<tr>
<th>Format a</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRESENT</strong> AFTER NEW control-id OR PAGE</td>
</tr>
<tr>
<td><strong>ABSENT</strong> PAGE OR control-id</td>
</tr>
<tr>
<td>control-id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format b</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRESENT JUST AFTER NEW PAGE</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format c</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GROUP INDICATE</strong></td>
</tr>
</tbody>
</table>

3.17.2 **PRESENT/ABSENT AFTER Clause: Coding Rules**

1. If you specify the **PAGE** keyword, the RD for your report must have a PAGE LIMIT clause. If you specify a control-id, then it must be one of the controls listed in your **CONTROL** clause in the RD, except that REPORT or FINAL is always assumed to be present in the CONTROL clause.

2. You may code this clause at the group or elementary level and may nest clauses.

3. You may use format a in any body group (DETAIL or CH/CF), but, if you use the control-id option in a CONTROL HEADING or CONTROL FOOTING group, the control level you refer to must be higher than the control level of the report group in which the PRESENT AFTER clause is coded.

4. You may also use format a in a PAGE HEADING or PAGE FOOTING group, but only with the control-id option.

5. Format b (with the **JUST** phrase) can be used only in a **body group**.

6. Format c, the **GROUP INDICATE** clause, is provided for compatibility with current standards. Except for one minor but useful difference in its action with **OCCURS** (see below), it is equivalent to the clause:
PRESENT AFTER NEW PAGE OR lowest-control-id

where PAGE is present if the report has a PAGE LIMIT clause, and the control-id is present if the report has a CONTROLS clause.

For example, if the RD entry has the format:

```
RD ...
PAGE LIMIT 60 LINES
CONTROLS BRANCH-NO ZONE-ID.
```

the GROUP INDICATE clause is equivalent to:

PRESENT AFTER NEW PAGE OR ZONE-ID.

7. It is **not** advisable to refer to an item subject to PRESENT/ABSENT AFTER or GROUP INDICATE in a SUM clause. This is because, according to the ANS standard that applies if the option is in effect, summing takes place before the page-fit, so it is not always easy to predict whether the item to be summed will actually be present.

3.17.3 PRESENT/ABSENT AFTER Clause: Operation

1. The PRESENT AFTER clause operates in a way similar to a PRESENT WHEN clause except that our condition is set from within the report itself. **PRESENT AFTER NEW control-id** behaves like a clause of the form:

PRESENT WHEN this group has never yet been output
OR a control break has occurred at that level or above since the last time it was output

while PRESENT AFTER NEW PAGE behaves like a clause of the form:

PRESENT WHEN this group has never yet been output
OR a page advance has taken place since the last time it was output

To understand this clause fully you should therefore refer to the **PRESENT WHEN clause** (see 3.18).

2. If you code **PRESENT AFTER NEW control-id**, report writer will output the field (elementary or group field), provided that this is the first occasion this report group has been output since the start of the report or since the last control break at the level of control-id or above. For example, if you write PRESENT AFTER NEW BRANCH-NO, your report field will be produced at the beginning of the report and at the first GENERATE after each change of BRANCH-NO (or any higher control). Otherwise, the field is **ignored**, together with any subordinate entries, in exactly the same manner as with the PRESENT WHEN clause.

In the following example, the field YEAR-NO is to be output the first time only and then whenever it has changed, while SEASON-NO is to be output the first time and then whenever it or YEAR-NO has changed.
### Table: Past Season-Names

<table>
<thead>
<tr>
<th>Year Season</th>
<th>Name</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>1991 Winter</td>
<td>CODER T.J.</td>
<td>BADMINTON</td>
</tr>
<tr>
<td></td>
<td>HACKER S.</td>
<td>RUGBY</td>
</tr>
<tr>
<td></td>
<td>MANAGER D.P.</td>
<td>SWIMMING</td>
</tr>
<tr>
<td>SPRING</td>
<td>EDITOR F.</td>
<td>RIDING</td>
</tr>
<tr>
<td></td>
<td>LOAD V.</td>
<td>CRICKET</td>
</tr>
<tr>
<td></td>
<td>TESTER S.</td>
<td>GOLF</td>
</tr>
<tr>
<td>SUMMER</td>
<td>DUMP J.</td>
<td>TENNIS</td>
</tr>
<tr>
<td>1992 SUMMER</td>
<td>ANALYST R.</td>
<td>SWIMMING</td>
</tr>
</tbody>
</table>

---

**RD** ... **CONTROLS ARE** YEAR-NO, SEASON-NO.

**01 NEW-MEMBER** TYPE DE LINE + 1.
**05 COL 2 PIC 9(4) SOURCE YEAR-NO PRESENT AFTER NEW YEAR-NO.**
**05 COL 7 PIC X(6) SOURCE WS-SEASON (SEASON-NO) PRESENT AFTER NEW SEASON-NO.**

3. As is usual with controls, a higher control break implies a control break at all the lower levels. Thus if you code PRESENT AFTER MONTH when YEAR and MONTH are the controls, the field will be PRESENT also after a change of YEAR, for JAN 1992 is certainly different from JAN 1991!

4. If you code **PRESENT AFTER NEW PAGE**, report writer will produce the entry if this is the first occasion this group has been produced since the start of the report or since the last page advance. Otherwise, again, the field is not output.

As the following example shows, it may cause the entry to be produced at any position within the page provided the group containing it has not yet appeared on the page. If you want the entry to appear only if this group is also the first body group of the page, you should instead use the form: **PRESENT JUST AFTER NEW PAGE**. In the example that follows, this would prevent the subheading UNFILLED ORDERS from being printed in the body of the page.

In the following example, one group has a subheading for unfilled order details and we want this subheading to appear only the first time that the group is printed on the page:
## SPORTS RETAIL STOCK SUMMARY

<table>
<thead>
<tr>
<th>ITEM</th>
<th>QTY IN STOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOLF 5 IRON</td>
<td>3</td>
</tr>
<tr>
<td>SQUASH RACKET</td>
<td>20</td>
</tr>
<tr>
<td>TENNIS SOCKS</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNFILLED ORDERS: NAME</th>
<th>QTY</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMKINS</td>
<td>10</td>
<td>JUL 01</td>
</tr>
<tr>
<td>MABBOT</td>
<td>6</td>
<td>AUG 20</td>
</tr>
<tr>
<td>SWIMMING TRUNKS</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>RIDING HAT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADRIAN</td>
<td>1</td>
<td>JUN 20</td>
</tr>
</tbody>
</table>

5. If you write `PAGE OR control-id` (this order can be reversed), the field will be produced if either or both conditions arise. For instance, you might want the YEAR-NO and SEASON fields in the examples above (see above) to be printed again at the start of a new page even though there may have been no change. In this case, you must write:

```cobol
05 ... PRESENT AFTER NEW YEAR-NO OR PAGE
05 ... PRESENT AFTER NEW SEASON-NO OR PAGE
```

6. There may be a field, line, etc. that you would like produced the first time only. To accomplish this, use `PRESENT AFTER NEW REPORT` (REPORT or FINAL is the highest control level and is always assumed even if not coded in the CONTROL clause.) In the next example, it is in the PAGE HEADING:

```cobol
** START OF REPORT **
```

To anchor the vertical starting line of this relative group, you may include a HEADING sub-clause in your RD.
7. If you write **ABSENT** instead of **PRESENT**, the clause will have exactly the opposite effect. In other words, the field will be produced whenever it would have been ignored and vice versa. In the next example, we use it to produce a "(CONTINUED)" message in our PAGE HEADING. It will appear on every page except the first page after each new control value. (This message is also useful in CONTROL HEADING groups.)

<table>
<thead>
<tr>
<th>PAGE 1</th>
<th>AREA 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE 2</td>
<td>AREA 1</td>
</tr>
<tr>
<td>PAGE 3</td>
<td>AREA 2</td>
</tr>
</tbody>
</table>

01 TYPE PH  LINE 60.
  05 COL 1   "PAGE:".
  05 COL + 2 PIC <999  SOURCE PAGE-COUNTER.
  05 COL 90   "AREA:".
  05 COL 96   PIC 9   SOURCE AREA-NO.
  05 COL 100  "(CONTINUED)" ABSENT AFTER NEW AREA-NO.

8. If there is an OCCURS clause, or a multiple LINES or COLUMNS clause, in the same entry, the PRESENT AFTER applies to the entire set of repetitions, so the occurrences are either all present or all absent. (Compare **GROUP INDICATE** immediately below.)

9. **GROUP INDICATE** behaves differently from the equivalent PRESENT AFTER clause if it is subject to OCCURS. As soon as the first occurrence of the GROUP INDICATE item has been output, the GROUP INDICATE is switched off. Hence the GROUP INDICATE clause only enables one entry to be output (the leftmost of topmost, depending on the axis), whereas PRESENT AFTER affects each occurrence equally and is switched off only when the whole table has been output. This fact can be put to practical use if you want some text to appear with the first entry only:

```plaintext
ADDRESS: 345 MERMAID STREET
          RYE
          SUSSEX
          ENGLAND
```

01 ADDRESS-GROUP  TYPE DE.
  03 LINE  OCCURS 10 TIMES  VARYING R-LINE
  ABSENT WHEN ADDR-LINE (R-LINE) = SPACES.
  05 COL 1   VALUE "ADDRESS:"  GROUP INDICATE.
  05 COL 11  PIC X(32)  SOURCE ADDR-LINE (R-LINE).

(Note that if you want "ADDRESS:" to be printed again each time ADDRESS-GROUP is GENERATED you need to ensure that a control break occurs on each GENERATE of ADDRESS-GROUP.) Using **GROUP INDICATE** in this example is better than coding **PRESENT WHEN R-LINE = 1** since the latter will not work here if the first occurrence of ADDR-LINE might contain spaces.)
10. If you use the PRESENT AFTER clause at the 01-level of a DETAIL group:

```
01 DETAIL-GROUP TYPE DE PRESENT AFTER NEW BRANCH-NO.
```

and you require this group to appear as soon as there is a change to the control
BRANCH-NO, your program should GENERATE this group before every other
GENERATE for the report. In spite of any appearance to the contrary, you may
rest assured that the PRESENT AFTER clause will not make a group appear if your
program does not GENERATE it.

3.17.4 Compatibility

1. Only new Report Writer provides the PRESENT/ABSENT AFTER clause.

2. New Report Writer allows GROUP INDICATE to appear at any level - not only at
   elementary level.
3.18 **PRESENT WHEN clause**

This clause places a general COBOL condition on any report entry. By evaluating the condition, report writer determines whether your entry is to be **output** as normal or **skipped**.

### 3.18.1

**Format a**

```
PRESENT WHEN condition UNLESS CONTROL IS control-id
```

**Format b**

```
PRESENT WHEN OTHER
```

### 3.18.2 **PRESENT/ABSENT WHEN Clause: Coding Rules**

1. The condition may be **any valid COBOL conditional expression**. You are not restricted to simple conditions. For example, the following compound condition is quite acceptable:

```
PRESENT WHEN SALARY > 20000 AND (AGE < 20 OR MARITAL-STATUS = "M" OR "D")
```

   Each of the data items you use in your condition may come from any COBOL SECTION (but see the note **Use of Total Fields** in 3.23.4 if you use SUM fields from the REPORT SECTION itself). You may optionally place parentheses "(   )" around the whole conditional expression for clarity. The symbols ">=" (greater than or equal to) and "<=" (less than or equal to) are allowed. For more information, you should refer to your COBOL language reference under **conditional expressions**, for a list of the many varied ways of forming COBOL conditions.

2. The **CONTROL IS control-id** form is allowed only if the current group is a **multiple CONTROL FOOTING**, that is, one with a TYPE CONTROL FOOTING clause with **more than one** control-id operand, or TYPE CONTROL FOOTING FOR ALL.

3. The keyword **PRESENT** is implied if omitted. Experience has shown that your code will be clearer if you include the PRESENT keyword in the **single** form of the clause but omit it in the **multiple-choice** form described below and in the following example:
4. The keyword **ABSENT** gives the clause the exact opposite meaning from the **same** clause with **PRESENT**: **ABSENT WHEN** condition is equivalent to **PRESENT WHEN NOT** condition. So you may find it more expressive to write:

```
ABSENT WHEN AGE < 21 AND LOCN = "NY"
```

rather than:

```
PRESENT WHEN AGE NOT < 21 OR LOCN NOT = "NY".
```

Whenever we refer to the **PRESENT WHEN** clause, you can assume that this term includes the **ABSENT WHEN** form.

5. **PRESENT UNLESS** is an older syntax, synonymous with **ABSENT WHEN**.

6. You may write these clauses in **any** report group entry other than the **RD** itself. An entry that has a **PRESENT WHEN** clause is often referred to as a **conditional** entry. The same field may be subject to any number of **PRESENT WHEN** clauses at any number of different levels. The following sample group has one of the clauses in all the possible positions:

```
01 TYPE PH  ABSENT WHEN PREPRINTED = "Y".  *>group level
 03 LINE 2.
   05 COL 20 "HACKNEY BEAUTY SUPPLIES".
   05 COL 60 "END OF YEAR"
   03 PRESENT WHEN THIS-MONTH = 12.  *>elementary level
   04 LINE  "AMOUNT             DATE".
   04 LINE  PRESENT WHEN LAST-YEAR = "Y".  *>LINE level
   05 COL 1 "LAST YR           LAST YR".
   05 ABSENT WHEN YEAR-NO = ZERO.  *>group of columns
   06 COL 50 "YEAR:".
   06 COL +2 PIC 9999 SOURCE YEAR-NO.
```

7. If you nest more than one **PRESENT WHEN** clause by writing one clause in a higher entry and another in a lower entry beneath it, as in the preceding example, the **outer** condition takes precedence over the **inner** condition. It is up to you to ensure that both conditions can be true at the same time. Report writer will not check for contradictory combinations of conditions such as:

```
03 LINE 1  PRESENT WHEN A = 1.
   05 COL 5 "X"  PRESENT WHEN A = 2.
```

Here, if A is not 1, the whole LINE will not be output and that is an end to the matter: the test for A = 2 is short-circuit ed and the "X" will never be output.
3.18.3 PRESENT/ABSENT WHEN Clause: Operation

1. Report writer tests the condition operand of your PRESENT WHEN clause each time it begins the processing of the report group. If the condition is true, the field is output normally. If the condition is false, the entry is ignored. (For ABSENT WHEN, this rule applies vice versa.) If the entry is a group entry, all the entries beneath it, including all its elementary entries, are also ignored. When report writer "ignores" your entry, it treats the entry, for that instant only, as though you had not coded it. Imagine the entry drawn in a rectangle of paper and imagine that you have a blank card that you can place over the rectangle to "mask" it out:

```
03  LINE 3.
  05  COL 1  PIC X(6)  SOURCE CITY.
  05  COL 8  PIC XXX  SOURCE REGN.
  05  COL 13 "POLICY NO.:".
  05  COL 25 PIC X(20)  SOURCE POLICY-NO.
  05  COL 50 PIC X(20)  SOURCE NAME.
```

Remember, when the PRESENT WHEN is not on an elementary entry, your masking "card" must be large enough to cover all the lower-level entries down to elementary level, as far as the next entry at the same level or higher. Now, to find out what will be produced when an entry is ABSENT, just cover the entry with the mask, so that it disappears from view. The remaining entries are what report writer "sees" at that instant, and therefore what it will output. For example, suppose that the field CITY-FLAG is 1 and INS-FLAG is zero in the example above. What will report writer produce? Just move the mask over the INS-FLAG group entry, as shown above, and this is what results:

```
03  LINE 3.
  05  COL 1  PIC X(6)  SOURCE CITY.
  05  COL 8  PIC XXX  SOURCE REGN.
  05  COL 50 PIC X(20)  SOURCE NAME.
```

Keeping this simple principle in mind, consider the practical applications described in the following paragraphs:

2. To "blank out" a field with an absolute COLUMN, place your PRESENT WHEN on the absolute COLUMN entry and follow it with another absolute COLUMN entry:

```
03  LINE 4.
  05  COL 4  PIC ZZZ9  SOURCE HOUSE-NUMBER.
  05  COL 10 PIC X(20)  SOURCE STREET-NAME.
```
If HOUSE-NO-FLAG is not 1, the entry in column 4 disappears. Because the following column is in an absolute position, the first entry is blank:

<table>
<thead>
<tr>
<th>EMPTY</th>
<th>22 LONDON ROAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DARK LANE</td>
</tr>
</tbody>
</table>

↑ no house number

This method also enables you to blank out an absolute LINE entry from a set of absolute LINE entries, leaving a gap in the printed output.

3. To insert a conditional field into the report line so that it displaces the fields that follow, simply make the entries that follow relative:

<table>
<thead>
<tr>
<th>01 AMOUNT-LINE</th>
<th>TYPE DETAIL</th>
<th>LINE + 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 COL 13 VALUE &quot;DEBIT &quot; PRESENT WHEN AMOUNT &lt; ZERO.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 COL +1 PIC 9&lt;9(6) SOURCE AMOUNT.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When triggered by the PRESENT WHEN, the field in COLUMN 13 displaces the relative field that follows rightwards.

The word **DEBIT** displaces the next field to the right because the next field has a relative COLUMN clause (COL +1). Place your "mask card" over the **DEBIT** entry and you will see what occurs when its group is not produced: the AMOUNT field is the first and only field to be produced, and its COLUMN number is +1. It therefore appears in column 1.

Similarly, the HOUSE-NUMBER field in the previous paragraph could be treated as a conditional insertion item by revising the code as follows. (The "<" symbol eliminates the gap after short HOUSE-NUMBERs.)

| 03 LINE 4. |
| 05 COL 2. |
| *COL 4 can be coded here instead of COL +2: |

| 05 COL +2 PIC <9999 SOURCE HOUSE-NUMBER PRESENT WHEN HOUSE-NO-FLAG = 1. |
| 05 COL +2 PIC X(20) SOURCE STREET-NAME. |

This time, if **HOUSE-NO-FLAG** is not 1, there is no "gap" because the item that follows is relative.
22 LONDON ROAD

DARK LANE

↑ no house number, but this time no gap

4. To place one of a series of alternative entries into the same column positions, code several entries, each with a PRESENT WHEN clause.

Take care that the conditions are mutually-exclusive so that none of the fields can overlap:

<table>
<thead>
<tr>
<th>COL</th>
<th>PIC</th>
<th>SOURCE</th>
<th>PRESENT WHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X(16)</td>
<td>Surname</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>Minor</td>
<td>AGE &lt; 16.</td>
</tr>
<tr>
<td>20</td>
<td>ZZ9</td>
<td>Age</td>
<td>AGE &gt; 15 AND &lt; 65.</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>Old Timer</td>
<td>AGE &gt; 64.</td>
</tr>
</tbody>
</table>

GOLIGHTLY
THOMPSON  MINOR
PREWITT OLD TIMER

Notice that the conditional fields need not all start in the same column and that the column numbers need not be strictly in sequence, provided that they are in sequence when we mask out the entries that are ABSENT.

If all your entries start in the same column and are all literals or identifiers with the same PICTURE, you will find it more convenient to use a multiple-choice entry. (See 3.18.5 The Multiple-Choice Form, below.)

You can also use the technique shown here to choose one LINE from a set of alternative absolute LINE entries.

5. To string out (concatenate) a number of conditional fields along the report line, where any combination could be present, use a series of relative conditional entries:

<table>
<thead>
<tr>
<th>NAME</th>
<th>... SPORTS PLAYED ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>JENKINS</td>
<td>RUGBY TENNIS GOLF SWIMMING</td>
</tr>
<tr>
<td>LLOYD</td>
<td>GOLF</td>
</tr>
<tr>
<td>BURKE</td>
<td>TENNIS SWIMMING</td>
</tr>
</tbody>
</table>

01 TYPE PH LINE 1 " NAME ... SPORTS PLAYED ... ".
01 SPORTS-LINE TYPE DE LINE + 1.
05 COL 1 PIC X(12) SOURCE NAME.
05 COL 14.
05 COL + 2 VALUE "RUGBY" PRESENT WHEN RUGBY-FLAG = "Y".
05 COL + 2 VALUE "TENNIS" PRESENT WHEN TENNIS-FLAG = "Y".
05 COL + 2 VALUE "GOLF" PRESENT WHEN GOLF-FLAG = "Y".
05 COL + 2 VALUE "SWIMMING" PRESENT WHEN SWIMMING-FLAG = "Y".
The dummy entry specifying **COL 14** gives you an anchor point at which to begin. It ensures that the first sport-field will appear in column 16.

If you include many optional fields using relative COLUMNS, you may run the risk of causing line overflow if a large number happen to be present. If you want report writer to ‘wrap the data round’ on to a continuation line, you must code a **WRAP clause** (see 3.28). Otherwise, it will truncate the line and create a run time line overflow message.

A similar problem might arise if you define an entry in an **absolute** column following your series of optional entries. If there were too many entries present, report writer might signal a run time column overlap error. If there is a risk of line overflow or column overlap, you must do some extra processing before you GENERATE the line, to make certain that no more than the maximum number of fields will be present.

6. To print one or more of a series of conditional **lines** in a report group, use **relative LINE clauses** and code your PRESENT WHEN clauses at the LINE level or above:

```cobol
01  MEMBER-RECORD           TYPE DE.
  03  LINE + 2.
     05  COL 1           "MEMBER:"   ... etc ...
     03  PRESENT WHEN FAMILY-MEM-FLAG = "Y".
     04  LINE + 1.
     05  COL RIGHT 30    "AGE OF SPOUSE:".
     05  COL 34          PIC <99  SOURCE SPOUSE-AGE.
     04  LINE + 1         ABSENT WHEN NO-CHILDREN = ZERO.
     05  COL RIGHT 30    "NO. OF CHILDREN:".
     05  COL 34          PIC 9       SOURCE NO-CHILDREN.
     03  LINE + 1.
     05  COL RIGHT 30    "BASIC SUBSCRIPTION:".
     05  COL 34          PIC $9<99   SOURCE BASIC-SUB.
     03  LINE + 1          PRESENT WHEN GOLF-FLAG = "Y".
     05  COL RIGHT 30    "GOLF SUBSCRIPTION:".
     05  COL 34          PIC $9<99   SOURCE GOLF-SUB.
```

In this example, the lines showing **AGE OF SPOUSE** and **NO. OF CHILDREN** are not to be produced unless there is a “family membership” condition, and the line showing **GOLF SUBSCRIPTION** is not to appear unless the member plays golf. Here is some suitable report writer code:
Because report writer only "sees" the lines that have not been "masked out", it performs a sophisticated page-fit test. Report writer will test the availability of only the number of lines that will actually be output. If THOMPSON in the above example begins on line 59 of a 60-line page, his record will appear on that page because at that particular instant the DETAIL group is actually only 2 lines in depth.

The form CONTROL IS control-id is a special condition that can be used only in a PRESENT WHEN clause within a multiple CONTROL FOOTING. It is true if the level of the CONTROL FOOTING currently being generated is that of control-id. For example, if the TYPE clause is:

```
01 TYPE CF FOR COUNTY, CITY, STREET.
   you may introduce any amount of variation for each level by coding, for instance:
```

```
05 PRESENT WHEN CONTROL IS CITY.
   07 COL 1       "Name of city:".
   07 COL +2     PIC X(20)   SOURCE CITY.
```

7. If PRESENT WHEN is used in the same entry as an OCCURS clause or a multiple SOURCES or VALUES, the PRESENT WHEN applies to each occurrence individually.

3.18.4 Effect of PRESENT WHEN on SUM

The general principles of the PRESENT (or ABSENT) WHEN clause apply when you use it with a SUM clause or with an item that is totalled. Report writer acts according to certain vital principles, as follows:

1. Only those SUM operands that were PRESENT when they were processed will be added to a total field. An item will not be added if it is subject to a PRESENT WHEN and the condition is false. This means that you may total a series of "optional" fields and obtain a total that is true in the sense that only the fields that are processed appear in the total field. (The word processed includes the case of unprintable fields - those having no COLUMN clause - which do not appear in the report.)

The following example illustrates this principle:
SPORTS CLUB MEMBERSHIP PAYMENT NOTICE

NAME: T.S. ANALYST
BASIC SUBSCRIPTION: $25
PLAYER’S SUBSCRIPTION: $20
SQUASH SUPPLEMENT: $40
TOTAL DUE PLEASE: $85

NAME: A.J. CODER
INITIATION FEE: $40
BASIC SUBSCRIPTION: $25
PLAYER’S SUBSCRIPTION: $20
GOLF SUPPLEMENT: $150
SQUASH SUPPLEMENT: $40
TOTAL DUE PLEASE: $275

01 PAYMENT-NOTICE TYPE DE.

... etc ...

03 LINE PRESENT WHEN NEW-MEMBER-FLAG = "Y".
  05 COL 4 VALUE "INITIATION FEE".
  05 R-JOIN COL 31 PIC $$$9 SOURCE WS-JOIN-FEE.

03 LINE.
  05 COL 4 VALUE "BASIC SUBSCRIPTION".
  05 R-BASIC COL 31 PIC $$$9 SOURCE WS-BASIC-SUB.

03 LINE PRESENT WHEN PLAYER-FLAG = "Y".
  05 COL 4 VALUE "PLAYER’S SUBSCRIPTION".
  05 R-PLAYER COL 31 PIC $$$9 SOURCE WS-PLAYER-SUB.

... etc ...

03 LINE.
  05 COL 11 VALUE "TOTAL DUE PLEASE".
  *SUM clause shows total of only the fields that appear in report
  05 COL 30 PIC $$$.9 SUM OF R-JOIN, R-BASIC, R-PLAYER, R-GOLF, R-SQUASH.

2. If you write a SUM clause and a PRESENT WHEN together in the same entry, the total will not appear in the report if the entry is not present. But neither will it be cleared to zero, because no total is cleared until it has been output. Values will continue to be accumulated in the total field even though the SUM entry may not always be present. In other words, the PRESENT WHEN in the SUM entry can delay the generating and clearing of your total field but it has no effect on the totalling itself.

These facts may be utilized if you wish to generate a number of fields repeatedly but delay the appearance of their cross-foot total. In the following example, we have a variable number (up to 31) of fields, but only up to five will fit on a line. The total also occupies a field in the line. The following solution takes advantage of the fact that if a DEPENDING ON operand is higher than the maximum, the maximum (in this case 5) is assumed. (This problem can also be resolved now using the WRAP clause.)
To accumulate a series of totals split up by some category, that is into separate "pigeonholes" or "buckets", you may use a series of unprintable SUM fields partitioned by a series of mutually exclusive (and exhaustive) PRESENT WHEN condition clauses. Suppose that you wish to print "Sales of Ice Cream" and you want to total separately the sales of Vanilla, Strawberry and Chocolate flavors.

This is done by splitting up the sales entry invisibly into vanilla, strawberry and chocolate sales, as though we were actually splitting it thus at the place where it is printed. We then SUM each unprintable entry to form three separate totals:
ICE CREAM SALES      YEAR 1992

<table>
<thead>
<tr>
<th>DATE</th>
<th>FLAVOR</th>
<th>SALES AMOUNT ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN 01</td>
<td>VANILLA</td>
<td>1000</td>
</tr>
<tr>
<td>FEB 02</td>
<td>STRAWBERRY</td>
<td>11000</td>
</tr>
<tr>
<td>FEB 28</td>
<td>CHOCOLATE</td>
<td>5500</td>
</tr>
<tr>
<td>MAR 12</td>
<td>CHOCOLATE</td>
<td>4500</td>
</tr>
<tr>
<td>APR 03</td>
<td>VANILLA</td>
<td>6000</td>
</tr>
</tbody>
</table>

SUMMARY OF TOTALS BY FLAVOR:

<table>
<thead>
<tr>
<th>FLAVOR</th>
<th>TOTAL ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VANILLA</td>
<td>32000</td>
</tr>
<tr>
<td>STRAWBERRY</td>
<td>12500</td>
</tr>
<tr>
<td>CHOCOLATE</td>
<td>15000</td>
</tr>
</tbody>
</table>

If you have a larger, or variable, number of categories, it will be easier to
defined repeating values and totals using OCCURS. Suppose that the flavors
VANILLA, STRAWBERRY etc. are held in a table W-FLAVOR OCCURS 20. The
following code could now replace the corresponding entries used above:

05 R-SALE OCCURS 20 TIMES VARYING R-FLAVOR-NO PIC 9(7)
   PRESENT WHEN R-FLAVOR = W-FLAVOR (R-FLAVOR-NO).

   ... LINE OCCURS 20 VARYING R-FLAVOR-NO.
   05 COL 18 PIC X(16) SOURCE W-FLAVOR (R-FLAVOR-NO).
   05 COL RIGHT 47 PIC Z(7)9 SUM R-SALE.
3.18.5 The Multiple-Choice Form

If you need to specify a series of alternative contents for a particular elementary field, you can do it with a single entry, instead of several separate entries containing `PRESENT WHEN` clauses. You code the entry with several `SOURCE`, `VALUE`, or `FUNCTION` clauses, each followed immediately by a `PRESENT WHEN` clause. One of the `PRESENT WHEN` clauses, and only one, may be of the form `PRESENT WHEN OTHER`. It is normally placed last in the list (although this is not syntactically required).

Report writer scans your `WHEN` conditions, starting with the first in the order of coding, until it finds a condition that is true. It then uses the `SOURCE`, `VALUE`, or `FUNCTION` clause preceding that `PRESENT WHEN` clause and ignores all the remaining `PRESENT WHEN` and `SOURCE`, `VALUE` or `FUNCTION` clauses. So your conditions need not be mutually exclusive: the testing is on a "first-come-first-served" basis. Here is an example:

```
05 COL 23 PIC ZZZ9 SOURCE INCOME WHEN TYP-IND = "A"
   SOURCE TAX WHEN TYP-IND ALPHABETIC
   SOURCE INCOME - TAX WHEN TYP-IND = "1" OR "2".
```

The contents of `TAX` will be produced when `TYP-IND` is between "B" and "Z" inclusive.

If your conditions do not cover all the possibilities, you have three choices:

a. Put `WHEN OTHER` instead of `WHEN` condition against the entry you would like to act as the catch-all, default, or wastebasket:

```
05 COL 25 VALUE "TOO BIG" WHEN AMOUNT > 99999
   VALUE "TOO SMALL" WHEN AMOUNT < 100
   VALUE "JUST RIGHT" WHEN OTHER.
```

b. Code an extra "choice" using `WHEN OTHER` to indicate an error:

```
05 COL 100 VALUE "LONDON" WHEN CITY-CODE = "LN"
   VALUE "BRISTOL" WHEN CITY-CODE = "BR"
   ...
   VALUE "UNKNOWN CITY" WHEN OTHER.
```
c. Leave it as it is. If there is a case not covered by your multiple-choice entry, the whole field will then simply be ABSENT. For example:

```
05 COL 1 VALUE "DEAR".
05 COL +2 VALUE "MR" WHEN TITLE-CODE = 1
    VALUE "MRS" WHEN TITLE-CODE = 2
    VALUE "MISS" WHEN TITLE-CODE = 3.
05 COL +2 PIC X(20) SOURCE SURNAME.
```

If **SMITH** has TITLE-CODE equal to 0, this will result in:

```
DEAR SMITH
```

4. If you specify only VALUE literal clauses without a PICTURE clause, they may be of different sizes, as in the preceding example. The actual size of the field produced will be that of the chosen value. Since SURNAME above has a relative COLUMN, you will obtain:

```
DEAR MR SMITH
or DEAR MRS SMITH
or DEAR MISS SMITH
```

5. You cannot repeat the PICTURE clause in an entry (nor any of the other clauses other than SOURCE, VALUE, and FUNCTION), so if your choices need different PICTURES you must usually code a series of separate entries. However, you may still be able to choose a PICTURE that suits each of the choices, such as PIC ZZZ9 to cover both PIC ZZZ9 and PIC ZZ9, and thus still be able to use a single entry with a multiple-choice PRESENT WHEN clause.

6. You can form a SUM of a multiple-choice entry. Report writer will select the correct choice (if any) before adding it to the total:

```
05 PAYMENT COL 33 PIC ZZZZ9
  AMOUNT - TAX WHEN CAT = 1
  AMOUNT - TAX - CALIFORNIA-TAX WHEN CAT = 2
  AMOUNT - EXPORT-TAX WHEN CAT = 3.

05 ... SUM OF PAYMENT ...
```

This adds only the one correct choice each time.

7. You can use the **CONTROL IS control-id** form of condition anywhere in a multiple-choice entry. For example:

```
01 TYPE CF FOR STATE COUNTY CITY.
 03 LINE + 2.
 05 COL 1 VALUE "Totals for".
 05 COL + 2 VALUE "state" WHEN CONTROL IS STATE
    VALUE "county" WHEN CONTROL IS COUNTY
    VALUE "city" WHEN OTHER.
```
8. You can use multiple **SOURCE** and **VALUE** clauses within your multiple-choice entry. For example:

```
05 COLS 23 43 64 PIC ZZZ,ZZ9.99
SOURCES  VAL-1 VAL-2 VAL-3     WHEN TYP = "D"
SOURCES (- VAL-1) (- VAL-2) (- VAL-3)  WHEN OTHER.
```

9. You cannot place a multiple-choice entry at the same level as a LINE clause. The construct:

```
03 LINE 1   COL 1   VALUE "WHITE" WHEN SHADE = 0
            "BLACK" WHEN SHADE = 1.
```

is therefore illegal and must be replaced by:

```
03 LINE 1.
  05 COL 1   VALUE "WHITE" WHEN SHADE = 0
             "BLACK" WHEN SHADE = 1.
```

### 3.18.6 Compatibility

Only new Report Writer provides the PRESENT / ABSENT WHEN clause.
3.19  **REPEATED clause**

The REPEATED clause arranges body groups *side-by-side* across the page.

### 3.19.1 Format

```
┌── Format ─────────────────────────────────────────────┐
│ ────┬───┬───┬───┬───┬───┬─── │
│     │    │    │    │    │    │    │
│     ▶▶─ ────────┬───┬───┬─────── ──┘
│   ►►─  REPEATED ──┬─ integer-1 TIMES ──┬─ integer-2 TIMES ──┐
│               ──┬─ EVERY ──┬───┬─── ──┐
│               ──┬───┬─── ──┐│ COLUMNNS ──┐
│               ──┴───┴─── ──┘│ COLS ──┘
│               ──┴───┴─── ──┘│            │
│               ──┴─ integer-1 TIMES ──┴─ integer-2 TIMES ──┘
│               ──┴─ EVERY ──┴───┬───┬─── ──┘
│               ──┴───┬───┬─── ──┐│ COLUMNNS ──┐
│               ──┴───┴───┴─── ──┘│ COLS ──┘
│               ──┴──────────────────────────────────────┘
```

### 3.19.2 REPEATED Clause: Coding Rules

1. Write this clause at the 01-level of body groups (DETAIL or CH/CF) only.
2. You must code *either* the TIMES phrase *or* the EVERY/WIDTH phrase, or (preferably) both. EVERY and WIDTH have the same meaning.
3. Code *only the left-hand report group*. Report writer will automatically offset successive groups to the right of the left-hand group.
4. If you use the EVERY/WIDTH phrase, draw an imaginary "smallest box" around your group:

```
    aaaaaaaaaaaaa aa aaaaaaaaaaa
    bbbbbbbbbbbb
    cccccccccccc cccc
    ddddddd ddddddddddddddddddddd
```

The width of your box must not be greater than the EVERY/WIDTH integer-2. If any of your lines can have a variable rightmost column position, you must use the **maximum** expected size while drawing your box, but report writer cannot always predict the actual size at precompilation time. If the actual width of the group then causes it to encroach into the area occupied by the group to its right, a run time error will be issued.

5. There are no other restrictions on the size of your group or the clauses that you can use within it. For example, you may specify any number of LINES.
6. If you omit the TIMES phrase, report writer will examine your EVERY/WIDTH phrase and calculate how many repetitions of the group it can fit within the LINE LIMIT. If you use the identifier form of the LINE LIMIT clause, report writer will do this **dynamically** when you GENERATE the groups.
7. If you omit the `EVERY/WIDTH` phrase, report writer will examine your `TIMES` phrase and will calculate how `widely` it can space the repeats of your group at `regular` intervals. You cannot use the `identifier` form of the `LINE LIMIT` clause in this case.

8. The rightmost column of the rightmost `REPEATED` group must not go beyond the `LINE LIMIT`. This possibility can arise only if you use both the `TIMES` phrase and the `EVERY/WIDTH` phrase. In all other cases, report writer does the fit for you and makes sure that the repeats will fit side-by-side without violating the `LINE LIMIT`.

9. The `REPEATED` clause is not allowed in a report group that has no `LINE` clauses.

### 3.19.3 REPEATED Clause: Operation

1. If you code a `REPEATED` clause in a `DETAIL` group, report writer will place consecutive groups `side-by-side`:

```
  group #1  group #2  group #3
```

Each instance of the group is produced by `one` `GENERATE`. For example, to produce the above 3-up pattern, you would issue `three` successive `GENERATE`s. Each instance may display different data and may differ from its companions in certain characteristics by virtue of any `PRESENT` or `OCCURS... DEPENDING` clauses that you have coded.

2. Report writer sets up a `buffer` to hold your repetitions. If your group is `REPEATED` `three` times, as in the diagram above, this is what happens:

**First `GENERATE`**

The group is not produced but is placed in the buffer; the special register `REPEATED-COUNTER` is set to 1.

**Second `GENERATE`**

The second group is also not produced but is placed in the buffer; `REPEATED-COUNTER` is set to 2.

**Third `GENERATE`**

All three groups are produced side-by-side, with the second and third groups offset by the number of columns given in the `EVERY` phrase. The buffer is then cleared, ready for the next three groups. `REPEATED-COUNTER` is reset to zero.

The starting point for the repetitions is given by the `COLUMN` numbers in your group description. These will become the `COLUMN` numbers of the left-hand group. The following example shows the effect obtained:
01 ADDRESS-LABEL REPEATED 3 TIMES EVERY 20 COLUMNS.
   03 LINE + 2.
   05 COL 20 VALUE "xxxx".
   05 COL 26 VALUE "yyyy".
   03 LINE + 1.
   05 COL 23 VALUE "zzzzz".

If a different body group is produced, no more groups are placed side-by-side and any groups already in the buffer are output. Any unused repetitions will result in blanks on the right-hand side. The effect of writing:

```
INITIATE report
GENERATE group-A
GENERATE group-A
GENERATE group-A
GENERATE group-A
GENERATE group-A
GENERATE group-B
TERMINATE report
```

is shown in the following:

```
  group-A
  group-A
  group-A
  group-A
  (blank entry)
```

where group-A and group-B are DETAIL groups and A is REPEATED 3 TIMES.
Group B may also be REPEATED, but even if it is also REPEATED 3 TIMES, it will not be printed alongside group A, but will start a new series of repetitions. The same effect is seen if your program issues a TERMINATE when there are groups still in the buffer. Report writer will first produce them with blank unused entries on the right, so that no data will be lost.

4. If you need to have different groups placed side-by-side, you will have to define a single group and use the PRESENT WHEN clause above several LINE entries to create the impression of different groups at run time.

5. If a control break occurs that results in a CONTROL HEADING or CONTROL FOOTING group, it has the same effect as when you GENERATE a different DETAIL group. That is to say, any groups in the buffer are first output. This is true even if your CONTROL group is a dummy (with no LINE clauses to cause output). Briefly, your groups always appear in chronological order.

6. If a clause in your group references LINE-COUNTER (in a condition or as a SOURCE), you will always obtain its correct effective value. Report writer updates LINE-COUNTER just as though the group were actually being output, although the group is being placed in a buffer in memory. Then, when the next group is GENERATEd, LINE-COUNTER immediately reverts to the value it had at the start of the preceding group.

7. The page-fit test is applied to each repetition in turn. So, if your REPEATED group has a depth that may vary (because of a PRESENT WHEN clause or an OCCURS...DEPENDING at LINE level), room must exist for the largest group that has been GENERATEd in the current "pass" across the line.

8. You may place a REPEATED clause in a CONTROL HEADING or CONTROL FOOTING group, but this is of no use except when you use summary reporting (GENERATE report-name), because there is no other way that the same CONTROL HEADING or CONTROL FOOTING group can appear twice in succession.

3.19.4 Compatibility

The REPEATED clause is unique to new Report Writer.
3.20 SIGN clause

The SIGN clause enables a printable sign character to be produced automatically when the PICTURE has an "S" symbol.

3.20.1 Format a

3.20.2 SIGN Clause: Coding Rules

1. If you write a SIGN clause in an elementary entry, the entry must have a numeric PICTURE with an "S" symbol.

2. This clause acts on elementary fields, but you may also code it in a group level entry, where it applies to all numeric elementary entries within the group whose PICTURES begin with the "S" symbol.

3. Format b is unique to the REPORT SECTION. Each literal must be a single character non-numeric literal. At least one of the phrases must be present.

3.20.3 SIGN Clause: Operation

1. The use of the format a SIGN clause has been superseded by the "+" and "-" symbols of the PICTURE clause. It is included for consistency with basic COBOL standards. Refer to your COBOL language reference for further information.

2. The format b SIGN clause enables you to place symbols of your choice on the left or the right of any signed printed item to represent a negative amount. The LEADING literal (if specified) is placed immediately before the first character of the report field. If leading spaces are suppressed with the Z or floating currency symbol, the literal is placed immediately before the first non-space character. The TRAILING literal (if specified) is placed in the last character position. For example, to place parentheses around a negative payment (a common accounting requirement), you would code:

```
PIC SZZZ9 SIGN IS LEADING "(" TRAILING ")" SOURCE PAY
```

which would output a value of -12 as: (12)

while a value of -1234 appears as: (1234)
This feature can be made to work for positive values by specifying the negative of the source (SOURCE (- PAY) in the case above).

3.20.4 Compatibility

The use of the SIGN clause in the report section is allowed only by new Report Writer, which uniquely provides the LEADING literal TRAILING literal phrases.
3.21  **SOURCE clause**

This clause specifies the source field (or expression) that provides the contents of a field in your report. The source field is usually outside the REPORT SECTION, but may also be implicitly defined within it.

### 3.21.1 Format

```
SOURCE IS identifier expression
SOURCES ARE ROUNDED
```

### 3.21.2 SOURCE Clause: Coding Rules

1. Any valid COBOL identifiers or arithmetic expressions may be used as operands, including any COBOL term that may be the source item of a MOVE or COMPUTE for example, report writer special registers such as PAGE-COUNTER, GLOBAL or EXTERNAL items, and special compiler registers such as LENGTH OF.

2. Each operand may be subscripted and/or qualified if necessary. For example:

   ```
   SOURCE IS MIN-STOCK IN MAIN-RECORD (AREA-NO, DISTRICT-NO)
   ```

   The operand may have as many qualifiers, subscripts or indexes as are normally permitted for the subject of a MOVE statement. Relative subscripting and reference modification may be used.

3. You **must also code a PICTURE clause** in the same entry (unlike VALUE, which need not have a PICTURE). The PICTURE used must be compatible with the PICTURE of the operand(s). If you are in any doubt, look at the description of the identifier where it is originally defined in the FILE, WORKING-STORAGE or LINKAGE SECTION of your program. Ensure that its PICTURE is unedited and that you are not attempting to produce a COMPUTATIONAL field from a PICTURE X format. The two PICTURES may be of unequal length, in which case there will be truncation or space-filling on the right (for a non-numeric field) or truncation or zero-filling on the left (for a numeric field). The rules for the SOURCE statement are exactly those of the MOVE or COMPUTE statements of procedural COBOL.

4. The **SOURCE IS** and **SOURCES ARE** keywords may be omitted, except when immediately following a VARYING clause. We shall still refer to the clause as a **SOURCE clause**, however.
5. An expression may be any arithmetic expression containing any of the following symbols and keywords:

- for addition
- for subtraction
* for multiplication
/ for division
** for forming an exponent
( and ) to prioritize evaluation or to "structure" the code for documentation
SUM for a total (automatically reset to zero after output)
COUNT to show the number of times another REPORT SECTION item has appeared (also reset to zero after output)

In addition, the expression may have any number of identifiers, and these may be subscripted or qualified. Here are some examples of expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE IS NUMBER-ORDERED * UNIT-PRICE / 100</td>
<td></td>
</tr>
<tr>
<td>SOURCE IS ((SUM OF REP-SALARY) / NUMBER-IN-DEPARTMENT (DEPT-NO))</td>
<td></td>
</tr>
</tbody>
</table>

The rules for forming an expression are exactly as for the COBOL PROCEDURE DIVISION, as described in your COBOL language reference. For more information, see 3.23 SUM clause. The effect of a potential zero divide or size error depends on the choice of action on overflow. (See 2.8 OVERFLOW clauses.)

6. The ROUNDED phrase may be used in the same entry if you use a numeric PICTURE that has fewer digits to the right of the decimal point than the SOURCE identifier. It will ensure that the value produced is the numerically closer of the two possible values, instead of always truncating the unwanted digits. You can use ROUNDED with a single identifier as well as with an expression. (So you need not code "+ 0" for ROUNDED to be legal.) In the following example:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 COL 20 PIC 999 SOURCE SALARY ROUNDED.</td>
<td></td>
</tr>
</tbody>
</table>

if SALARY contains 100.50 or 100.60 or 100.99, the value produced will be 101, not 100.

For more details of the ROUNDED keyword, see in your COBOL language reference under the COMPUTE verb. ROUNDED is a clause in its own right and thus may be written at any location in the entry. If you have a multiple form of the SOURCE clause, or more than one such clause (see 3.21.4 Multiple SOURCES), ROUNDED will affect all of them wherever applicable.

7. You can indicate a multiple-choice entry by appending WHEN or UNLESS condition to the SOURCE clause, and then coding further consecutive pairs of SOURCE and WHEN/UNLESS clauses in the same entry (see 3.18.5 The Multiple-Choice Form).
3.21.3  **SOURCE Clause: Operation**

1.  **Rules for Generating Report Field**

   The effect of the *SOURCE* clause is best described by reference to the COBOL
   *MOVE* or *COMPUTE* statements, because it obeys identical rules:

<table>
<thead>
<tr>
<th>Form of SOURCE clause</th>
<th>Equivalent procedural statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier (without ROUNDED)</td>
<td>MOVE identifier TO report-field</td>
</tr>
<tr>
<td>identifier ROUNDED</td>
<td>ADD ZERO, identifier GIVING report-field ROUNDED</td>
</tr>
<tr>
<td>arithmetic-expression (without ROUNDED)</td>
<td>COMPUTE report-field = expression</td>
</tr>
<tr>
<td>arithmetic-expression ROUNDED</td>
<td>COMPUTE report-field ROUNDED = expression</td>
</tr>
</tbody>
</table>

   The special registers  **CURRENT-DATE** and  **TIME-OF-DAY** of OS/VS COBOL and
   DOS/VS COBOL make use of the conceptual data items  **DATE** and  **TIME**.

   - If the report field has "<" PICTURE symbols, or begins in a variable position, report
     writer will not store the result directly in the report field, but will use an
     intermediate area.

2.  **Reference to controls**

   If the SOURCE clause refers, directly or indirectly, to a  **CONTROL** operand, and
   the SOURCE is fetched at  **CONTROL FOOTING** time, the contents of the control
   identifier  **before the control break** will be used.  This means you will obtain
    **before-the-break** contents for your control fields in the following report groups:

   a.  In every  **CONTROL FOOTING**;
   b.  In the PAGE FOOTING and PAGE HEADING, if the page advance was
       caused by a  **CONTROL FOOTING** group.

   This rule applies whether the  **CONTROL** operand is used as a SOURCE by itself, or
   as a subscript, or as part of an expression.  It also applies if you refer to the
    **CONTROL** operand via a  **redefinition**, or via a  **group field** that contains it, or a
    **subordinate** field.  This is because the pre-break values are  **temporarily stored
   directly back in the control fields** outside the REPORT SECTION while  **CONTROL
   FOOTING** processing is being done.

   One important consequence of this rule is that, if you use fields defined under
   an FD in the  **FILE SECTION** as controls, your program must not execute any report
   writer statements - not even a  **TERMINATE** -  **after the input file** releases the
   buffers.  The correct order for "close down" is therefore:

   **TERMINATE** report
   **CLOSE** all report files and input files

   See 2.6  **CONTROL clause**, and 4.2  **GENERATE statement** for further details.
### 3.21.4 Multiple SOURCES

If you use the multiple form of the SOURCE clause by writing *more than one* identifier or expression after the keyword, you will avoid the effort of coding several separate entries. Note the following:

1. You may include the keyword **NONE** to indicate that a particular field is to have no contents stored in it. It is then treated as **ABSENT**. An example of the use of NONE will be found under the LINE clause (see 3.10.4 Multiple LINES Clause).

2. Your entry must be subject to at least one of the following:
   - A fixed OCCURS clause (not OCCURS...DEPENDING), or
   - A multiple LINES clause, or
   - A multiple COLUMNS clause.

3. The number of terms in your multiple SOURCES must equal the total number of repetitions the entry is subject to in all dimensions, or the number of repetitions of one or more of the inner dimension(s). For example, with the following layout:

<table>
<thead>
<tr>
<th>03</th>
<th>LINE OCCURS 4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>OCCURS 3.</td>
</tr>
<tr>
<td>05</td>
<td>COLS +2, +1</td>
</tr>
<tr>
<td></td>
<td>PIC... SOURCES ARE .....</td>
</tr>
</tbody>
</table>

The number of SOURCES should be either 2 (just the inner dimension), 6 (the product of the inner two dimensions), or 24 (all the repetitions).

4. If the terms cover more than one dimension, they are distributed along the innermost dimension, periodically stepping to the next entry in the outer dimension(s). For example:

<table>
<thead>
<tr>
<th>03</th>
<th>LINE 2 STEP 1 OCCURS 3.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>COL 2 STEP 5 OCCURS 4 PIC XXX SOURCES ARE</td>
</tr>
<tr>
<td></td>
<td>MONTH-NAME(1) MONTH-NAME(2) MONTH-NAME(3) MONTH-NAME(4)</td>
</tr>
<tr>
<td></td>
<td>MONTH-NAME(8) MONTH-NAME(6) MONTH-NAME(7) MONTH-NAME(5)</td>
</tr>
<tr>
<td></td>
<td>MONTH-NAME(9) MONTH-NAME(10) MONTH-NAME(11) MONTH-NAME(12).</td>
</tr>
</tbody>
</table>

will result in:

<table>
<thead>
<tr>
<th>JAN</th>
<th>FEB</th>
<th>MAR</th>
<th>APR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUG</td>
<td>JUN</td>
<td>JUL</td>
<td>MAY</td>
</tr>
<tr>
<td>SEP</td>
<td>OCT</td>
<td>NOV</td>
<td>DEC</td>
</tr>
</tbody>
</table>

This technique is useful when your SOURCE items are not already conveniently arranged in a table or when, as in the case above, the order is *irregular*.

5. If there are two or more dimensions and the number of terms matches only the inner dimension(s), the terms are recycled from the first SOURCE operand for each repetition of some outer dimension. For example, in the following case:
03  LINE OCCURS 4.
05  COLS 1, 6, 11  PIC... SOURCES ARE JAN, FEB, MAR.

the values of JAN, FEB and MAR will be repeatedly stored in each line. However, you may vary the contents actually stored by allowing a VARYING operand to advance in step with each occurrence of the outer dimension and by using it as a subscript in the lower-level entry:

03  LINE OCCURS 4  VARYING LINE-SUB.
05  COLS 1, 6, 11  PIC... SOURCES ARE QTR-MO-1 (LINE-SUB), QTR-MO-2 (LINE-SUB), QTR-MO-3 (LINE-SUB ).

6. If a ROUNDED phrase is present in the entry, every SOURCE item will be rounded.
7. The multiple SOURCES may be used as one or more of the alternatives within a multiple-choice entry. You need not use a multiple SOURCES in every alternative:

05  COLS 1, 12, 25             PIC ZZ,ZZZ,ZZZ9 
SOURCES BASIC-AMOUNT TAX EXTRA WHEN REBATE-FLAG = "N" (- BASIC-AMOUNT) (- TAX) (- EXTRA) WHEN OTHER.

8. You may omit the SOURCE keyword, except when immediately following a VARYING clause.
9. Other examples of the multiple SOURCES clause will be found under 3.4.4 Multiple COLUMNS Clause and 3.10.4 Multiple LINES Clause.

3.21.5 Compatibility

All aspects of the following features are unique to new Report Writer:

- SOURCE keyword being optional,
- Arithmetic-expression format,
- ROUNDED phrase,
- Use of SUM or COUNT term as operand,
- Multiple format,
- Multiple-choice format.
3.22 STYLE clause

This clause enables the program to make use of any special effects provided by the printer or output device.

3.22.1 Format

```
STYLE IS ┌──┬──┐
  ┌ │  │
  │ ▼ │
  │   │
  │   └───┬─┬───┬──────┐
  │       │ │      │       │
  └───┬─┬───┬──────┘       └───
        │                    │
        │                  └───
        │              NORMAL
        │              └───
        │                  WHEN condition
        │                  └───
        │                        ┌──┐
        │                        │  │
        │                        │  │
        └───┬─┬───┬──────┘       └───
              │                    │
              │                  └───
              │              NORMAL
              │              └───
              │                  WHEN condition
              │                  └───
              │                        ┌──┐
              │                        │  │
              │                        │  │
              └───┬─┬───┬──────┘       └───
                    │                    │
                    │                  └───
                    │              NORMAL
                    │              └───
                    │                  WHEN condition
                    │                  └───
                    │                        ┌──┐
                    │                        │  │
                    │                        │  │
                    └───┬─┬───┬──────┘       └───
                          │
                          │
                          │
                          └───
```

3.22.2 STYLE Clause: Coding Rules

1. **STYLE** may be coded at any level, including in the FD (see 2.2 Report Files) or RD (see 2.3 REPORT SECTION and RD). The WHEN condition phrase cannot be used in the FD or RD entry. STYLE cannot be used if the device-name of the corresponding file is **NONE** (see 2.2.2 rule 11 above).

2. **NORMAL** may be coded instead of a style-name, meaning that no special effect is to be produced. It must be the only style-name in the clause and there must not be a WHEN condition phrase.

3. Apart from **NORMAL**, each style-name names a style or special effect that must be obtainable from the output device. The type of output device in use is given in the **TYPE** phrase of the SELECT statement. The style-names available are either predefined or user-defined for the particular output device. This check, and the processing of the styles themselves, may be delayed until run time by writing **DEFERRED** in the **TYPE** phrase. This enables the program to run, in theory, with a variety of different output devices, even when they are widely dissimilar. For a mainframe, special device handling is usually the province of a user-written report file handler - see Installation and Operation. For example, **STYLE HIGHLIGHT** might be implemented by any of the following means:
   a. shadow printing,
   b. switching to a different font,
   c. printing in larger letters,
   d. on a screen, by displaying intense,
   e. "double-hammering" on an impact line-printer.

4. The following are standard device-independent style-names. The first two are available with every **TYPE** of printer (other than **TYPE NONE**). The last two are available with all printers except the most basic.

   **UNDERLINE** causes the report field to be underscored. It is used for headings.

   **HIGHLIGHT** causes the report field to stand apart from the others, normally by appearing in bold or intense. It is used to give emphasis to certain fields.
ALT-FONT causes the report field to appear in a second contrasting font or typeface, such as italic.

GRAPHIC causes the report field to appear in a third contrasting font.

The remaining style-names used in the examples that follow are purely for the purpose of illustration, and are not necessarily available on any particular device.

5. Style-names are grouped into mutually-exclusive classes. Styles HIGHLIGHT, ALT-FONT and GRAPHIC are mutually-exclusive but UNDERLINE belongs to a separate (one-member) class. The classes are defined in the Printer Description File. It is not valid to code two styles belonging to the same class in the same entry. Thus the following clause is not valid:

STYLE IS ALT-FONT GRAPHIC

However it is valid to place different members of the same class in nested entries, in which case the prevalent style is noted and restored at the end of the nested entry. Thus, in the following entries, TOMATO is RED while all the other entries are GREEN:

<table>
<thead>
<tr>
<th>03</th>
<th>LINE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>STYLE GREEN.</td>
</tr>
<tr>
<td>07</td>
<td>COL 1 PIC X(20) SOURCE BEAN.</td>
</tr>
<tr>
<td>07</td>
<td>COL 21 PIC X(20) SOURCE TOMATO STYLE RED.</td>
</tr>
<tr>
<td>07</td>
<td>COL 41 PIC X(20) SOURCE ARTICHOKE.</td>
</tr>
<tr>
<td>07</td>
<td>COL 61 PIC X(20) SOURCE PEA.</td>
</tr>
</tbody>
</table>

6. The STYLE clause cannot be repeated in an entry. Hence, if a multiple-choice entry is required, with a different STYLE on each choice, separate entries must be coded.

7. If STYLE is used in a report that uses the PAGE BUFFER feature it should not be coded in a Report Group Description at a level higher than the LINE level.

8. No style-name may be the same as one which is already in effect. For example, the following is illegal:

<table>
<thead>
<tr>
<th>05</th>
<th>LINE  STYLE IS ALT-FONT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>COL 1 STYLE IS ALT-FONT UNDERLINE.</td>
</tr>
</tbody>
</table>

9. The STYLE clause cannot be used on an unprintable elementary entry.

3.22.3 STYLE Clause: Operation

1. Report writer implements the STYLE feature in one of three ways:

   a. By inserting non-printable control characters or escape sequences into the report data, before or after the data affected, or both.

   b. By re-printing a line or part of a line without advancing the carriage. This method is commonly used to highlight text and occasionally to produce an underline effect.
3.22 – STYLE

By some special technique chosen and implemented by the user (see Independent Report File Handlers).

The method of implementation of each style is defined explicitly for each TYPE of device, and may be altered by the user.

2. The STYLE clause is transparent to the layout of the report. That is, it does not affect any of the other clauses or entries in the report description. For example, COLUMN numbers are unchanged, even though report writer may be inserting extra control sequences into the report lines. You can therefore simply add STYLE clauses to enhance existing programs.

3. Several STYLE names can be combined in one clause, for example:

   STYLE HIGHLIGHT UNDERLINE

   Here the different characteristics are simply superimposed on each other (but see next item).

4. The scope of the STYLE clause is decided by the level of the entry in which it is coded, thus:

   a. In an elementary entry, the STYLE clause applies only to the elementary field, for example:

      03 LINE.
      05 COL 1 PIC X(20) SOURCE CUST-NAME.
      05 COL 25 PIC $(5)9.99 SOURCE ACCOUNT-BALANCE.
      05 COL 37 PRESENT WHEN ACCOUNT-BALANCE < 0 VALUE "IN ARREARS" STYLE HIGHLIGHT.

   b. In a LINE entry, the STYLE clause applies to the whole line, as in:

      03 LINE  STYLE EXTRA-WIDE.
      05 COL 1 "Annual Report".
      05 COL 31 PIC X(40) SOURCE COMPANY-NAME.
      05 COL 75 PIC "Year: "X(4) SOURCE ACCOUNT-YEAR.

   c. In a report group entry, the STYLE clause applies to the whole group, for example:

      01 WARNING-NOTICE TYPE DE STYLE RED ITALIC.

   d. In an RD entry, the STYLE clause applies to the entire report

      RD PERSONNEL-SHEETS
      STYLE IS LANDSCAPE.

      If any control characters are output, this happens during the execution of each INITIATE or TERMINATE for the report, or both.

   e. In an FD entry, the STYLE clause applies to the entire report file:

      FD REPORT-FILE
      REPORT IS PERSONNEL-SHEETS
      STYLE IS LOAD-COURIER-FONT-1 LOAD-HELVETICA-FONT-2
      SWITCH-ON-SWITCH-OFF.
If any control characters are output, this happens during the execution of each `OPEN` or `CLOSE` for the file, or both.

5. The WHEN clause causes the STYLE to take effect only when the condition is true, for example:

   ```cobol
   03 LINE + 2 STYLE LARGE WHEN WS-PAPER-WIDTH > 80.
   05 COL 21 PIC ZZZ9 SOURCE PERCENTAGE
   STYLE UNDERLINE WHEN PERCENTAGE > 100.
   ```

6. Since you can only code STYLE once per entry, you cannot vary the STYLEs in a multiple-choice entry, and instead must code separate entries, as here:

   ```cobol
   05 COL 15 PRESENT WHEN W-VAL > 0 VALUE "POSITIVE" STYLE HIGHLIGHT.
   05 COL 15 PRESENT WHEN W-VAL = 0 VALUE "ZERO" STYLE ALT-FONT.
   05 COL 15 PRESENT WHEN W-VAL < 0 VALUE "NEGATIVE" STYLE GRAPHIC.
   ```

7. If STYLE is defined on an elementary field that has suppressed zeros or trailing spaces, the STYLE will apply only to the characters printed. For example, the coding:

   ```cobol
   05 COL 11 PIC ZZZZZ9 SOURCE W-PAYMENT STYLE UNDERLINE.
   05 COL + 2 VALUE "PAYMENT OVERDUE".
   ```

   might result in:

   ```
   15 PAYMENT OVERDUE
   3156 PAYMENT OVERDUE
   152722 PAYMENT OVERDUE
   ```

   If you do not want this effect, code the STYLE clause at a group level, e.g.

   ```cobol
   05 STYLE UNDERLINE.
   07 COL 11 PIC ZZZZZ9 SOURCE W-PAYMENT.
   ```

3.22.4 **Compatibility**

The STYLE clause is unique to new Report Writer.
3.23 **SUM clause**

This clause automatically forms **totals** from any other numeric field. You may also use it as a term in an arithmetic-expression in a SOURCE clause.

3.23.1 **Format**

```
SUM OF report-section-data-name
    identifier
    expression
    UPON group-name
```

```
RESET ON control-id ROUNDED
```

3.23.2 **SUM Clause: Coding Rules**

1. There are two ways to code the SUM clause:
   a. **As a clause in its own right** You write the clause in place of a SOURCE VALUE or FUNCTION clause. For example:
      
      | 05 COL 21 PIC ZZZ9 SUM OF SALE. |
      |
   b. **As a term in an expression**; for example
      
      | 05 COL 21 PIC ZZZ9 SOURCE IS ((SUM OF REVENUE) + (SUM OF TAX)) / 100. |
      |
      This type of expression can only be used as a SOURCE operand (although, as usual, the SOURCE keyword is optional). You cannot use a SUM term as part of a condition or, for instance, as a parameter to a FUNCTION.

2. The **item summed** can be either of two things:
   a. **The name of any numeric data item in your REPORT SECTION** (other than a FUNCTION entry). To use this, you must place a data-name on the item in the REPORT SECTION you want to total. Suppose you need to form a total from the following report field:
      
      | 05 COL 41 PIC 99999 SOURCE WAGES. |
      |
      Just place a data-name of your own choosing on it, such as:
      
      | 05 R-WAGES COL 41 PIC 99999 SOURCE WAGES. |
and you may now write **SUM OF R-WAGES** in another item in your report to form a total. This is called a **REPORT SECTION SUM** clause. A REPORT SECTION SUM operand **never** has any subscripts or arithmetic symbols - you can write only: **SUM OF data-name**.

The data item referred to in the SUM clause may be qualified by the report-name, as in **SUM R-PAYMENT IN SUMMARY-REPORT**. By default, the qualifier is the same report as the one in which the SUM is defined. So if the same data-name appears in more than one report, it may be referred to by a SUM clause **in its own report** without qualification. (It cannot be referred to by a SUM in a different report without a qualifier.) This means that you can duplicate a complete report description with SUM clauses without changing any of the data-names on SUM entries.

b. **A numeric identifier or expression from outside the REPORT SECTION.** In this form, it is similar in appearance to the operand of a SOURCE clause. This is called a **non-REPORT SECTION SUM** clause.

(Unless a naming convention is observed, there is no way of telling from the data-name whether the SUM clause references a REPORT SECTION SUM or not. In the examples in this volume, we sometimes use a prefix such as "R-" to designate a REPORT SECTION item, so that "SUM OF R-..." will be recognized as a REPORT SECTION SUM. Observing a standard like that will make your REPORT SECTION easier to follow.)

A SUM clause may total **several** items, and you may combine REPORT SECTION and non-REPORT SECTION items; for example:

```
SUM OF W-BASIC, R-BONUS, R-OVERTIME
```

where **W-BASIC** is a non-REPORT SECTION item, **R-BONUS** is a REPORT SECTION item in a different report group and **R-OVERTIME** is a REPORT SECTION item in the same report group. In this case, the total field is always the sum of the individual totals that would be formed from each of its items. You should then interpret the remarks in the remainder of this section for each of the single SUM operands individually. This is **not** similar to the multiple SOURCES clause (despite their syntactic similarity), since only one printable item is defined with the SUM clause.

3. Your report may contain as many entries with SUM clauses as you wish, in any **TYPE** of group (even RH). Only **elementary** entries can have a SUM clause.

4. If you code the **UPON** phrase, each group-name must be the name of a DETAIL group and should not be the same as the group you are currently defining. The group-name may be **qualified**. If not, the current report is assumed.

5. If your SUM clause is in a **DETAIL** group and the operand of your SUM clause is **not in the REPORT SECTION** (a rare situation), your report should contain more than one DETAIL group and you should logically code the **UPON** phrase, in order to specify on which GENERATE you want adding to take place.

6. If you code the **RESET** phrase, the **control-id** operand must be one of those defined in the CONTROL clause of your report (including REPORT, whose presence is assumed there). If you are currently defining a **CONTROL FOOTING** group, the level of the control in your RESET ON should be **higher** than the level of this group. (It may also be **equal** to the level of the current group, in which case resetting occurs at the normal time, and the phrase is therefore redundant.)
7. The RESET phrase cannot be defined anywhere in a multiple CONTROL FOOTING group.

8. You **may** code the SUM clause **more than once** in an entry. The effect of this is to add together the totals formed from each of the clauses. Hence:

| SUM A | SUM B | gives the same result as | SUM A B |

(and, if A and B are non-REPORT SECTION items, SUM (A + B)). This separation is essential only if you need to use a certain UPON phrase with one but not the other, for example:

```plaintext
... SUM A UPON DETAIL-1 SUM B UPON DETAIL-2.
```

9. **ANS-85 note.** Since a reference-modified identifier is regarded as **non-numeric**, you cannot SUM it, either directly as the operand of your SUM clause, or indirectly by naming a REPORT SECTION item where it is used as a SOURCE.

### 3.23.3 SUM Clause: Operation

Report writer performs the **totalling**, **presenting**, and **resetting** (to zero) of your totals completely automatically. These three stages are covered in the next three numbered paragraphs.

1. **Totalling**

   The method used for totalling depends on whether the item referred to by SUM is a data-name on a REPORT SECTION entry (called henceforth a "REPORT SECTION SUM"), or not.

   a. **REPORT SECTION SUM**

      Whenever the originating data item named in your SUM clause is output in the report, the item it references is also added into the (internal) total field. If the REPORT SECTION data item totalled contains a SOURCE or VALUE clause, the amount added to the total is the **SOURCE or VALUE** operand outside the REPORT SECTION, **not** the intermediate REPORT SECTION field. For example, if you write:

      ```plaintext
      05 R-FLD1 COL 1 PIC 999 VALUE 100.
      05 R-FLD2 COL 5 PIC 9999 SOURCE W-PAYMENT.
      ... SUM OF R-FLD1, R-FLD2.
      ```

      the fields added into the total will be 100 and W-PAYMENT, **not** the report fields R-FLD1 and R-FLD2. Thus, if **W-PAYMENT** has a PICTURE of **9(5)V99** rather than **9(4)**, the full originating value - not the truncated value that appears in the report line - will be added.
There are two names used to distinguish two cases of REPORT SECTION SUMming:

i. Cross-Footing

In the next illustration, the item to be totalled is in the same report group as your SUM clause:

<table>
<thead>
<tr>
<th>FURNITURE SALES</th>
<th>THIS YEAR BY QUARTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPRING</td>
<td>SUMMER</td>
</tr>
<tr>
<td>$2300</td>
<td>$3400</td>
</tr>
</tbody>
</table>

01 YEARS-SALES TYPE DE LINE + 1.
05 R-QUARTER COL 3 STEP 10 OCCURS 4 VARYING SEASON PIC $(6)9 SOURCE QTLY-SALE (SEASON).
05 COL +10 PIC $(7)9 SUM OF R-QUARTER.

The SUM entry may appear earlier in the report group than the item it is totalling and this may be carried on to any number of stages. Thus the physical order of totals and items being totalled within a single group is immaterial. Report writer decides for itself the order in which totalling must be done. Hence the correct result is obtained by coding for example:

05 R-A ... SUM OF R-B ...
05 R-B ... SUM OF R-C ...
05 R-C ... SUM OF R-D ... etc.

However, you must avoid circular dependencies such as:

or

05 R-A ... SUM OF R-A ...
05 R-A ... SUM OF R-B ...
05 R-B ... SUM OF R-A ...

ii. Rolling Forward

In the next illustration, the item to be totalled is in a different group from your SUM clause:
<table>
<thead>
<tr>
<th>FURNITURE SALES</th>
<th>CITY: DENVER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>THIS YEAR</td>
</tr>
<tr>
<td>ARMCHAIRS</td>
<td>$3500</td>
</tr>
<tr>
<td>SETTEES</td>
<td>$300</td>
</tr>
<tr>
<td>WRITING DESKS</td>
<td>$2200</td>
</tr>
<tr>
<td>DINING TABLES</td>
<td>$1700</td>
</tr>
<tr>
<td><strong>TOTALS</strong></td>
<td><strong>$3500</strong></td>
</tr>
</tbody>
</table>

In this example, the "boxes" contain instances of the different report groups. The values to be added appear in a DETAIL, and the total is in a CONTROL FOOTING.

You may use also this method of summing between any reasonable combination of groups, namely any of the following:

- **Lower CONTROL FOOTING** to **higher CONTROL FOOTING**,  
- **DETAIL** to **DETAIL**,  
- **Any group** to a **REPORT FOOTING**,  
- **Any body group** to a **PAGE FOOTING**,  
- **PAGE FOOTING** to a **CONTROL FOOTING**.

Frequently there are several items that could be rolled forward to produce the same result. For example, your YEAR totals are the total of the twelve MONTH totals; they are also the total of your fifty-two WEEK totals and your 365 DAY totals. Rolling forward the immediately-lower-level total (such as MONTH totals into YEAR totals) is the most efficient technique.
b. **Non-REPORT SECTION SUM**

Here totals are gradually accumulated from values held outside the REPORT SECTION. This method was much used in the earlier versions of report writer and you may possess some old programs that use it. This is usually referred to as **subtotaling**. Because the identifier or expression lies outside the REPORT SECTION, it is not as clear as it is with a REPORT SECTION SUM clause exactly when the values are added into the total, so the following rules are important:

i. If **SOURCE SUM correlation** is in effect (see 3.23.5 **Subtotalling and SOURCE SUM Correlation**), adding takes place when any DETAIL is being GENERATED that contains the same identifier as a SOURCE item as the identifier appearing in the SUM clause.

ii. If the **UPON** phrase is used, adding takes place when a DETAIL group specified in that phrase is GENERATED.

iii. If **SOURCE SUM correlation** is not in effect and there is no UPON phrase, adding takes place on each GENERATE that refers to the report.

Almost all totalling can also be accomplished by means of REPORT SECTION SUM clauses.

2. **Presenting the total**

When the entry containing you SUM is output, the (unedited) total to-date is used as an internal "source" for the contents of the field. Thus, if you had coded:

```cobol
05   COL 5    PIC $$,$$$,$$9.99   SUM R-PAYMENT.
```

the internal total field (which is fully described below under **Use of Total Fields**) is MOVEd to this report field, and edited according to the given PICTURE according to the same rules as the **SOURCE** clause.

3. **Resetting the total**

The total is reset (cleared to zero) at the end of the processing for the report group in which it is defined, unless you override this using the RESET phrase as described below (see 3.23.7 **The RESET Phrase**). Thus the total field remains available for use anywhere in the same group (within a SOURCE for instance) before its contents are erased.

4. When adding a value into the total, the report writer code obeys the rules of the **ADD** statement. Unless you coded **SUM OVERFLOW PROCEDURE IS OMITTED** in the RD, a SIZE ERROR test is always done and if there is size error, a run time error is signalled at once.

5. If the SUM clause appears in a **multiple CONTROL FOOTING** group, the SUM clause has its usual effect in the lowest level group and in the higher levels acts by **rolling forward** each previous level. For example, in the following structure:
the SUM clause acts like three different SUM clauses. At the lowest level (CITY), it behaves as defined (by subtotalling W-PAY). The CITY total is then rolled forward into the COUNTY total on change of CITY and the COUNTY total is rolled forward into the STATE total on change of COUNTY.

6. Summing a Repeated Item

If you wish to sum a repeating item, you may form totals along any of four different axes. These are, from minor to major:

Axis 1: COLUMN Axis

Axis 2: COLUMN-Group Axis (below LINE and above COLUMN)

Axis 3: LINE Axis

Axis 4: LINE-Group Axis (above LINE and below report group)

Report writer will automatically total the repetitions of your field along any axes necessary to form the total. The direction of the adding depends not on the syntax of the clause you use but on where you place it. If your SUM clause has no repetitions at all (is not subject to an OCCURS clause or a multiple LINE or COLUMN clause), report writer will total all occurrences of the field. You can thus total a REPORT SECTION table of any number of dimensions. If your SUM clause is part of a repeating report field, then the item being summed must repeat the same number of times along the same axes as the SUM entry. Your field will be output along any of the axes shared and totalled along any of the axes not shared by the SUM entry. An example will make this clearer. In the following case, the field R-QUARTER has an OCCURS clause in two axes: the LINE axis and the COLUMN axis.

<table>
<thead>
<tr>
<th></th>
<th>SALES OF SPORTS GEAR</th>
<th>YEARLY BY QUARTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPRING</td>
<td>SUMMER</td>
</tr>
<tr>
<td>1988</td>
<td>$2300</td>
<td>$3400</td>
</tr>
<tr>
<td>1989</td>
<td>$3200</td>
<td>$3600</td>
</tr>
<tr>
<td>1990</td>
<td>$3500</td>
<td>$4000</td>
</tr>
<tr>
<td>1991</td>
<td>$3600</td>
<td>$4300</td>
</tr>
<tr>
<td>1992</td>
<td>$3800</td>
<td>$4200</td>
</tr>
</tbody>
</table>

The following code produces this layout:

01  SALES-DETAIL  TYPE DE.
03  LINE + 1      OCCURS 5      VARYING YEAR-NO.
05  COL 1         PIC 9(4)      SOURCE 1987 + YEAR-NO.
05  R-QUARTER COL 6 PIC $(6)9  OCCURS 4 STEP 10 VARYING SEASON SOURCE SALE (YEAR-NO SEASON).
To obtain row totals, code the SUM clause as an entry at a level within the same LINE as the items referred to in SUM. (The entry below is named ROW-TOTAL but you may choose any data-name). To obtain column totals, place the SUM clause within a separate LINE and give it the same number of horizontal repetitions as its operand. (The entry below is arbitrarily named COL-TOTAL.)

The final result is as follows:

<table>
<thead>
<tr>
<th></th>
<th>SPRING</th>
<th>SUMMER</th>
<th>FALL</th>
<th>WINTER</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>$2300</td>
<td>$3400</td>
<td>$1600</td>
<td>$3500</td>
<td>$10800</td>
</tr>
<tr>
<td>1989</td>
<td>$3200</td>
<td>$3600</td>
<td>$2700</td>
<td>$3000</td>
<td>$12500</td>
</tr>
<tr>
<td>1990</td>
<td>$3500</td>
<td>$4000</td>
<td>$3400</td>
<td>$4300</td>
<td>$15200</td>
</tr>
<tr>
<td>1991</td>
<td>$3600</td>
<td>$4300</td>
<td>$4800</td>
<td>$4500</td>
<td>$17200</td>
</tr>
<tr>
<td>1992</td>
<td>$3800</td>
<td>$4200</td>
<td>$3900</td>
<td>$4750</td>
<td>$16650</td>
</tr>
<tr>
<td>TOTALS</td>
<td>$16400</td>
<td>$19500</td>
<td>$16400</td>
<td>$20050</td>
<td>$72350</td>
</tr>
</tbody>
</table>

Notice that the field ALL-TOTAL could also have been coded as SUM OF R-QUARTER (which is less efficient) or SUM OF ROW-TOTAL. Also notice that the data-names ROW-TOTAL and ALL-TOTAL are not referenced and could have been omitted.

7. You may require just a single total of all the entries (the "corner" total in the above illustration), without any row or column totals. Just write the ALL-TOTAL entry alone:

```
$72350
```

8. Other Axes

The examples shown earlier cover only two out of the four possible axes. You may also SUM along the other two axes, that is: groups of COLUMNS and groups of LINES. As an example of groups of COLUMNS, take this layout, where we total the four weeks in each month horizontally:
<table>
<thead>
<tr>
<th>ITEM</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>WEEKS</th>
<th>TOTAL</th>
<th>WEEKS</th>
<th>TOTAL</th>
<th>WEEKS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>APR 1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>MAY 1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>GOLF</td>
<td>2580</td>
<td>670</td>
<td>300</td>
<td>200</td>
<td>1100</td>
<td>450</td>
<td>600</td>
</tr>
<tr>
<td>SQUASH</td>
<td>980</td>
<td>410</td>
<td>200</td>
<td>50</td>
<td>340</td>
<td>250</td>
<td>30</td>
</tr>
<tr>
<td>TENNIS</td>
<td>1810</td>
<td>450</td>
<td>200</td>
<td>70</td>
<td>90</td>
<td>90</td>
<td>45</td>
</tr>
</tbody>
</table>

This example also illustrates the fact that cross-foot totals need not appear to the right, or below, the figures from which they are formed.

9. If your program contains more than one report, you are not restricted only to rolling forward within one report. A SUM clause in one report may refer to a named numeric data item in a different report. As with totalling in a single report, the value of the referenced field is added whenever the field is output. (This does not apply to non-REPORT SECTION summing, however, when SOURCE SUM correlation is in effect: see Clause 2.4 ALLOW clause.) Note that all total fields in a Report are cleared by the INITIATE statement for the report. Therefore, if you need to SUM from one report to another, be sure that both are INITIATEd at the start of the processing.

10. A PRESENT WHEN clause may test the value of any total fields referenced in its condition-operand. However, it should then not contain within its scope either an entry with a SUM clause or an entry that is summed by a SUM clause. See Clause 3.18.4 Effect of PRESENT WHEN on SUM.

11. Totalling Unprintable Items

You may total unprintable items in your REPORT SECTION. This is useful when you want to print the totals only, not the unprinted individual values. In the next example, the report computes a yearly total from 12 unprinted monthly values:

```
   YEARY EXPENDITURE
1991  $2300.00
1992  $3766.50
```

```
   ── jan + feb + ... + dec
```

```
  03 LINE.
  05 COL 2   PIC 9(4) SOURCE YEAR.
  05 R-EXPENSE   PIC 9(6)V99 OCCURS 12
  VARYING MONTH SOURCE F-EXPENSE (MONTH).
  05 COL 7   PIC $(6)9.99 SUM OF R-EXPENSE.
```
12. It is not possible to accumulate in an instant a single total all the entries in a table outside your REPORT SECTION without using an intermediate unprintable table as in the example above. However, you may use a non-REPORT SECTION SUM clause to accumulate - over time - single entries in a table into a corresponding matching number of totals, typically in a CONTROL FOOTING. Since your SUM operand is not in the REPORT SECTION, you may use subscripts with it, just as you would in the SOURCE clause:

```
SPORTS CLUB REVENUE REPORT

END-OF-REPORT SUMMARY:
TOTALS: GOLF     TENNIS     SWIMMING     SQUASH     RESTAURANT
$32400     $19500     $16400     $20050     $72350
```

01  CF FOR REPORT.
03  LINE  ...
05  COL 7 STEP 11 OCCURS 5 VARYING SPORT
PIC $(6)9  SUM OF W-SPORT-REVENUE (SPORT).

13. If the SUM operand is not in the REPORT SECTION, you may also total arithmetic expressions, such as:

```
05  COL 54  PIC ZZZZ9  SUM OF (W-INCOME - W-TAX) * 100.
```

An evaluation is performed for the expression on every GENERATE before the result is added into the total field, so the use of this technique is less efficient - and more susceptible to rounding errors - than a method using unprintable intermediate totals. (The accumulation is not affected by the status of SOURCE SUM correlation.)

3.23.4 Use of Total Fields

1. If you give a data-name to an entry that contains a SUM clause (not a SUM term that is part of an expression), report writer will relate the data-name to its own (internal) total field, not to the (edited) external field in the report line, as would be the case with a SOURCE VALUE or FUNCTION clause. (The total field is called a sum-counter in older texts, but total field is clearer as it does not conflict with the COUNT clause.) Compare these two cases:

```
a.  With SOURCE etc.:
05  R-PAYMENT COL 21  PIC $(7)9.99  SOURCE IS WS-PAYMENT.

Field R-PAYMENT is actual report-line field, except when referenced in a SUM clause, in which case WS-PAYMENT is what is added.
```
b. With SUM:

<table>
<thead>
<tr>
<th>Field TOT-PAYMENT</th>
<th>COL 21</th>
<th>PIC $(7)9.99</th>
<th>SUM OF WS-PAYMENT.</th>
</tr>
</thead>
</table>

Field TOT-PAYMENT is the REPORT SECTION's internal total field.

Field TOT-PAYMENT is the REPORT SECTION's internal total field.

$$9.99$$

Report line field has no name.

2. The internal total field is a pure numeric, signed COMPUTATIONAL field with as many integral and fractional digits as the SUM entry. Hence, no precision will be lost when the total field is stored in the report field. If the SUM clause refers to a REPORT SECTION item, the precision of the total field is increased if necessary so that it has at least as many integral and fractional digits as the entry being totalled. Hence the total will have at least the precision of the field being totalled, rounding or truncation taking place, if indicated, when the field is output not while it is being accumulated. Here are some examples:

<table>
<thead>
<tr>
<th>PICTURE in SUM Entry</th>
<th>PICTURE of SUM Operand</th>
<th>PICTURE of Internal Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999</td>
<td>(not in REPORT SECTION)</td>
<td>S9(5) COMP</td>
</tr>
<tr>
<td>$(6)9-</td>
<td>$(5)9.99-</td>
<td>S9(6)V99 COMP</td>
</tr>
</tbody>
</table>

Note that the total field is always signed, even if the report line field is not signed, so the adding into the total field is always algebraic.

3. Accessing the Total Fields

You may access the internal total field directly in the following four ways.

a. In a SOURCE Clause

You may capture an internal total field with a SOURCE clause, either as its single operand or as a term in an operand that is an expression. The value you obtain depends on the category of the total field you access, as follows:

i. SOURCE Refers to Total Field Defined in the Same Report Group

You will always obtain a true value of the total as specified. The position of the SOURCE clause within the group is immaterial, as report writer always computes the totals before producing any of the group's report lines (which is when SOURCE operands are filled in).

This feature is useful when you wish to form an unprintable total from a non-REPORT SECTION field without loss of precision and then output it rounded. You might wish to form totals to cents precision and output them to the nearest dollar. You should then write two entries in the same report group, one of which has no COLUMN clause and is therefore unprintable, as in this example:

3.23 – SUM
If you had written instead:

```
  05 COL 21 PIC -(6)9 SUM OF WS-FIELD.
```

the internal total field would have the implied PICTURE S9(6) COMP, so you would lose precision each time the (truncated) value of WS-FIELD is added in. Truncation would not arise if the SUM operand were a REPORT SECTION data-name, since the precision of the summed field is then also taken into account in establishing the required precision of the sum field.

Alternatively, you may make use of report writer’s less efficient but highly accurate default PICTURE S9(12)V9(6) for SUM terms in SOURCE clauses, by coding:

```
  05 COL 21 PIC -(6)9 SOURCE (SUM OF WS-FIELD).
```

The next example uses an arithmetic expression involving total fields:

```
  05 COL 10 VALUE "TOTAL INCOME:".
  05 R-TOT-INCOME COL + 2 PIC Z(6)9 SUM OF INCOME.
  05 COL 35 VALUE "TOTAL TAX:".
  05 R-TOT-TAX COL + 2 PIC Z(6)9 SUM OF TAX.
  05 COL 55 VALUE "NET PAY:".
  05 COL + 2 PIC Z(6)9 SOURCE (R-TOT-INCOME - R-TOT-TAX).
```

It would also have been correct to code the last entry as:

```
  05 COL + 2 PIC Z(6)9 (SUM OF INCOME) - (SUM OF TAX).
```

However, the first code shown is more economical since we are printing the individual totals R-TOT-INCOME and R-TOT-TAX already elsewhere and can therefore re-use them in the expression instead of summing them again.

ii. SOURCE Refers to Total Field Not Defined in Same Group

If the SUM total field referred to in turn refers to a non-REPORT SECTION item (subtotalling) or to an entry in a different report group (rolling forward), you will obtain at that instant the up-to-date accumulated value, a technique which is useful for obtaining brought forward and carried forward totals, as in the layout on the following page:
SPORTS CLUB CASH BOOK

01/01/83   BAR             $200.00
01/01/83   LOAN            $150.00
...                          ...
07/04/83   PETTY CASH       $50.00

CARRIED FORWARD:               $2200.50

---

SPORTS CLUB DONATIONS

BROUGHT FORWARD:               $2200.50

08/05/83   ADVERTISING     $200.00
09/06/83   RESTAURANT      $300.00
...                          ...
GRAND TOTAL:                   $5503.00

CARRIED FORWARD:               $5503.00

---

WORKING-STORAGE SECTION.
01 WS-TOT-CASH       PIC S9(4)V99 COMP.
REPORT SECTION.
...                      
01 TYPE PH.          
03 LINE 1       COL 1 VALUE "SPORTS CLUB DONATIONS".
03 LINE + 2       ABSENT AFTER NEW REPORT.
05 COL 1       VALUE "BROUGHT FORWARD:".
05 COL 25       PIC $(5)9.99   SOURCE WS-TOT-CASH.
01 CASH-ENTRY     TYPE DE           LINE.
05 ...                      
05 P-CASH    COL 26   PIC $(4)9.99   SOURCE CASH.
01 TYPE CF FINAL.       
03 LINE.            
05 COL 1       VALUE "GRAND TOTAL:".
05 TOT-CASH  COL 26   PIC $(4)9.99   SUM OF P-CASH.
01 TYPE PF          LINE 60.
05 COL 1       VALUE "CARRIED FORWARD".
05 COL 25       PIC $(5)9.99   SOURCE WS-TOT-CASH.
...                      
PROCEDURE DIVISION.
...                      
GENERATE CASH-ENTRY
MOVE TOT-CASH TO WS-TOT-CASH
We reference a SUM total field in a SOURCE to capture a snapshot of the running total. The SUM entry is gradually accumulated over a certain part of the report, so we obtain any number of versions of it at any state up to the eventual total. This, incidentally, gives us an alternative to the RESET phrase for forming cumulative (running) totals.

In the sample code we save the SUM total field procedurally in a working location (WS-TOT-CASH) after each GENERATE and use that in the SOURCE for the next cycle. This is done for two reasons:

It is independent of whether or not the option is in effect. The (standard but "less logical") option does rolling forward and subtotalling before the page-fit test (see GENERATE Processing Cycle), so our total fields will already have been updated by the new value of CASH and we would otherwise have to subtract it again in the SOURCE to get the true total. The option corrects this anomaly, but there is a second problem:

Whichever option is in effect, the total field is reset to zero immediately after its CONTROL FOOTING group has been output, so it cannot appear in the PAGE FOOTING beneath it (the last page in our example, since the group is a CF FINAL). Hence the use of WS-TOT-CASH which is the current total "suspended from the last GENERATE".

If we did not wish to print a grand total field from which to "snapshot" the values, we could define the total as an unprintable item (with no COLUMN clause).

b. **In a PRESENT WHEN clause**

You may test the value of a total field in a PRESENT WHEN's condition to control what is produced in the report group. The effective value will be the same as for the case with SOURCE above. However, if you do this, do not code any SUM clauses or any summed entries within the scope of your PRESENT WHEN clause. This avoids a deadlock situation where summing depends on conditions and conditions depend on summing. A useful example is the common requirement to suppress zero totals. The code should be as follows:

```cobol
04 TOTAL-CASH    PIC 9(4)     SUM OF CASH.
04 LINE + 2      PRESENT WHEN TOTAL-CASH NOT = ZERO.
05 COL 1         "TOTAL = ".
05 COL 11        PIC ZZZ9     SOURCE TOTAL-CASH.
```

TOTAL-CASH is an unprintable SUM entry that is - as necessary - outside the scope of the PRESENT WHEN. (If your entire LINE is subject to the PRESENT WHEN, place your unprintable field in another LINE of the same group. Since it is unprintable, its placement is immaterial.)

You may use the total field even if it is defined later in the same group because totalling is completed for each group before production of any of the report lines begins.
c. **In the Main-Line PROCEDURE DIVISION**

You may capture the contents of a subtotalled or rolled forward total field in a main-line COBOL procedural statement. As an example of the use of this property, you may move a total field into a record in a different output file. This will save you the effort and inefficiency of totalling the same field independently. However, the contents of an unconditional cross-foot total will always be zero, because it will always have been output as soon as it was totalled. (A total field is reset to zero at the end of the report group in which it is defined, unless the **RESET** phrase defers this action – see 3.23.7 [The RESET Phrase].)

You may also procedurally alter (that is, ADD TO, SUBTRACT FROM etc.) the value of any named total field at any time. Of course, if you do so the results that appear in the total fields will be different from those you would expect if report writer alone were accumulating the totals. You may also need to handle any possible size errors in the total field. The values you added or subtracted will be reset with the rest of the value at the usual time.

Total fields are cleared to zero by the **INITIATE** statement. They will also normally be zero after execution of the **TERMINATE** statement. This is because they are always cleared after being output, and the **TERMINATE** statement outputs all CONTROL FOOTING groups and any PAGE FOOTING and REPORT FOOTING. However, if a SUM clause was coded in another type of group, such as a DETAIL, that is not automatically output when your program issues the **TERMINATE**, there could be a non-zero total waiting in vain to be output. Also, if the SUM clause was conditional, the field with the SUM might have been absent on the last occasion, in which case a non-zero total might still be waiting to be output. In all these cases, report writer will issue a run time error and the total will be cleared if and when your program issues another **INITIATE** for the corresponding report.

d. **In a Declarative SECTION**

You may also access total fields in a **USE BEFORE REPORTING** Declarative SECTION for any report group. As is the case with reference to sum totals in a SOURCE, described above (see a(ii)), the state of any total fields you may access depends on whether the OSVS option is in effect or not.

Any cross-foot totals for the group will already have been calculated in all cases.

If OSVS is in effect, any rolling forward (from this group into a different group) and subtotalling (if this is a DETAIL group, or a PH or PF triggered by a DETAIL group) are done before entry to the Declarative SECTION. Hence you will obtain all totals fully updated. (See 4.2.4 [GENERATE Processing Cycle].)
If OSVS is not in effect, by contrast, any rolling forward and subtotalling are done after entry to the Declarative SECTION. So, especially if your group is a PAGE FOOTING (or HEADING), you will obtain an accurate value of the total at that “physical point” in the report.

4. Precision of SUM terms

If you use SUM or COUNT as a term in a SOURCE's expression, report writer will assign the internal total field with PICTURE S9(12)V9(6) COMP, irrespective of the PICTURE of the report field.

3.23.5 Subtotalling and SOURCE SUM Correlation

Subtotalling is the term used when your SUM clause or SOURCE clause SUM term specifies a non-REPORT SECTION operand, that is, an item in your FILE, WORKING-STORAGE, or LINKAGE SECTION. OS/VS and DOS/VS COBOL report writer, which is based on ANS-68, depends on subtotalling for forming most totals. With report writer, you should rely instead on rolling forward, by giving a data-name to a DETAIL group's SOURCE entry, so that there is no need to worry about when the items are added into the total fields. However, there are still times when subtotalling might be preferred, such as when you are creating total lines in a summary report and have not coded SOURCE items in a DETAIL group to refer to. The rules of operation are as follows:

1. Report writer adds the SUM operands (“addends”) directly into the total field, following the normal rules of the ADD statement. If there is a size error, the same action takes place as for REPORT SECTION SUM totals, as described above.

2. SOURCE SUM Correlation.

The decision as to when to add the items depends on whether SOURCE SUM correlation is in effect. This is the chief difference between the ANS-68 standard used in OS/VS and DOS/VS COBOL, which automatically applies SOURCE SUM correlation, and the ANS-74 and ANS-85 standards, which do not. You will observe a difference only when the following circumstances occur simultaneously:

a. When your report has more than one DETAIL group; and
b. When your program SUMs an operand that is also a non-REPORT SECTION SOURCE operand in one or more of the DETAIL groups.

Otherwise, report writer will not apply SOURCE SUM correlation. SOURCE SUM correlation is in effect if:

c. The precompiler was installed with the OSVS option set on, or
d. You code an ALLOW SOURCE SUM CORR clause in your RD.

SOURCE SUM correlation is not in effect if:

e. Your system was installed with the OSVS option set off (ANS74 or ANS85), or,
f. You code an ALLOW NO SOURCE SUM CORR clause in your RD.
3. If SOURCE SUM correlation is in effect, report writer takes each non-REPORT SECTION operand specified in your SUM clause and scans the DETAIL groups of your report to establish whether the same item is coded in more than one of them as a SOURCE operand. (The SOURCE keyword may, as usual, have been omitted.) Arithmetic expressions are not examined. Redefinitions of the same item and differences in subscripts or qualifiers (apart from interchanging the words IN and OF) mean that there is no match. All your DETAIL groups are scanned in this way. The appearance of the same operand more than once as a SOURCE in the same DETAIL counts as only a single match. If a match is found, report writer will add the item into the total field only when the DETAIL or DETAILS that contain your item are GENERATED. If no match is found, report writer will add the item into the total field on every GENERATE for that report.

4. Using a dummy DETAIL group for Summary Reporting.

With the ANS-68 standard of OS/VS and DOS/VS COBOL, a corresponding SOURCE was required for every SUM. This meant that in a totals only report, a dummy DETAIL group needed to be defined, as in the following sample:

```
RD  SUMMARY-REPORT
  CONTROL IS STATE.
01  DUMMY-GROUP TYPE DETAIL.
    05  SOURCE IS WS-PAY-THIS-YEAR.
    05  SOURCE IS WS-PAY-LAST-YEAR.
01  TYPE CF FOR STATE.
  03  LINE PLUS 2.
    05  COLUMN 1   PIC ZZZ9  SUM WS-PAY-THIS-YEAR.
    05  COLUMN 11  PIC ZZZ9  SUM WS-PAY-LAST-YEAR.
    ...
  GENERATE DUMMY-GROUP
```

Although report writer accepts this code unaltered, the dummy group is now unnecessary - even if SOURCE SUM correlation is in effect. The dummy group in the example above may be deleted and the GENERATE replaced by GENERATE report-name:

```
RD  SUMMARY-REPORT
  CONTROL IS STATE.
01  TYPE CF FOR STATE.
  03  LINE PLUS 2.
    05  COLUMN 1   PIC ZZZ9  SUM WS-PAY-THIS-YEAR.
    05  COLUMN 11  PIC ZZZ9  SUM WS-PAY-LAST-YEAR.
    ...
  GENERATE SUMMARY-REPORT
```

More details will be found under Summary Reporting (see GENERATE statement).

5. If SOURCE SUM correlation is not in effect, report writer will add every operand of the SUM clause into the total field on every GENERATE for the report.
3.23.6 Three Methods of Subtotalling

In the items just preceding, you observe that, if SOURCE SUM correlation is not in effect, adding into your total field will take place on each GENERATE for the report. This may not be suitable, especially when you have several DETAIL groups. If you have two record types, say SALARY and NAME-ADDRESS, with DETAIL groups that correspond to these, and you want the total of SALARY, you will clearly want the field SALARY to be added only when your program generates the SALARY DETAIL group. There are three ways of avoiding this problem, which are illustrated below.

1. The recommended method of report writer is to place a data-name on the entry to be totalled (omitting the COLUMN clause if it is not be printed), and SUM the item using that data-name.

2. If SOURCE SUM correlation is in effect and both the SOURCE operand and the SUM operand names are identical, the item will be added, as expected, only when the SALARY group is GENERATED.

3. Using the UPON phrase. You may use this whether or not SOURCE SUM correlation is in effect, because UPON overrides its effect. By writing UPON SALARY-GRP (where SALARY-GRP is the 01-level name of your DETAIL group), you ensure that the item will be added into the total only when the SALARY-GRP DETAIL group is GENERATED.

The next example compares these methods:

```
SPORTS CLUB OFFICERS: SALARIES

NAME: J.C. CODER        SALARY:  $10000
BANK: BARCLAYS
      1 FARNHAM ROAD, TENTERDEN
NAME: T.A. ANALYST      SALARY:  $12000
BANK: ...

TOTAL SALARIES:   $89000
------------------------
```

<- SALARY group

<- NAME-ADDRESS group
a. **Method Using REPORT SECTION SUM** (recommended)

```plaintext
01 SALARY-GRP    TYPE DE   LINE + 1.
    .
  05 R-SALARY    COL 50   PIC $(6)9   SOURCE SALARY.
01 NAME-ADDRESS  TYPE DE   ...
  01 CF         ...
    .
  05 COL 49     PIC $(7)9    SUM OF R-SALARY.
```

b. **Using SOURCE SUM Correlation**:

```plaintext
RD    ...
*Following may be omitted if set on customization.
  ALLOW SOURCE SUM CORR.
01 SALARY-GRP    DE   LINE + 1.
    .
  05 COL 50        PIC $(6)9     SOURCE SALARY.
01 NAME-ADDRESS  TYPE DE   ...
  01 CF         ...
    .
  *Correlation between SOURCE and SUM.
  05 COL 49     PIC $(6)9    SUM OF SALARY.
```

c. **Using the UPON Phrase**:

```plaintext
01 SALARY-GRP    TYPE DE   LINE + 1.
    .
  05 COL 50        PIC $(6)9     SOURCE SALARY.
01 NAME-ADDRESS  TYPE DE   ...
  01 TYPE CF      ...
    .
  *UPON phrase indicates "ADD only when SALARY-GRP generated".
  05 COL 49     PIC $(6)9    SUM OF SALARY UPON SALARY-GRP.
```
The **RESET Phrase**

In all our examples presented earlier in this section, the total field is reset to zero at the end of the processing for the group in which it has appeared. Sometimes you will not want this to happen. Such a case is called a cumulative (or running) total. Here the total is not necessarily cleared or reset after it has been output. Values then continue to be added into the total. You will have seen above that you can capture cumulative totals by "taking a snapshot" of a higher-level total field. Another method is to use the **RESET** phrase. Its operand states at which (normally higher) level of control break, if any, the total field is to be cleared. If you specify **RESET ON REPORT** (or **RESET ON FINAL**), the total field will not be reset after the final control footing has been presented, during the execution of the **TERMINATE**.

For example, by writing:

```cobol
03 COL 20 PIC ZZZ9.99   SUM OF ACC-BAL   RESET ON BRANCH-CODE.
```

you ensure that the total field is not cleared until a new BRANCH-CODE is reached.

Here is an example of a cumulative total:

<table>
<thead>
<tr>
<th>DATE</th>
<th>AMOUNT</th>
<th>CUMULATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>010270</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>010370</td>
<td>150.00</td>
<td></td>
</tr>
<tr>
<td>TOTALS JAN</td>
<td>350.00</td>
<td>350.00</td>
</tr>
<tr>
<td>020770</td>
<td>50.00</td>
<td></td>
</tr>
<tr>
<td>TOTALS FEB</td>
<td>50.00</td>
<td>400.00</td>
</tr>
<tr>
<td>030370</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>160370</td>
<td>40.00</td>
<td></td>
</tr>
<tr>
<td>270370</td>
<td>60.00</td>
<td></td>
</tr>
<tr>
<td>TOTALS MAR</td>
<td>200.00</td>
<td>600.00</td>
</tr>
<tr>
<td>...etc...</td>
<td>......</td>
<td>......</td>
</tr>
<tr>
<td>050171</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>270171</td>
<td>200.00</td>
<td></td>
</tr>
<tr>
<td>TOTALS JAN</td>
<td>300.00</td>
<td>300.00</td>
</tr>
</tbody>
</table>

RD ...
CONTROLS YEAR MONTH.
01 CF FOR MONTH.

```cobol
05 COL 21 PIC ZZZ9.99   SUM OF AMOUNT.
05 COL 33 PIC ZZZ9.99   SUM OF AMOUNT   RESET ON YEAR.
```
3.23.8 Compatibility

1. The following features are unique to new Report Writer:
   - Optional word OF after keyword,
   - SUM operand can be an expression formed from non-REPORT SECTION items,
   - SUM as a term in an expression in a SOURCE clause,
   - SUM clause in report groups other than of TYPE CONTROL FOOTING,
   - The ROUNDED phrase.

2. OS/VS and DOS/VS COBOL Report Writer, in common with ANS-68 and ANS-74 requires that any REPORT SECTION item whose data-name is the operand of a SUM must itself be a SUM entry, that is, only SUM entries may be rolled forward or used for cross footing. OS/VS and DOS/VS COBOL Report Writer always assumes SOURCE SUM correlation. It uses the ANS-68, not the ANS-74, rules.

3. The automatic check for overflow is unique to new Report Writer. A migrated program which produced truncated totals will now produce blank fields instead, plus run time error message 11. In the unlikely event that this truncation is intentional, the clause SUM OVERFLOW PROCEDURE IS OMITTED should be placed in the RD.
3.24  **TYPE clause**

This clause is used at the 01-level to indicate which of the seven possible types of report group is being defined.

### 3.24.1 Format

```
  TYPE IS
  REPORT HEADING
  PAGE HEADING
  CONTROL HEADING
  DETAIL
  CONTROL FOOTING
  PAGE FOOTING
  REPORT FOOTING
```

where control-id is an identifier in the CONTROL(S) clause, or the word **REPORT** or **FINAL**.

### 3.24.2 TYPE Clause: Coding Rules

1. Write the TYPE clause at the 01-level entry only. You may omit the keywords **TYPE IS**. If you do not code a TYPE clause for a group, it will be assumed to be **TYPE DETAIL**.

2. All TYPES of report group are optional, whatever the circumstances, but every report must have at least one body group (CONTROL HEADING, DETAIL, or CONTROL FOOTING).

3. **PAGE HEADING**, **PAGE FOOTING** and the **OR PAGE** phrase of **CONTROL HEADING** groups are allowed only if you have a **PAGE LIMIT clause** in the RD.

4. In the **CONTROL HEADING** and **CONTROL FOOTING** forms each control-id operand must be chosen from the list of controls in your **CONTROL** clause, including **REPORT** (or **FINAL**) which is always assumed present.

5. If **CONTROL HEADING** is coded with no control-id operand, there must not be more than one control-id in the CONTROL(S) clause. The clause is then taken to mean **CONTROL HEADING FOR control-id**. If there is just one control-id in the CONTROL(S) clause, or **CONTROL HEADING FOR REPORT** if there is no CONTROL(S) clause at all.
6. If more than one (different) control-id is coded in the CONTROL FOOTING form, the result is a set of multiple CONTROL FOOTING report groups. The control-ids need not be coded in any particular order and need not form a consecutive hierarchic sequence. For example, if the CONTROLS clause is:

```
CONTROLS ARE STATE CITY STREET
```

it is permissible to code:

```
01 TYPE CF FOR STATE, STREET.
```

which means the same as

```
01 TYPE CF FOR STREET, STATE.
```

If a CONTROL FOOTING is required for all levels of control, ALL may be coded instead of the exhaustive list of control-ids.

If a DETAIL group also has the same format as some CONTROL FOOTINGs, it is usually worth the effort to define it as the lowest-level CONTROL FOOTING group, and force a lowest-level control break on each GENERATE, in order to make full use of this feature. In this case summary reporting is used (see 4.2 GENERATE statement) and each totalled value, generated by a SUM clause, will be the "total" of just a single value in the case of the lowest CONTROL FOOTING.

7. If CONTROL FOOTING is coded with no control-id operand, it is taken to mean CONTROL FOOTING FOR ALL, or, CONTROL FOOTING FOR control-id, if there is just one control-id in the CONTROL(S) clause, or CONTROL FOOTING FOR REPORT if there is no CONTROL(S) clause at all.

8. In any report you may have any number of DETAIL report groups, but only one PAGE HEADING, PAGE FOOTING, REPORT HEADING, REPORT FOOTING, and each control-id may appear in only one CONTROL HEADING and only one CONTROL FOOTING group. If CONTROL FOOTING FOR ALL is coded, it must be the only CONTROL FOOTING in the report.

9. The physical order of report group descriptions is irrelevant. For example, the PAGE HEADING group need not necessarily appear before the DETAIL and PAGE FOOTING. Report writer produces them in the correct sequence according to the rules for page-fit and control break testing. However, for ease of maintenance, the order implied in the format above is recommended. Also, it is helpful if CONTROL HEADING groups are coded in major-to-minor order and CONTROL FOOTING groups in minor-to-major order, paralleling the sequence in which these groups will be presented in the report.

10. In a report with a PAGE LIMIT clause, if your group contains any absolute LINE clauses, report writer will check that each line of the group will lie within the region of the page appropriate to the TYPE of the group. Refer to the diagram of the regions of the page (see 2.9.3 PAGE LIMIT Clause: Operation). REPORT HEADING and REPORT FOOTING groups may appear anywhere on the page from the HEADING position onwards. If your group contains only relative LINE clauses, and the report has a PAGE LIMIT clause, report writer will check at compilation time that your group is not larger than the appropriate region of the
All the CONTROL HEADING groups with an OR PAGE phrase must be able to fit on the page above any other DETAIL or CONTROL HEADING group. Stated precisely: (a) each DETAIL or other CONTROL HEADING must either have only relative LINE clauses or must begin with an absolute LINE which is higher than the highest possible line number produced by the CONTROL HEADING groups with an OR PAGE phrase, and (b) the LASTDETAIL value must be sufficient to accommodate the largest such combination.

3.24.3 Type Clause: Operation

1. You use the TYPE clause to indicate, implicitly, where and how your group is to be produced in the report. Here is a summary of how each TYPE is handled:

   **REPORT HEADING**
   This group will appear once, at the very start of the report. (See 4.2.3 GENERATE Statement: Operation.)

   **PAGE HEADING**
   This group will be produced as the first group in each page. (See 3.10.3 LINE Clause: Operation.)

   **CONTROL HEADING**
   This group will appear automatically at the start of each different actual value of the corresponding control. (See 2.6 CONTROL clause.) If you code the OR PAGE phrase, the report group will also be triggered by a page advance. See 3.24.4 below for full details.

   **DETAIL**
   All the remaining report groups that are not one of the other six types are of TYPE DETAIL. DETAIL groups are the only report groups that can be GENERATED explicitly by the program. The remaining six TYPES of report group are produced automatically whenever necessary before the DETAIL group is processed. (Note that with summary reporting, the other six TYPES are the only ones that can be produced. See 4.2.2 GENERATE Statement: Coding Rules.)

   **CONTROL FOOTING**
   This group will appear automatically at the end of each different actual value of the corresponding control or controls. (See 2.6 CONTROL clause.)

   **PAGE FOOTING**
   This group will be produced as the last group in each page (except on a page occupied only by a REPORT HEADING and immediately after a REPORT FOOTING). (See 3.10.3 LINE Clause: Operation.)

   **REPORT FOOTING**
   This group will appear once, at the very end of the report. (See 4.6 TERMINATE statement.)
2. **RH and PH together**

If your report contains **both** a **REPORT HEADING** and a **PAGE HEADING** group, report writer will attempt to place them **both** in the Page Heading region of the page. If it can do this without overlap, without overflowing the region, and without violating any of the above rules, then your REPORT HEADING group will be produced on the **first page** of the report, **above the PAGE HEADING**. If the PAGE HEADING has **relative LINE** clauses, it begins relative to the last line of the REPORT HEADING. If, despite this, you require the REPORT HEADING to appear on a page by itself, you should code **NEXT GROUP NEXT PAGE** in the 01-level entry of the REPORT HEADING, to suppress this check.

If you want the REPORT HEADING to appear on the first page above the PAGE HEADING and "push it down" so that your DETAIL groups start lower on the first page than they do on the remaining pages, you should **omit the FIRST DETAIL** sub-clause. Your DETAIL groups will then begin on the **line following the PAGE HEADING**. To get extra space before your first DETAIL group, you may define a blank LINE entry at the end of your PAGE HEADING:

```
RD ... *no FIRST DETAIL clause
PAGE LIMIT 60.
01 TYPE RH.
  03 LINE 1.          VALUE "END OF YEAR REPORT".
  03 LINE + 1.         VALUE "******************".
  03 LINE + 2.         => blank lines for first sheet layout
01 TYPE PH.
  03 LINE + 1.         VALUE "WOLFITDOWN PETFOODS".
  03 LINE + 2.         => blank lines before DETAILS begin
```

You may also achieve the same result **without using a REPORT HEADING**, by including **all** the lines in a **single PAGE HEADING** and using a **PRESENTER AFTER REPORT** clause (see 3.17 **PRESENTER AFTER clause**).

3. **PF and RF together**

If your report contains **both** a **PAGE FOOTING** and a **REPORT FOOTING** group, report writer will attempt to place them **both** in the Page Footing region of the page. If it can do this without overlap, without overflowing the region, and without violating any of the above rules, then your REPORT FOOTING group will be produced on the last page of the report, **below the PAGE FOOTING**. If the REPORT FOOTING has **relative LINE** clauses, it begins **relative to the last line of the PAGE FOOTING**. If, despite this, you require the REPORT FOOTING to appear on a page by itself, you should code an **ON NEXT PAGE** phrase in the first LINE clause of the REPORT FOOTING, to suppress this check.
3.24.4 The OR PAGE Phrase of the CONTROL HEADING

You may add the **OR PAGE** phrase after the **control-id** operand for **TYPE CONTROL HEADING**. This causes your CONTROL HEADING to be produced at the **top of each page**. This enables you to repeat essential "key" information after your regular PAGE HEADINGS. The precise rules of operation are as follows:

1. If your report contains any such **TYPE CH** groups with the **OR PAGE** phrase, the **CONTROL HEADING** group will be presented after a control break at the relevant level, exactly as when the **PAGE** option is not present, but in addition, the actions on page advance processing are modified as follows:
   a. If a DETAIL group causes a page advance then, after the usual page advance has taken place, the CONTROL HEADING group is printed. If more than one **TYPE CH** group has the **OR PAGE** option, these CONTROL HEADING groups are printed in hierarchic order, from highest to lowest. Each CONTROL HEADING is, of course, printed on the new page, irrespective of whether or not it would fit on the previous page.
   b. If a CONTROL HEADING group causes a page advance, the same action occurs as in (a) above, except that the CONTROL HEADING group that caused the page advance is output only once on the new page, whether it has the **OR PAGE** phrase or not.
   c. If a CONTROL FOOTING group causes a page advance, the same action occurs as for a DETAIL except that no CONTROL HEADINGS below the level of the CONTROL FOOTING are printed.

2. If a group has **CH FOR PAGE** without a **control-id**, it will be treated as equivalent to **CH FOR PAGE ** OR REPORT**.

3. The following shows the layout you might require:

```
** HEADING **     PAGE 1
YEAR: 1991
MONTH: 01
........   20
........   30
TOTAL FOR MONTH:   50
MONTH: 02
........   10
........   20
TOTAL FOR MONTH:   30
MONTH: 03
........   20
........   30
........   40
(A)

** HEADING **     PAGE 2
YEAR: 1991
MONTH: 03
........   50
TOTAL FOR MONTH:   140
MONTH: 09
........   20
........   30
TOTAL FOR MONTH:   50
MONTH: 11
........   20
........   40
........   10
TOTAL FOR MONTH:   70
```
Note the following labelled points in the diagram:

(A) Because of the ** OR PAGE ** phrases, both CH groups re-appear at the top of the page even though neither control has changed.

(B) When a control break occurs and the corresponding CH group will not fit on the current page, it appears once only at the top of the new page, even though it has an ** OR PAGE ** phrase.

(C) If a ** CF ** group causes a page advance, the CH groups are produced at the top of the next page, but only those at the same level or above that of the CF group.

The following is the report writer code required to produce this layout:

```
RD REP-ONE
PAGE LIMIT 40
CONTROLS ARE YEAR, MONTH.

01 PH                  LINE 1.
  05 COL 1           "** HEADING **".
  05 COL 19          "PAGE".
  05 COL 21          PIC Z9   SOURCE PAGE-COUNTER.

01 CH FOR YEAR OR PAGE      LINE + 1.
  05 COL 1           "YEAR:"
  05 COL + 2         PIC 9(4)         SOURCE YEAR.

01 CH FOR MONTH OR PAGE     LINE + 1.
  05 COL 1           "MONTH:"
  05 COL + 2         PIC 99           SOURCE MONTH.

01 REP-DET TYPE DE         LINE + 1.
  05 COL 1           "........".
  05 PVAL   COL + 2   PIC ZZZ9         SOURCE WVAL.
```
01 TYPE CF FOR MONTH LINE + 1.
  05 COL 1 "TOTAL FOR MONTH:".
  05 M-TOT COL + 2 PIC ZZZ9 SUM OF PVAL.

01 TYPE CF FOR YEAR LINE + 1.
  05 COL 1 "TOTAL FOR YEAR:".
  05 COL + 2 PIC ZZZ9 SUM OF M-TOT.

3.24.5 Compatibility

All aspects of the following features are unique to new Report Writer:

- Allowing the TYPE keyword to be **omitted**.
- The optional words **ON** and **FOR**
- The **OR PAGE** phrase of CONTROL HEADING,
- Default of DETAIL if the TYPE clause is omitted.
- The multiple CONTROL FOOTING option.
3.25 USAGE clause

This clause is allowed for documentary purposes in the REPORT SECTION, for consistency with basic COBOL, or to emphasize that an entry is DBCS.

3.25.1 Format

```
USAGE IS DISPLAY
DISP
DISPLAY-1
```

3.25.2 USAGE Clause: Coding Rules

1. The USAGE clause may be coded at any level, but no item may be subject to both USAGE DISPLAY and USAGE DISPLAY-1.

2. DISP is synonymous with DISPLAY.

3. Only non-DBCS items may be subject to USAGE DISPLAY and only DBCS items may be subject to USAGE DISPLAY-1.

4. No other forms of the USAGE clause are permitted in the REPORT SECTION.

3.25.3 USAGE Clause: Operation

1. DISPLAY is retained for consistency with basic COBOL but it is never required.

2. USAGE DISPLAY-1 indicates that the item (or items if on a group level) is DBCS (Double-Byte Character Set), such as Japanese Kanji. However, it is not required in the REPORT SECTION, since it is implied by the presence of a DBCS PICTURE string (containing the symbols "G" and "B" only) or a DBCS literal, of the form G"so...si" or G"so...si" where so and si are the shift-out and shift-in characters. DBCS items are stored with a shift-out character on the left and a shift-in on the right. Each double-byte character occupies one print column position even though it takes up two bytes of memory. COLUMN numbers (absolute or relative) take this into account. Spaces inserted between DBCS items are the regular (non-DBCS) space.

3.25.4 Compatibility

3.26 **VALUE clause**

This clause may be used whenever the report field to be output consists of a fixed literal value.

**3.26.1 Format**

```
+-----------------+
| VALUE IS literal-1 |
+-----------------+
```

**3.26.2 VALUE Clause: Coding Rules**

1. You may specify a **numeric** or **non-numeric** literal, including a figurative constant, or a DBCS literal. An ANSI-85 SYMBOLIC CHARACTER is also permitted.

2. Unless you specify **ALL** or a figurative constant, you do **not** need a PICTURE clause. For example:

   ```cobol
   05   COL 21    VALUE "*** ALL-IN SPORTS CLUB ***".
   ```

   You may use 'apostrophes' instead of "quotes" if your current compiler options expect them. The Precompiler accepts either delimiter at the start of a literal, scanning for the same delimiter to close the literal. Continued literals are also permitted. As usual, two quotes juxtaposed within a "literal" signify a single quote as part of the value, and similarly for apostrophe.

3. You may use the **ALL "literal"** form or a numeric literal or figurative constant, but in all these cases you must specify a PICTURE. Here is an easy method of coding a repeated value:

   ```cobol
   05   COL 1       PIC X(24)     VALUE ALL "XOX".
   ```

   which gives you the repeated pattern: XOXOXOXOXOXOXOXOXOXOX

4. If the item is defined as DBCS by virtue of its PICTURE clause or USAGE DISPLAY-1 clause, the literal must also be DBCS. However a PICTURE clause is **not** required, a PICTURE of **G(n)** where **n** is the number of double bytes being assumed in default.

5. Any literal may also be **hexadecimal**. However, it is inadvisable to use this facility to insert printer control characters into your print data, since these will (a) make your program **non-portable** and unreadable (b) put your COLUMN positions out of alignment. The **STYLE** clause is designed specifically for this purpose and has none of these drawbacks. (See 3.22 **STYLE clause**.)
3.26.3 VALUE Clause: Operation

1. The VALUE clause results in the specified fixed literal appearing in your report field. Assuming that your program does not alter the value by overwriting the report field procedurally (which it can do if the field is named), the value will be unchanged throughout the report.

2. Report writer will either "pre-set" (fill in) your report field with the specified literal at compile time; or it may move the literal into the report field at run time, in cases where the report field is in a variable position, or where the report line cannot hold pre-set values because it is subject to an OCCURS clause.

3. If the item is DBCS, it is stored in the report line with each double-byte character occupying one column position. (See 3.25 USAGE clause.)

3.26.4 Multiple VALUES

If you use the multiple form of the VALUE clause, by writing more than one literal after the keyword, it will save you the effort of coding several separate entries. Note the following:

1. If you wish to place no value in a particular occurrence, you may simply code a space character: " ".

2. Your entry must be subject to at least one of the following:
   a. A fixed OCCURS clause (not OCCURS...DEPENDING), or
   b. A multiple LINES clause, or
   c. A multiple COLUMNS clause.

3. All the literals must be either DBCS or non-DBCS.

4. The rule for the number of literals allowed in your multiple VALUE is similar to that of the multiple SOURCES clause (see 3.21.4 Multiple SOURCES); that is, it must exactly equal either the total number of repetitions in all the dimensions of the entry, or the product of the numbers of repetitions of one or more of the inner dimension(s). For example, with the following layout:

   03 LINE OCCURS 2.
   04 OCCURS 3.
   05 COLS +3, +3, +3, +1       VALUES ARE .....  

   the number of literals should be either 4 (just the inner dimension), 12 (the product of the inner two dimensions), or 24 (all the dimensions).

5. If the literals cover more than one dimension, they are distributed along the innermost dimension, periodically stepping to the next entry in one or more outer dimensions. For example:

```
03 LINES 2    STEP 1    OCCURS 3.
05 COL 2    STEP 5    OCCURS 4
VALUES "JAN" "FEB" "MAR" "APR"
 "MAY" "JUN" "JUL" "AUG"
 "SEP" "OCT" "NOV" "DEC".
```

will result in:

```
<table>
<thead>
<tr>
<th>JAN</th>
<th>FEB</th>
<th>MAR</th>
<th>APR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAY</td>
<td>JUN</td>
<td>JUL</td>
<td>AUG</td>
</tr>
<tr>
<td>SEP</td>
<td>OCT</td>
<td>NOV</td>
<td>DEC</td>
</tr>
</tbody>
</table>
```
6. If there are **two or more dimensions** and the number of terms matches only the **inner** dimension(s), the series of literals is **re-cycled** from the first operand for each of the outer repetitions. For example, the following case:

<table>
<thead>
<tr>
<th>03 LINE OCCURS 4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 COLS 1 13 VALUES &quot;THIS YEAR&quot; &quot;SOME TIME&quot;.</td>
</tr>
</tbody>
</table>

will result in:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>THIS YEAR</td>
<td>SOME TIME</td>
</tr>
<tr>
<td>THIS YEAR</td>
<td>SOME TIME</td>
</tr>
<tr>
<td>THIS YEAR</td>
<td>SOME TIME</td>
</tr>
<tr>
<td>THIS YEAR</td>
<td>SOME TIME</td>
</tr>
</tbody>
</table>

7. The **multiple VALUE operand** may be used as one or more of the alternatives in a **multiple-choice** entry. You need not use a multiple VALUE in every alternative:

<table>
<thead>
<tr>
<th>05 COLS 1 12 25 VALUES &quot;BLEU&quot; &quot;ROUGE&quot; &quot;JAUNE&quot; WHEN LANGUAGE = &quot;F&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BLAU&quot; &quot;ROT&quot; &quot;GELB&quot; WHEN LANGUAGE = &quot;G&quot;</td>
</tr>
<tr>
<td>&quot;BLUE&quot; &quot;RED&quot; &quot;YELLOW&quot; WHEN LANGUAGE = &quot;E&quot;</td>
</tr>
<tr>
<td>&quot;????&quot; WHEN OTHER.</td>
</tr>
</tbody>
</table>

8. If, as is usual, you **omit** a PICTURE clause, the size of each field is the size of its corresponding literal, as seen in the following example:

| 05 COLS 1 +2 +2 VALUES "CAESAR" "QUELLED" "VERCINGETORIX". |

which yields:

| CAESAR QUELLED VERCINGETORIX |

This is also true of multiple-choice entries.

9. In all cases, you may **omit** the VALUE keyword.

10. Other examples of the multiple VALUE clause will be found under 3.4.4 **Multiple COLUMNS Clause** and 3.10.4 **Multiple LINES Clause**.

### 3.26.5 Compatibility

Only new Report Writer provides the following features:

- The VALUE keyword being optional,
- **DBCS** and **hexadecimal literals** in the REPORT SECTION,
- PICTURE clause being **optional** with a non-numeric literal,
- The **multiple VALUE format**.
3.27  VARYING clause

This clause enables you to vary the value of a numeric counter (typically for use elsewhere as a subscript) during the production of a repeating field.

3.27.1

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶►VARYING data-name ▼</td>
</tr>
<tr>
<td>FROM expression-1</td>
</tr>
</tbody>
</table>

3.27.2  VARYING Clause: Coding Rules

1. You may write any number of different data-name operands in this clause, each with an optional associated FROM and BY phrase.

2. Your entry must also have either an OCCURS clause (see 3.14) or a Multiple COLUMNS Clause (see 3.4.4) or a Multiple LINES Clause (see 3.10.4).

3. Your data-names must not be defined already anywhere else in the program and you should not attempt to define them separately. Report writer creates a description for them itself, internally. (This is similar to the way COBOL handles index-names.)

4. You can re-use the same data-names in different VARYING clauses, provided that you do not do this when the clauses are nested (enclosed one within the other). For example, you could write VARYING R-LINE on each repeating LINE, and VARYING R-COL on each repeating COLUMN, throughout your program.

5. If you intend FROM 1 you may omit the FROM phrase and report writer will infer it. Likewise, if you intend BY 1 you may omit the BY phrase and report writer will infer it. (FROM 1 and BY 1 are the most usual requirements, so these assumptions are convenient.)

6. Each expression may be any integer, or an identifier, or an arithmetic expression, provided that the result has an integer value. The expression may contain data-names of an enclosing VARYING clause. It can also use a data-name of the same VARYING clause, but only in its BY expression. It must not contain data-names of an enclosed VARYING clause, or of any other VARYING clause. Thus the following are legal:

   a. 05 OCCURS 3 VARYING R-MONTH.
      07 OCCURS 4 VARYING R-INDEX FROM R-MONTH.

   b. 05 OCCURS 3 VARYING R-MONTH FROM 1 BY (R-MONTH + 1).
but the following are illegal:

c. 05 OCCURS 3 VARYING R-INDEX FROM R-MONTH.
    07 OCCURS 4 VARYING R-MONTH.
    ...

d. 05 OCCURS 3 VARYING R-MONTH FROM (R-MONTH + 1).

e. 05 OCCURS 3 VARYING R-MONTH.
    ...
    05 OCCURS 3 VARYING R-MONTH FROM R-MONTH.

3.27.3 VARYING Clause: Operation

1. When report writer is about to produce the first occurrence, it places the FROM value in an internal named data item set up implicitly by the VARYING clause. This is repeated, in the order given in the clause, for any additional data-names that may have been specified in the VARYING clause.

2. When report writer is about to produce each of the remaining occurrences, it adds the BY value to the data item. This is also repeated, in the order given in the clause, for any additional data-names that may have been specified in the VARYING clause.

3. The VARYING clause enables you to produce different source-items or values in successive appearances of a repeated field. Here are some examples:

   a. To generate the numbers 1 through 10 in a line:

   1 2 3 4 5 6 7 8 9 10

   03 LINE.
   05 COL + 5 PIC Z9 OCCURS 10 VARYING COL-INDEX SOURCE COL-INDEX.

   b. To output a two-dimensional array in Working-Storage into a two-dimensional array in your report (for example daily costs for four seven-day weeks):

   | WEEK 1 | $19 | $230 | $34 | $56 | $378 | $270 | $9  |
   | WEEK 2 | $340| $236 | $43 | $23 | $248 | $0  | $354|
   | WEEK 3 | $120| $134 | $58 | $442| $98  | $739 | $121|
   | WEEK 4 | $39 | $0  | $800| $344| $801 | $89  | $387|

   03 LINE OCCURS 4 VARYING WEEK-NO.
   05 COL 1 "WEEK".
   05 COL 6 PIC 9 SOURCE WEEK-NO.
   05 COL 9 PIC $$9 OCCURS 7 STEP 6 VARYING DAY-NO SOURCE WS-VALUE (WEEK-NO DAY-NO).
c. You could display each week's entries from right to left by writing:

```
VARYING DAY-NO FROM 7 BY -1.
```

d. Now let's display the same entries, except that they are all held in a one-dimensional array of twenty-eight entries. (This example is important.)

```plaintext
03 LINE OCCURS 4 VARYING START-DAY-NO FROM 1 BY 7,
   WEEK-NO FROM 1 BY 1.
05 COL 1 "WEEK".
05 COL 6 PIC 9 SOURCE WEEK-NO.
05 COL 9 PIC $$$9 OCCURS 7 STEP 6
   VARYING DAY-NO FROM START-DAY-NO BY 1
   SOURCE WS-VALUE (DAY-NO).
```

As each week is processed, START-DAY-NO takes values 1, 8, 15 and 22. DAY-NO takes the seven values 1 to 7, then 8 to 14, then 15 to 21, then 22 to 28 in turn, each time starting with the new value of START-DAY-NO. In the clause `VARYING START-DAY-NO FROM 1 BY 7`, the phrase `FROM 1` could have been omitted and in the clause `VARYING DAY-NO FROM START-DAY-NO BY 1`, the phrase `BY 1` could have been omitted.

e. To "fold round" a large array in boustrophedon ("as the ox turns") sequence (if NO-ACROSS is the horizontal repeat factor):

```plaintext
03 LINE OCCURS ... TIMES
   VARYING INIT FROM 1 BY ((NO-ACROSS - 1) * INCR) + NO-ACROSS,
   INCR FROM 1 BY (- INCR - INCR).
05 ... OCCURS 0 TO 100 TIMES DEPENDING ON NO-ACROSS ...
   VARYING SUBSCRIPT FROM INIT BY INCR
   SOURCE TABLE-ENTRY (SUBSCRIPT).
```
f. To produce a **pyramid-shaped** design:

```
***
 ****
 *****
******
*******
********
```

```cobol
03 LINE OCCURS 6 VARYING SUB-1 FROM 5 BY -1  
     SUB-2 FROM 1 BY 2.
05 COL + 1 OCCURS 0 TO 5 DEPENDING ON SUB-1.
05 COL + 1 VALUE "*" OCCURS 1 TO 11 DEPENDING ON SUB-2.
```

g. Note that VARYING can also be used with a multiple COLUMN or LINE, as the following example shows:

```cobol
03 LINES 2 3 4 VARYING LINE-INDEX.
05 COLS 10 20 31 52 VARYING COL-INDEX PIC ZZZ9  
SOURCE SALES (LINE-INDEX COL-INDEX).
```

4. To set up a "running index" which continues each time from its **latest** value, do not code something like `VARYING R-WEEK FROM R-WEEK + 1`, but calculate the starting value explicitly.

5. To give your counter a series of values, say `W-CNT(1), W-CNT(2)`, which are not formed by simple incrementing, write:

```cobol
VARYING R-MOD FROM 0 BY 1  
R-COUNTER FROM W-CNT(1) BY W-CNT(R-MOD) - R-COUNTER
```

6. By experimenting with the VARYING clause, you will discover many novel and surprising uses.

### 3.27.4 Compatibility

The VARYING clause is unique to new Report Writer.
3.28 **WRAP clause**

The WRAP clause is used to produce an automatic wrap round to a new continuation line when the next field will not fit on the current line.

### 3.28.1 Format a

```
<table>
<thead>
<tr>
<th>►►─WITH WRAP─►◄</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN</td>
</tr>
<tr>
<td>integer-1</td>
</tr>
<tr>
<td>identifier-1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TO COLUMN</td>
</tr>
<tr>
<td>integer-2</td>
</tr>
<tr>
<td>identifier-2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>STEP integer-3 LINES─►◄</td>
</tr>
</tbody>
</table>
```

### 3.28.2 WRAP Clause: Coding Rules

1. The **AFTER** phrase gives the rightmost column number that any field may occupy before wrap round becomes necessary. If `integer-1` is specified, it must lie in the range 1 to maximum line width. Its value acts as a maximum line width for any lines in its scope. The rightmost column position of every entry with an `absolute COLUMN` must therefore **not** exceed `integer-1`. If `identifier-1` is specified, a similar check is made at run time. If the phrase is omitted, a value of the **LINE LIMIT** is assumed for `integer-1` and, if the `identifier` form of LINE LIMIT is in use, its value will be computed, as usual, at **INITIATE** time. Thus, by default, lines are allowed to reach the usual page width before wrap round.

2. The **TO** phrase gives the column number at which printing continues after wrap round. If `integer-2` is specified, it must lie in the range 1 to maximum line width. If `identifier-2` is specified, a similar check is made at run time. If the phrase is omitted, a value of 1 is assumed for `integer-2`. Thus, by default, a **line wraps round to column 1**.

3. The **STEP** phrase gives the relative vertical offset for any continuation lines. If the phrase is omitted, a value of 1 is assumed for `integer-3`. Thus, by default, a line is continued onto the **immediately following line**. If a **PAGE LIMIT clause** is present in the RD, the value of `integer-3` is used by the precompiler in calculating the (maximum) vertical size of the group to check that it will fit correctly into its assigned region of the page.

4. WRAP may be coded either (a) **in an entry containing a LINE clause** or (b) **at a higher level** having one or more LINE entries beneath it. This second possibility allows you to avoid repeating the same clause in several LINE entries.
5. Only relative COLUMN entries are allowed to cause wrap round. If a LINE entry has a WRAP clause, the COLUMN entries forming the description of the line must end in one or more relative COLUMN entries. Entries with absolute COLUMN numbers still cannot exceed the maximum line width.

6. A superfluous WRAP clause is not permitted. So the COLUMN entries (in particular the trailing relative COLUMN entries) must be such that the LINE LIMIT could be exceeded. (If this is not foreseeable at precompilation time, it will be assumed that this could happen at run time.)

7. If the WRAP clause is coded at a higher level with more than one LINE entry beneath it, it is sufficient if at least one of the LINE clauses obeys these rules. For example, it is permissible to code the WRAP clause at the 01-level, even if only one of LINE entries in the group can cause wrap round. However, in general any number - even all - the LINE entries in a report may be capable of causing wrap round.

8. The format b NO WRAP clause is allowed only at a level subordinate to a format a WRAP clause. The entry containing NO WRAP must represent more than one physical elementary printable field. No other nesting of the clause is permitted.

3.28.3 WRAP Clause: Operation

1. The WRAP clause causes data to wrap round automatically to a new continuation line when the next field or group of fields will not fit on the line. It may be used in any TYPE of report group. The point where wrap round begins is always after a complete elementary field. Hence a horizontal "fit test" is performed before each field is output, unless it has an absolute COLUMN or is known to fit. (Compare the "page-fit test" performed in the vertical direction.) If the field's right-hand column would extend beyond the column position given in the AFTER phrase, the line is output without the field and then space-filled. The field in question is now placed in the new line, starting in the column position given in the TO phrase. The initial spacing implied by the "PLUS integer" of the COLUMN clause is ignored. The vertical line spacing is given by the STEP phrase.

If the identifier form of the AFTER or TO phrase is used, the identifier's current value is used.

The following complete example shows how a variable number of error messages may be output:
3.28 – WRAP

<table>
<thead>
<tr>
<th>VALIDATION REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>JONES</td>
</tr>
<tr>
<td>TIMSON</td>
</tr>
<tr>
<td>RO%ERS</td>
</tr>
<tr>
<td>SMITH</td>
</tr>
</tbody>
</table>

**The phrase AFTER COL 64 gives the last column that can be occupied by any field in the line. The phrase TO COL 36 gives the starting column for the wrap round.** 

STEP 1 is assumed in default, so continuation is onto the next line. Note that the initial spacing implied by the "+3" of the first COLUMN clause on the new line is ignored.

2. As well the conditional case just described, where the continuation line appears in exceptional circumstances, it is possible (though less useful) for the wrap round to occur every time, as in this example:

<table>
<thead>
<tr>
<th>03 LINE + 2 WRAP AFTER COL 30 TO COL 11.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 COL 1 PIC X(30) SOURCE W-SURNAME.</td>
</tr>
<tr>
<td>05 COL + 3 PIC X(20) SOURCE W-GIVEN-NAME (1).</td>
</tr>
<tr>
<td>05 COL + 3 PIC X(20) SOURCE W-GIVEN-NAME (2).</td>
</tr>
</tbody>
</table>

which is exactly equivalent to:

<table>
<thead>
<tr>
<th>03 LINE + 2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 COL 1 PIC X(30) SOURCE W-SURNAME.</td>
</tr>
<tr>
<td>03 LINE + 1.</td>
</tr>
<tr>
<td>05 COL 1 PIC X(30) SOURCE W-SURNAME.</td>
</tr>
<tr>
<td>05 COL 11 PIC X(20) SOURCE W-GIVEN-NAME (1).</td>
</tr>
<tr>
<td>05 COL + 3 PIC X(20) SOURCE W-GIVEN-NAME (2).</td>
</tr>
</tbody>
</table>

3. Horizontal Fit Test

The "fit test" takes into account a field's variable length. By changing the PICTURES in the previous example to make them variable length, we have the following, more common, situation:
01 FULL-NAME TYPE DE.
  03 LINE + 2 WRAP AFTER COL 30 TO COL 11.
  05 COL 1 PIC X<X(29)> SOURCE W-SURNAME.
  05 COL + 3 PIC X<X(19)> SOURCE W-GIVEN-NAME (1).
  05 COL + 3 PIC X<X(19)> SOURCE W-GIVEN-NAME (2).

The following sample output shows one of each possible outcome:

```
JONES  THOMAS  EDWARD
PIETRASZEWSKI  HILDEGAARD
  GRAZyna
HAUBENSTOCK-MASTELLINE
  FERRUCCIO  EMILIO
PILGERSTORFER-TARTSHOFF
  PHILOMELA
  CLYTAEMNESTRA
```

The longer names in this layout are a good example of the rare conditions that must be allowed for, but which, without this special feature, often take up a disproportionate amount of programming effort.

4. If the containing group is a body group beginning with a relative LINE clause, the continuation lines are taken into account during the group’s page-fit test. For example, if only two lines are available for the printing of the last group in the preceding example, a new page would be started:

```
PILGERSTORFER-TARTSHOFF
  PHILOMELA
  CLYTAEMNESTRA
```

whereas in all the other instances the group would fit on the current page.

5. When an OCCURS clause or a Multiple LINES Clause is coded at the same level as WRAP, the WRAP clause applies separately to each occurrence.

6. NO WRAP indicates that the entries subject to NO WRAP must appear together in the same line. The horizontal “fit test” is therefore performed before the entire set of fields is output to ensure that they are not to split over a line boundary. (Compare NO MULTIPLE PAGE which does the same for the page boundary.) When an OCCURS clause is coded at the same level as NO WRAP, the NO WRAP clause applies separately to each occurrence. The following example lists names across several lines, keeping initials and surnames together:
3.28.4 Compatibility

The WRAP clause is unique to new Report Writer.
Procedural Statements

COBOL Report Writer will not act until your program executes a procedural statement. Three main commands or "verbs", INITIATE, GENERATE, and TERMINATE, are sufficient to produce most of the output from your Report Descriptions. INITIATE and TERMINATE are performed at the beginning and the end, respectively, of the processing for your report; while GENERATE is executed repeatedly, producing one DETAIL (except in summary reporting), preceded by any of the other TYPES of group that may be needed as page breaks and/or control breaks are encountered.

As with the two preceding parts, users migrating from OS/VS or DOS/VS COBOL may refer to the Compatibility paragraph following each section.
4.1 Report Writer Verbs: Overview

The three main report writer verbs INITIATE, GENERATE, and TERMINATE may be used in the same way as any other COBOL verb and may be used anywhere in the program except in a USE BEFORE REPORTING Declarative SECTION.

Of the remaining procedural statements, the USE BEFORE REPORTING directive (see 4.7) enables you to write a section of code in the DECLARATIVES portion that is to be performed automatically just before the specified report group is output, and the Report Writer SET statements (see 4.4) make it possible to place report groups irregularly on the page.

4.1.1 Sequence of Operations

For a single report using a simple file as input, the normal sequence of operations is as follows:

1. (once at start) OPEN INPUT input file
   OPEN OUTPUT or EXTEND report file

2. (once at start) INITIATE report

3. (for each record in input file) GENERATE detail groups or report

4. (once at end) TERMINATE report

5. (once at end) CLOSE input file, report file

with the following basic plan for the PROCEDURE DIVISION:

```
OPEN INPUT input-file OUTPUT report-file
INITIATE report-name
read first input record
PERFORM UNTIL END-OF-FILE = 1
   GENERATE detail-group
* or GENERATE report-name if doing summary reporting
   read next input record
END-PERFORM
TERMINATE report-name
CLOSE input-file report-file
```
### 4.1.2 Keyword Table

The following table lists the **PROCEDURE DIVISION** elements associated with COBOL Report Writer with a summary of their purposes. The third and fourth columns tell you whether or not the item is part of the current standard (ANS 85) COBOL and, if so, whether COBOL Report Writer extends the facilities.

**Report Writer Verbs: Keyword Table**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Purpose</th>
<th>OS/VS DOS/VS COBOL?</th>
<th>Extensions to OS/VS and DOS/VS COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INITIATE</strong></td>
<td>Prepares report for processing</td>
<td>yes</td>
<td>- INITIATE...UPON file-name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- report may be GLOBAL</td>
</tr>
<tr>
<td><strong>GENERATE</strong></td>
<td>Handles main report processing</td>
<td>yes</td>
<td>- improved order of totalling,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>page-fit and DECLARATIVES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- if NOOSVS option used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- report may be GLOBAL</td>
</tr>
<tr>
<td><strong>TERMINATE</strong></td>
<td>Concludes all processing for report</td>
<td>yes</td>
<td>- report may be GLOBAL</td>
</tr>
<tr>
<td><strong>USE BEFORE REPORTING</strong></td>
<td>Invokes SECTION in DECLARATIVES when named report group is printed</td>
<td>yes</td>
<td>- may specify DETAIL group</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- may be GLOBAL</td>
</tr>
<tr>
<td><strong>SUPPRESS PRINTING</strong></td>
<td>Prevents data being printed for a report group</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>MOVE 1 TO PRINT-SWITCH</strong></td>
<td>Alternative to SUPPRESS PRINTING</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td><strong>SET PAGE/LINE/COLUMN</strong></td>
<td>Controls PAGE BUFFER operations</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
4.2 **GENERATE statement**

The **GENERATE** statement is COBOL Report Writer's main verb for the production of output. It passes control to report writer to allow it to perform all the necessary mechanical tasks, including any control-break and page-break processing needed before producing all the lines and fields described in your DETAIL group, if specified.

### 4.2.1 Format

```plaintext
  ⬠GENERATE detail-group-name ⬠report-name
```

### 4.2.2 GENERATE Statement Coding Rules

1. If **GENERATE detail-group-name** is coded, it must be the name of a DETAIL group coded in the current program, or in a GLOBAL report defined in a containing program. (The group-name appears immediately after the 01 level-number.) You may qualify the detail-group-name with the report-name, as in: **GENERATE MAIN-DETAIL IN SUMMARY-REPORT**. This is necessary if your detail-group-name is not unique in the REPORT SECTION.

2. The form **GENERATE report-name** has a special significance and is known as summary reporting. It causes any DETAIL group to be suppressed, so do not use this form unless you require only CONTROL HEADING or CONTROL FOOTING groups in the body of the report at the point that you execute the GENERATE. If you use this form, you must have at least one CONTROL HEADING or CONTROL FOOTING group in the report.

3. **GENERATE** must not appear in a **USE BEFORE REPORTING directive**

### 4.2.3 GENERATE Statement: Operation

1. The **GENERATE** statement causes report writer to perform three main actions in an average report:

   a. It tests for control breaks, producing CONTROL FOOTING and HEADING groups where necessary.

   b. It performs a page-fit test, outputting PAGE FOOTING and PAGE HEADING groups where necessary; (these may also be produced as a result of a CONTROL HEADING or CONTROL FOOTING).

   c. It generates each line in the DETAIL group, unless you are doing summary reporting (GENERATE report-name).

2. Once a report has been **INITIATED**, your program may execute any number of **GENERATE** statements for each DETAIL group in the report. If your **Report Description** contains several DETAIL groups, you may code a sequence of different **GENERATE** statements in any part of the program and, in this way, build up any required report layout. You may also write a **GENERATE** for the same DETAIL group in more than one place in the program.
3. **Summary reporting**, where you code the report-name instead of a DETAIL group-name after the GENERATE, has the following effects:

   a. **No DETAIL group** is output.

   b. Any **rolling forward** of SUM operands takes place as usual, except for any rolling forward from a DETAIL group.

   c. Any **cross-footing** of SUM operands takes place as usual, except for cross-footing within a DETAIL group.

   d. Any **subtotaling** of (non-REPORT SECTION) SUM operands is executed as follows:

      i. If SOURCE SUM correlation is in effect, **all** the SUM operands that correspond to a SOURCE operand in a DETAIL group are added into their totals, as though you had GENERATED each DETAIL group in turn. Any non-REPORT SECTION SUM operands that do not correspond to a SOURCE operand are added into the totals once.

      ii. If SOURCE SUM correlation is **not** in effect, the SUM operands are added into the totals once.

   e. Testing for **control breaks** takes place as usual. If a control break is detected, any CONTROL FOOTING and/or CONTROL HEADING groups are output as usual, together with any PAGE FOOTING and/or PAGE HEADING groups that may be required as the result of a page advance.

The **GENERATE report-name** statement can therefore only produce output (a) on the **first** GENERATE after an INITIATE, and (b) **after a control break**.

Between an INITIATE and TERMINATE your program may execute both the GENERATE report-name and the GENERATE group-name forms of the statement.

The following example illustrates the different effects of the GENERATE report-name and GENERATE group-name clauses:
### 4.2 – GENERATE

#### GENERATE Processing Cycle

The following is a more thorough description of each stage in the execution of a GENERATE statement:

1. If the **identifier** form of the LAST DETAIL sub-clause is used, its value is checked and, if valid, is stored in the Report Control Area.

2. If your report is associated with a DUPLICATED file, the value of **REPORT-NUMBER** is examined to see whether it is the same as it was for the previous GENERATE for this report, thus checking that the correct duplicate of the report is in the main Report Control Area. If not, this is swapped in.

3. If the report has not yet been **INITIATED**, run time error diagnostic 14 is logged.

4. If this is the **first GENERATE** since the INITIATE:
   a. If there is a **REPORT-HEADING** group, this is produced.
   b. If there are any **CONTROL-HEADING** groups, each of them is produced, from highest down to lowest, and the initial value of each control is saved.
5. If this is not the first GENERATE since the INITIATE, each control identifier is compared with the corresponding saved previous value, beginning with the highest level. If no control has changed, no special action takes place. If a difference in value (a control break) is detected, comparison ceases and the following control break action takes place:

   a. The value of each control-id is temporarily altered to the value it had immediately before the control break;
   b. CONTROL FOOTING groups are produced, from the lowest up to the one at the level of the control break, if any;
   c. The value of each control-id is restored to its value after the control break;
   d. CONTROL HEADING groups are produced from the one at the level of the control break, if any, down to the lowest.

Since CONTROL HEADING and CONTROL FOOTING groups are independent report groups in their own right, several of the same operations described below will be applied to them as for a DETAIL group, namely: the output of any pending REPEATED groups, page-fit test, storing of the latest value of the CODE (not done for CONTROL FOOTINGs), all types of totalling, performing of USE BEFORE REPORTING section, production of print lines and clearing of totals, plus the setting on of any PRESENT AFTER (or GROUP INDICATE) flags, when appropriate.

6. If there are any REPEATED groups in this report other than the current DETAIL, a check is made whether any have been buffered. If so, they are first output and the buffer is cleared.

7. If there are any cross-foot totals for this group, they are computed in the order implied by any inter-dependencies among them.

8. If OSVS is in effect any additional summing is now performed for the group with the following possible actions:

   a. If there is any general "subtotalling" for the report (SUM clauses with non-REPORT SECTION operands, without UPON, and with no SOURCE SUM correlation), each SUM's operands are added into the total fields. If you are generating a DETAIL group which is absent because of a PRESENT/ABSENT WHEN/AFTER clause in the 01-level entry, this general subtotalling is also skipped.

   b. If there is any special subtotalling triggered by this DETAIL group due either to an UPON phrase referring to this group or to SOURCE SUM correlation that implies this group, the SUM operands are added into the total fields.

   c. If there is a SUM clause in another group referring to an entry in this group, then rolling forward of values into its total field takes place.
9. If there is a **USE BEFORE REPORTING** section for this group in the DECLARATIVES, it is performed. If **PRINT-SWITCH** is non-zero as a result (meaning that printing is to be **SUPPRESSed**), then
   a. If **OSVS** is in effect no further action takes place for this group;
   b. If **NOOSVS** is in effect in effect then if no further totalling to be performed for this group, no further action takes place; otherwise the only further steps to be performed are 10 (**PRESENT** at 01-level, if applicable, and 8 (totalling)).

10. If there is a **PRESENT/ABSENT WHEN** or **PRESENT/ABSENT AFTER** clause at the 01-level of this group, a test is made of the condition and, if the group is **absent** then
   a. If **OSVS** is in effect no further action takes place for this group;
   b. If **NOOSVS** is in effect then, if there is no general subtotalling to be performed, no further action takes place; otherwise the only further step to be performed is 4.2.4 8a above (general subtotalling).

11. If there is an identifier form of a CODE clause in the RD, the contents of the identifier are moved to the **CODE-VALUE** location in the Report Control Area.

12. If this group has a **REPEATED** clause, the REPEATED buffer is prepared to receive the next instance of the group or, if this group is the last of the set, to produce the buffered groups alongside it.

13. If any lines are being produced and the report has a **PAGE** clause, a page-fit test is performed to test **LINE-COUNTER**, to establish whether or not a page advance is required before the group may be output. If the group has a **MULTIPLE PAGE** clause, this test is performed for the first and each subsequent line (or group of lines with NO MULTIPLE PAGE).

14. If a page advance is required, the following action takes place:
   a. The **PAGE FOOTING** group is produced, if one exists;
   b. **PAGE-COUNTER** is incremented by 1;
   c. A form feed is output or, if an Independent Report File Handler is in use, a value of zero is placed in the current position location to cause this;
   d. The **PAGE HEADING** group is produced, if one exists;
   e. If there are any **CONTROL HEADING** groups specifying **OR PAGE**, they are produced, from highest down to lowest.

15. If **NOOSVS** is in effect and there is any further summing to be performed for this group, step 8 (subtotalling and rolling forward) is now executed.

16. Each report line field is stored in its report line, invoking **FUNCTION** routines where necessary and checking for column overlap, line overflow and any other possible error conditions, and then output. If an Independent Report File Handler is in use, it is invoked; otherwise, report writer issues a **WRITE** for each report line. In either case, **LINE-COUNTER** is first set to the target line position just before each line is produced.

4.2 – GENERATE 217
17. If the group has a **NEXT GROUP** clause, LINE-COUNTER may be adjusted in accordance with the rules for that clause. (See **NEXT GROUP clause**.) In the case of NEXT GROUP absolute, this may be deferred by setting the **Saved Next Group Integer**.

18. All total fields defined in this group are reset to zero, unless they are not **PRESENT** during this **GENERATE** or have a **RESET** phrase that defers resetting to a higher control break.

19. If there are any **PRESENT AFTER** (or **GROUP INDICATE**) clauses in the group, their indicators are set off.

**4.2.5 Compatibility**

1. The coding rules for the **GENERATE** are identical for **OS/VS COBOL**, **DOS/VS COBOL** and new Report Writer. The **GENERATE** statement may perform many more steps, but only because of the additional functions provided by new Report Writer.

2. Access to a **GLOBAL** report is not available with **OS/VS** or **DOS/VS COBOL**.
4.3 **INITIATE statement**

The **INITIATE** statement must be the first report writer statement to be executed for a report.

### 4.3.1 Format

```plaintext
  ┌── Format ─────────────────────────────────────────────┐
  │           ┌─────────┐                           │
  │   ┌─────▼   │           │                        │
  │   │       │   │           │                        │
  │   │     └─►◄       └─►◄       └─►◄                │
  │   │                      INITIATE—report-name  UPON file-name  │
  │   │                      ──┬────────────────┬───►◄        │
  │   │                      │               │               │
  │   │                      │   │       │   │       │   │       │
  │   │                      │   │ UPON   │   │ UPON   │   │ UPON   │
  │   │                      │   │ file-name│   │ file-name│   │ file-name│
  │   └─UPON file-name─┘       └─UPON file-name─┘       └─UPON file-name─┘
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 │
  │                                                                 └────────────────────────────────────────────────────┘
```

### 4.3.2 INITIATE Statement: Coding Rules

1. Each report-name must be the name of a report in the current program, or that of a **GLOBAL** report defined in a containing program. (The report-name appears immediately after the **RD** level-indicator and also in the **REPORT** clause in the FD.)

2. If the **UPON** phrase is present, each report-name must be defined in a **REPORT(S)** clause in the FD of the specified file-name. The **UPON** phrase **must** be used if any of the report-names is defined in more than one FD entry.

3. **INITIATE** must **not** appear in a **USE BEFORE REPORTING** directive.

### 4.3.3 INITIATE Statement: Operation

1. An **INITIATE** **must** be executed for a report before any **GENERATE**, **INITIATE**, or Page Buffer SET verb referring to the same report (or a DETAIL in the report) is executed.

2. An **OPEN** for the corresponding report file must have been executed before the **INITIATE** is executed. The **INITIATE** does **not** **OPEN** the file. You may however execute an **INITIATE** once again for a report that was **TERMINATEd** without closing and re-opening the file. This fact may be used repeatedly to obtain REPORT FOOTING and REPORT HEADING groups in the interior of the report, or to obtain a fresh page with **PAGE-COUNTER** reset to 1.

3. A **CLOSE** **must** **not** be issued for the file to which a report is directed once the report has been **INITIATEd**, unless a **TERMINATE** is first done.

4. If an **UPON** phrase is present, the report will be written only to the file specified.

### 4.3.4 INITIATE Processing Cycle

The following is a more thorough description of each stage in the execution of an **INITIATE** statement:

1. The **error flag** is cleared.

2. If your report is associated with a **DUPLICATED** file and **REPORT-NUMBER** is zero, the remaining actions are performed for every duplicate report.
3. If your report is associated with an Independent Report File Handler, the file handler is invoked with an action code of 6.

4. If the identifier form of a **LINE LIMIT** clause was coded, the identifier is checked and, if valid, stored in the Report Control Area.

5. Other internal locations and special registers, such as the current position, "body group has appeared on page" indicator, REPEATED-COUNTER, and PAGE-COUNTER are cleared.

6. **LINE-COUNTER** is reset to zero.

7. **PAGE-COUNTER** is set to 1.

8. All total fields, sum overflow indicators, size error indicator and Saved Next Group integer, and PRESENT AFTER indicators, wherever appropriate, are cleared to zero.

9. The control break indicator is set to -1 to indicate "initial control break on INITIATE".

10. If a run time subroutine is used for control-break detection, the lengths of each control identifier (other than REPORT/FINAL) are determined and stored in a control area.

### 4.3.5 Compatibility

1. The coding rules for the INITIATE are identical for **OS/VS COBOL** and **DOS/VS COBOL** Report Writer, and new Report Writer. The INITIATE statement may perform additional actions if new Report Writer functions have been used in the Report Group Descriptions.

2. Access to a **GLOBAL** report is not available with OS/VS or DOS/VS COBOL.

3. The **UPON** phrase is not supported by OS/VS or DOS/VS COBOL.
4.4 Report Writer SET statements

These statements enable you to hold the current page, in whole or in part, and/or fill it in an irregular fashion.

4.4.1 Format a

```
►►SET PAGE STATUS TO HOLD
     IN report-name
     OF
   └──►◄
     └──REPORT
```

4.4.1 Format b

```
►►SET LINE TO integer-1
     IN report-name
     OF identifier-1
   └──►◄
     └──FIRST DETAIL DE
     └──UP DOWN BY integer-2
     └──identifier-2
```

4.4.1 Format c

```
►►SET COLUMN TO integer-3
     IN report-name
     OF identifier-3
   └──►◄
     └──LEFT RIGHT
     └──REPORT
```

4.4.2 SET Statements: Coding Rules

1. Format a (SET PAGE) cannot be used unless there is (a) a WITH PAGE BUFFER clause in the SELECT ... ASSIGN clause for the associated file and (b) a PAGE LIMIT clause in the associated RD entry. (The Page Buffer feature uses an Independent Report File Handler to produce the report output and will assume MODE PRINT if there is no MODE specified in your SELECT ... ASSIGN clause. File handlers are described later (see 5.3 Independent Report File Handlers).)

2. The SET LINE and SET COLUMN statements cannot be used unless there is either a WITH PAGE BUFFER clause or a WITH RANDOM PAGE clause in the SELECT ... ASSIGN clause.

3. If your program contains more than one Report Description, you must qualify your SET PAGE STATUS, SET LINE and SET COLUMN statements by IN or OF report-name. Without qualification, the statements are assumed to refer to your one and only Report Description.
4. Format b (\texttt{SET LINE}) is used for altering the value of \texttt{LINE-COUNTER}. \texttt{SET LINE TO ...} sets \texttt{LINE-COUNTER} equal to the value given and forces the next line to appear there. You can use the FIRST DETAIL form with the TO phrase. \texttt{SET LINE DOWN BY ...} adds to \texttt{LINE-COUNTER}, while \texttt{SET LINE UP BY ...} subtracts from it. In each case, the value that results must not be less than the FIRST DETAIL value and must not be greater than the LAST DETAIL value (or their defaults; see 2.9 PAGE LIMIT clause). If you use \texttt{SET LINE} to decrease \texttt{LINE-COUNTER}, your report's PAGE STATUS must be \texttt{HOLD}.

5. Format c (\texttt{SET COLUMN}) is used for altering the value of the horizontal margin. \texttt{SET COLUMN TO ...} sets it equal to the value given. \texttt{SET COLUMN RIGHT BY ...} adds to it, while \texttt{SET COLUMN LEFT BY ...} subtracts from it. In each case, the value that results must not be less than one and must not be greater than the LINE LIMIT, and any group produced must fit within the LINE LIMIT when the new left margin, resulting from \texttt{SET COLUMN}, is taken into account.

6. You cannot use any of these \texttt{SET} statements until a report is in an INITIATED state.

\textbf{4.4.3 \texttt{SET} Statements: Operation}

1. The \texttt{Page Buffer facility} is designed to cope with the type of layout where you may not wish to store the groups starting at the top and working down to the bottom of the page. Look at the following layout, for example:

```
(ORDER       1992)

(A) NAME AND ADDRESS
     JOHNSON T.L.
     216A EAST ST.
     NORTHWOOD
     SUSSEX

(B) 20 APR 1 x GOLF 5 IRON $180
     1 x SIZE 9 GOLF SHOES $30

(B) 18 MAY 2 x FEATHER SHUTTLES $12

Because groups A and B can be of any size, it is practically impossible to define the layout line-by-line. The design has an attractiveness born of a revolt against servile acceptance of the dictum that "printers cannot move backwards". The best way to take advantage of the facility is to \texttt{GENERATE} report groups as their data becomes conveniently available, addressing the page in random-access fashion. The Page Buffer facility enables you write code such as:

\begin{verbatim}
SET PAGE STATUS TO HOLD
GENERATE (A)
SET LINE TO FIRST DETAIL
SET COL RIGHT BY 20
GENERATE (B)
GENERATE (B) ...
\end{verbatim}

In fact, using this facility, you may return to any part of the page. You may also shift a group laterally (left or right) using \texttt{SET COLUMN}.
2. **SET PAGE STATUS (Format a)**

   The **HOLD** option places your report in **HOLD** status. It will stay in **HOLD** status until you issue a **SET PAGE STATUS TO RELEASE** or until the report is **TERMINATED**. When your report is in **HOLD** status, all the lines produced are stored in a page buffer instead of being output, giving you the opportunity to return to a previous higher position at any time.

3. You can issue the **SET PAGE STATUS TO HOLD** at any time - not just at the start of the page. You can then return to any vertical position on or below the position where you issue this command.

4. You may use the SET statements in a **Declarative section**. In this way you may re-position a non-DETAIL report group such as a CONTROL FOOTING, HOLD the page at the start of a PAGE HEADING, and so on.

5. When the report is in **HOLD** status, **LINE-COUNTER** advances as usual. Report writer performs the **page-fit test** on body (DETAIL and CH/CH) groups in the normal way by checking the value of **LINE-COUNTER** against the size of the group about to be printed. If the group cannot be fitted on the page, report writer will execute a page advance despite the **HOLD** status. **NEXT GROUP** and **NEXT PAGE** work as normal. When a page advance takes place, all the lines in the page buffer are first printed. **No data will be lost**. The new page will still have **HOLD** status.

6. **HOLD** status does not change any of the logical processes of report writer. It just makes it legal for you to return to a higher line using the SET LINE statement. **HOLD** status only defers the actual time when output occurs, but the end result is always the same. For efficiency, the best time to **RELEASE** a page is just after the last upward SET LINE on a page.

7. You cancel **HOLD** status by means of the **SET PAGE STATUS TO RELEASE** statement. The page buffer will then gradually be emptied as you write more lines, until such a time as a page advance takes place or the report is **TERMINATED**.

8. **SET LINE (Format b)**

   The **DOWN** and **UP** options increase or decrease **LINE-COUNTER** by the amount stated. The **TO** option is used to place your next group in a fixed vertical position. For example, if your next group begins at absolute line 6 and your report has passed line 6, you may issue **SET LINE TO 6**. (If you do not do this, a page advance will take place.) Similarly, if you **GENERATE** groups with relative lines and wish to return to the FIRST DETAIL position that has a value of 6, then again you would issue **SET LINE TO 6**. The **SET LINE TO FIRST DETAIL** option is available as an alternative way of stating this.

   The effect of **SET LINE** is cancelled by a page advance (except before the first page - **SET LINE** can therefore be done immediately after **INITIATE**).
9. **SET COLUMN** *(Format c)*

This statement changes the value of your report’s left margin. If you have not issued a SET COLUMN statement, the margin will be 1. This is the normal value, indicating that the horizontal position is not to be shifted. The **RIGHT** and **LEFT** options increase or decrease the setting of the left margin, while the **TO** option sets the margin to the value you specify. Your report does not need to be in HOLD status for you to use this statement.

10. If the left margin has been set greater than 1, all the lines produced for the current page will be shifted to the right by the additional factor. For example, if you issue **SET COLUMN TO 5**, then “COLUMN 1” in any print line is actually positioned on column 5.

11. When report writer executes a page advance it resets the left hand margin to 1. Your SET COLUMN statements are therefore effective only within the current page.

12. All formats

Using the **SET LINE** and **SET COLUMN** statements, you can now re-position your group to any position on the page. You can fill the page in any manner. Your groups may overlap, provided that you do not overwrite a character in the page buffer with a different character. Spaces are excluded from this rule. Spaces behave as “cellophane”, not as “white-out”, so you can overwrite with a space without losing what was there before. You can also overwrite a character with the same character. In all other cases, the file handler will signal a run time message.

13. The example on the following page shows how you may set up a page in “snaking columns” and then place a border around the whole page:
MEMBERSHIP LIST
****************************************************************
* T.J. CODER             G. ANALYST             C.S. PAGE-BREAK *
* 26 TONBRIDGE ST.       33 EAST DRIVE          45 BOOKHAM DRIVE *
* MARLBOROUGH            BENBECULA              ORPINGTON *
* WILTS                  TEL: 0955 1234          KENT *
* TEL: 0313 7775          R. WRITER              TEL: 0975 3124 *
* T.W. CODASYL           300 HOPE TERRACE       *
* 34AB SOUTH SIDE         HITCHAM                DR. S. COBOL *
* BRACKNELL              TEL: 0211-686 5432      99 STAINES ST. *
* BERKS                  ABINGER                *
* TEL: 0761 2376          SURREY *
* ▼                       LH5 3ED *
* ▼ etc.                 TEL: 0655 90101 *
* ▼ etc.                 *
* etc.                   *
****************************************************************

The following coding is suitable for this problem:

```
RD MEMBERSHIP-LIST
FIRST DETAIL 2
PAGE LIMIT 60
LINE LIMIT 132.
*
01 TYPE PH LINE 1 COL 40 VALUE "MEMBERSHIP LIST".
*
01 NAME-ADDR-BLOCK TYPE DE.
   03 LINE + 2 COL 3 PIC X(32) SOURCE NAME.
   03 LINE + 1 COL 3 PIC X(32) SOURCE ADDR-LINE (ADDR-LINE-NO)
      OCCURS 1 TO 7 DEPENDING ON ADDR-LINE-CNT VARYING ADDR-LINE-NO.
   03 LINE + 1 COL 3 PIC X(32) SOURCE TEL-NO.
*
01 BORDER TYPE DE.
   03 LINE 2 PIC X(132) VALUE ALL "*".
   03 LINE +1 OCCURS 57.
      05 COL 1 VALUE "*".
      05 COL 132 VALUE "*".
   03 LINE +1 PIC X(132) VALUE ALL "*".
```
* PROCEDURE DIVISION.
...
open all files
INITIATE MEMBERSHIP-LIST
SET PAGE STATUS TO HOLD
fetch first record
MOVE 1 TO MAJOR-COL-COUNT
PERFORM GENERATE-GROUP UNTIL EOF = 1
TERMINATE MEMBERSHIP-LIST
close all files
STOP RUN.
*
GENERATE-GROUP.
*TEST WHETHER THE GROUP WILL FIT ON THE PAGE
  IF   LINE-COUNTER + ADDR-LINE-CNT > 57
    PERFORM CHANGE-MAJOR-COLUMN.
    GENERATE NAME-ADDR-BLOCK.
    (read next record or set EOF = 1 if end of file).
  CHANGE-MAJOR-COLUMN.
*IF WE ARE ALREADY IN THE 3RD COLUMN, PRINT THE "BORDER"
*AND ALLOW PAGE TO ADVANCE, RETURNING US TO FIRST MAJOR COLUMN
*OTHERWISE MOVE RIGHT TO TOP OF NEXT MAJOR COLUMN
  SET LINE TO FIRST DETAIL
  IF   MAJOR-COL-COUNT = 3   *> Finish the page
    SET COLUMN TO 1
    SET PAGE STATUS TO RELEASE *> For efficiency
    GENERATE BORDER
    SET PAGE STATUS TO HOLD
    MOVE 1 TO MAJOR-COL-COUNT
  ELSE SET COLUMN RIGHT BY 40
    ADD 1 TO MAJOR-COL-COUNT.

4.4.4 Compatibility

These forms of the SET statement are unique to new Report Writer.
4.5 SUPPRESS PRINTING statement

The SUPPRESS PRINTING statement enables you to prevent a particular report group from being output on a particular occasion.

4.5.1 Format

```
SUPPRESS PRINTING

MOVE integer TO PRINT-SWITCH
```

4.5.2 SUPPRESS PRINTING Statement: Coding Rules

1. The SUPPRESS statement may be coded only in a USE BEFORE REPORTING directive Declarative SECTION (see 4.7).

2. The form MOVE 1 TO PRINT-SWITCH is an alternative IBM extension that means the same as SUPPRESS PRINTING. You may also write MOVE 0 TO PRINT-SWITCH to undo the effect of a MOVE 1 TO PRINT-SWITCH or SUPPRESS PRINTING, and generally treat PRINT-SWITCH as a numeric location, implicitly defined in your program.

4.5.3 SUPPRESS PRINTING Statement: Operation

1. The statement SUPPRESS PRINTING or MOVE 1 TO PRINT-SWITCH prevents the group specified in the USE BEFORE REPORTING from producing any output on this occasion. In other words, no data is set up in any of the lines of the group and none of the lines is produced. LINE-COUNTER is also left unaltered, so suppressing a body group will prevent a page advance. This statement suppresses only the storing of report data and the output of the report lines. It does not prevent other processing, such as the accumulation and clearing of totals and the setting and testing of CONTROL fields. In this respect, it is different from a PRESENT WHEN clause at the 01-level which does prevent all other processing.

2. For example, you may use SUPPRESS PRINTING to "restart" your report after a breakdown. Simply write a USE BEFORE REPORTING section for every group and SUPPRESS each group until your program clears a flag. Your report will now be in the same internal state as when output really took place.

3. Each execution of a SUPPRESS PRINTING or MOVE 1 TO PRINT-SWITCH will prevent output only on that single occasion. Report writer will reset PRINT-SWITCH to zero after each excursion into your USE BEFORE REPORTING section.

4. In USE BEFORE REPORTING there are further examples of SUPPRESS PRINTING.

4.5.4 Compatibility

Apart from the ANS-85 GLOBAL phrase, OS/VS and DOS/VS COBOL, and new Report Writer agree in their formats for this statement, but new Report Writer allows a USE BEFORE REPORTING section to be coded for a DETAIL group.
4.6 **TERMINATE statement**

The TERMINATE must be the last report writer statement to be executed for each report.

4.6.1 **Format**

![Diagram of TERMINATE statement format]

4.6.2 **TERMINATE Statement: Coding Rules**

1. Each `report-name` must be the name of a report in the current program, or that of a `GLOBAL` report defined in a containing program. (The `report-name` appears immediately after the RD level-indicator and also in the REPORT clause in the corresponding FD.)

2. `TERMINATE` must **not** appear in a `USE BEFORE REPORTING` directive.

4.6.3 **TERMINATE Statement: Operation**

1. A `TERMINATE` must be executed for every report that has been INITIATEd before the final close-down of the program.

2. The TERMINATE statement clears any pending `REPEATED` groups or Page Buffer contents. It also outputs any final `CONTROL FOOTING`, `PAGE FOOTING` and `REPORT FOOTING` groups that may be required at the end of the report. It then returns the report to an "uninitiated" state. `PAGE-COUNTER` and `LINE-COUNTER` will contain the final values they attained at the end of the report, but total fields will be zero (except under erroneous circumstances - see the end of 3.23.4 3 c above).

3. A separate, subsequent `CLOSE` should be executed for the associated report file. TERMINATE does **not** close the file.

4. If a TERMINATE is executed without any `GENERATE` statements being executed for the report since the INITIATE was executed, no output at all is produced. If you wish to ensure that at least the REPORT HEADING and REPORT FOOTING groups appear, you should in this case `GENERATE` a blank DETAIL group before the TERMINATE.

5. A report may be TERMINATEd and then INITIATEd again any number of times without closing the report file. The new INITIATE causes `PAGE-COUNTER` to return to 1 and, if the report has a PAGE LIMIT clause, will re-commence the report on a fresh page.
### 4.6.4 TERMINATE Processing Cycle

1. The following is a more thorough description of each stage in the execution of a TERMINATE statement:

2. If your report is **DUPLICATED** and **REPORT-NUMBER** is zero, the actions that follow are performed for each duplicate report.

3. If at least one **GENERATE** has been performed (indicated by the control break indicator being non-negative), the value of each **control-id** is temporarily altered to the value it had when the last **GENERATE** was executed, whilst each **CONTROL FOOTING** group is produced, from lowest to highest.

4. If any **REPEATED** groups are present, any buffered groups are output and the **REPEATED** buffer is flushed. Also, if the Page Buffer contains any data, this is output.

5. If a **PAGE FOOTING** group is present, it is produced.

6. If a **REPORT FOOTING** group is present, it is produced.

7. If your report is associated with an **Independent Report File Handler**, the file handler is invoked with an action code of 8. (If your report is **DUPLICATED**, this will only take place for reports that have been **INITIATED**.)

8. The current vertical position location is set to -1 to indicate "report not initiated".

9. If any **total fields** are still non-zero, indicating that they have not all been output, an error diagnostic 15 is signalled.

10. If normal batch printing is in effect, a check is made of the error diagnostic flag and an appropriate run time error message is logged if necessary.

### 4.6.5 Compatibility

1. The coding rules for TERMINATE are identical for **OS/VS COBOL** and **DOS/VS COBOL** Report Writer and new Report Writer. The TERMINATE statement may perform more actions because of the additional functions provided by new Report Writer.

2. Access to a **GLOBAL** report is not available with **OS/VS** or **DOS/VS COBOL**.
4.7 USE BEFORE REPORTING directive

The USE BEFORE REPORTING directive causes a SECTION in the DECLARATIVES of your PROCEDURE DIVISION to be performed automatically just before a selected specified report group is produced.

4.7.1 Format

```
PROCEDURE DIVISION.
  USING data-name.

DECLARATIVES.
  declarative-section END DECLARATIVES.
```

where declarative-section is defined as:

```
section-name SECTION segment-number.
  USE GLOBAL BEFORE REPORTING report-group-name.

paragraph-name.
  sentence
```

4.7.2 USE BEFORE REPORTING Directive: Coding Rules

1. The format above shows the PROCEDURE DIVISION and DECLARATIVES headers for the sake of completeness. They are used not just by report writer, and your program might already have Declarative sections for some other purpose. It is the distinctive format of the USE BEFORE REPORTING directive that tells report writer that the section is part of its responsibility. If you also have other Declarative sections, they may be intermixed with your USE BEFORE REPORTING sections in any order.

2. In order to code a USE BEFORE REPORTING section, you must ensure that the corresponding group has an 01-level data-name, so that you can refer to it as the report-group-name.
3. Your USE BEFORE REPORTING section may also PERFORM other sections. These are normally additional sections within the DECLARATIVES portion (as required by all ANS Standards). For this purpose, you may code additional sections within DECLARATIVES that have no USE statement. Report writer also allows your USE BEFORE REPORTING section to PERFORM sections in your mainline PROCEDURE DIVISION.

The example that follows shows how you may code one section to be performed when one of two report groups is about to be produced:

```
01 BRANCH-HDDR TYPE CH FOR BRANCH-CODE.
  ...  
01 DEPT-HDDR TYPE CH FOR DEPT-CODE.
  ...  
PROCEDURE DIVISION.
DECLARATIVES.
CHANGE-BRANCH SECTION.
  USE BEFORE REPORTING BRANCH-HDDR.
  CHB-000.
    PERFORM CHANGE-BRANCH-DEPT.
CHANGE-DEPT SECTION.
  USE BEFORE REPORTING DEPT-HDDR.
  CHD-000.
    PERFORM CHANGE-BRANCH-DEPT.
CHANGE-BRANCH-DEPT SECTION.
  CBD-000.
    ... (common code) ... 
END DECLARATIVES.
```

4. Your USE BEFORE REPORTING section must not contain any INITIATE, GENERATE, or TERMINATE statements. Neither may any of the sections it may perform.

5. If you specify GLOBAL, the named report group must exist either in the current program or in a contained program.

4.7.3 USE BEFORE REPORTING Directive: Operation

1. If any of your report groups has a USE BEFORE REPORTING section, report writer will implicitly PERFORM the section during the processing of the group. Assuming that your report has every possible feature, the section will be implicitly performed:

   After the testing of control breaks and the production of any CONTROL FOOTING and CONTROL HEADING groups (if your group is a DETAIL);

   After the computation of your group’s cross-foot totals (if any), so your USE BEFORE REPORTING section can reference them;

   If OSVS is in effect, after the rolling forward into your totals and subtotalling associated with your group;
If NOOSVS is in effect, before the rolling forward into your totals and subtotalling associated with your group, so your USE BEFORE REPORTING section can alter values that are due to be added into other groups' totals;

Before the moving of any CODE identifier, so your USE BEFORE REPORTING section can change the originating identifier;

Before the page-fit test and the production of any PAGE FOOTING and PAGE HEADING groups (if your group is a body group), so you may alter the originating fields due to be moved into them, or suppress them along with the body group itself.

Before any of the SOURCE, SUM, or FUNCTION fields are set up in the lines of your group, so you may change any of the originating fields due to be displayed in your group.

2. You may include the a SUPPRESS PRINTING statement or MOVE 1 TO PRINT-SWITCH in a USE BEFORE REPORTING Declarative, in order to prevent output for the group at that instant (see 4.5).

3. USE BEFORE REPORTING sections were used a great deal in the ANS-68 and ANS-74 COBOL report writer. So you may possess migrated programs that contain cases of their use. With report writer much of their functions are now performed by the PRESENT WHEN and PRESENT AFTER clauses. However, they can still be of considerable use. For example:

   a. You could use your Declarative section to WRITE additional records to another file, using the automatic control break processing to "drive" the rest of your program.

   b. You might use the USE BEFORE REPORTING section for a CONTROL HEADING group to READ an additional record at the start of each new CONTROL value, or fetch it from your database.

   c. You might need to suppress the printing of certain totals without preventing them from being reset to zero. (The PRESENT clause will prevent the resetting of a total field if it was not output.)

   d. You might want to force a CONTROL HEADING group to start on a new page under certain complex circumstances. Your USE BEFORE REPORTING section would then force page advance processing thus:

      \[
      \text{IF complex-condition} \\
      \quad \text{MOVE last-detail-value TO LINE-COUNTER.}
      \]

   e. You might want to search a table for a corresponding text field at the start of each CONTROL HEADING group, and move it to a WORKING-STORAGE field that is the operand of a SOURCE in a PAGE HEADING or the CONTROL HEADING itself. (If your group is a DETAIL, it will be clearer to do this in the main-line program.)
f. There may be an item associated with a control which is not itself a control (such as a STATE-NAME logically associated with a STATE-NO control) which you will want to output during CONTROL FOOTING time. Since the item is not a control you will not automatically obtain the before-the-break value. You could code a Declarative section for the corresponding CONTROL HEADING group. This would save the current value of the field in a WORKING-STORAGE location which is then used instead of the input item in all SOURCE clauses in PAGE HEADING, PAGE FOOTING and CONTROL FOOTING groups.

If you have no CONTROL HEADING group, you may code one as a "dummy" (see 3.2 Coding Report Group Descriptions) with no LINE clauses, in order to take advantage of report writer's automatic control break checking, as suggested in the following example:

```
RD ... CONTROL IS IN-STATE-NO.
01 NEW-STATE CH FOR IN-STATE-NO.
01 CF FOR IN-STATE-NO.
  05 COL ... SOURCE IS WS-STATE-NAME.
01 PH.
  05 COL ... SOURCE IS WS-STATE-NAME.
DECLARATIVES.
SAVE-CURRENT-STATE SECTION.
  USE BEFORE REPORTING NEW-STATE.
SAVE-000.
  MOVE IN-STATE-NAME TO WS-STATE-NAME.
END DECLARATIVES.
```

g. You may find it desirable to suppress printing of a minor CONTROL FOOTING if only one DETAIL is printed above it, since a “total” of a single value will seem out of place. Here is one way to do it:

```
01 DETAIL-ENTRY DETAIL.
  03 LINE + 1.
  05 ... .
  01 MINOR-CF CONTROL FOOTING FOR STATE-NO.
    03 LINE + 2.
      05 ... .
      05 R-DETAIL-COUNT PIC 999 COUNT OF DETAIL-ENTRY.
    ... .
DECLARATIVES.
ZAP-CF-SECTION SECTION.
  USE BEFORE REPORTING MINOR-CF.
ZAP-CF-000.
  IF R-DETAIL-COUNT = 1 SUPPRESS PRINTING.
END DECLARATIVES.
```

Do not instead code an ABSENTWHEN clause at the 01-level of the CONTROL FOOTING group, as this would have the undesirable side-effect of preventing the resetting and rolling forward of any SUM fields that you might have defined in the CONTROL FOOTING group.
4. If you specify **GLOBAL** your Declarative section will apply both to the current program, if it contains a report group of that name, and also to any contained program that has a report group of that name but no USE BEFORE REPORTING section of its own for that name. If a contained program also has a USE GLOBAL BEFORE REPORTING section for the same report-group-name, this overrides the effect of original section until the end of the contained program.

### 4.7.4 Compatibility

Apart from the **ANS-85 GLOBAL** phrase, OS/VS and DOS/VS COBOL, and new Report Writer agree in their formats for this statement, but new Report Writer allows a USE BEFORE REPORTING section to be coded for a DETAIL group.
This part treats some more advanced or specialized topics. The production of multiple reports, and the development of user-written extensions (functions and file handlers) are also described.
5.1 Multiple Reports

COBOL Report Writer provides a means for producing either one physical file from several reports, or several physical files from one report description. The following sections describe each case:

5.1.1 Several Reports to One Physical File

This is indicated when you write the clause

```
REPORTS ARE report-name-1 report-name-2 ...
```

in the FD entry for the report file. The different reports may be written either successively, alternately, or concurrently to the file.

5.1.2 Successive Reports

This describes the case when each successive report is processed through from INITIATE to the final TERMINATE and is then not accessed again:

```
INITIATE report-name-1
GENERATE report groups in report-name-1
...
TERMINATE report-name-1
INITIATE report-name-2
GENERATE report groups in report-name-2
...
TERMINATE report-name-2
```

This method is free of complications and has useful applications. One example is the case where a report has a front section or a trailing section, too complex to describe as a REPORT HEADING or REPORT FOOTING:

```
very complex "report heading"<

body of report

very complex "report footing"<
```

The "Report Heading" might have several parts, and the "Report Footing" might have many complex parts, totals, etc. You might instinctively try to describe this complex...
layout using a single **RD**, because it has been designed as a single report.
Remembering that a "**report file**" may be composed of several "**logical reports**", you
now re-code the layout using three RDs:

FD print-file ...
REPORTS ARE HEADING-REPORT, MAIN-REPORT, TRAILING-REPORT.
... 
RD HEADING-REPORT ...
RD MAIN-REPORT ...
RD TRAILING-REPORT ...

and code the **INITIATE**, all the **GENERATEs** and the **TERMINATE** for each report in
tandem. Assuming that each RD has a **PAGE** clause, each report will begin on a new
page, as required.

This example assumes that you have **no page numbering** in the "Report Heading" or
"Report Footing", and that you start with a page number of **one** in the main report. If
you do require page numbering throughout, remember that each report has its own
separate **PAGE-COUNTER**. Assuming that the page numbers are to run in sequence
throughout the report, you must carry forward the PAGE-COUNTER after each
**TERMINATE**, as follows:

PROCEDURE DIVISION.
...
INITIATE HEADING-REPORT
GENERATE heading-groups ...
TERMINATE HEADING-REPORT *

INITIATE MAIN-REPORT
ADD 1 PAGE-COUNTER IN HEADING-REPORT GIVING PAGE-COUNTER IN MAIN-REPORT
(process main report)
TERMINATE MAIN-REPORT *

INITIATE TRAILING-REPORT
ADD 1 PAGE-COUNTER IN MAIN-REPORT GIVING PAGE-COUNTER IN TRAILING-REPORT
GENERATE trailing-groups ...
TERMINATE TRAILING-REPORT

5.1.3 **Alternating-Page-Format Reports**
You may encounter the case where a report consists of two or more **alternating page
formats**, such as **details of area A...summary page for area A details of area
B...summary page for area B...**, where each page format begins on a fresh page and
has different PAGE HEADING, PAGE FOOTING and body groups from the others:
When approaching this problem, keep in mind the points made under **Successive Reports** above. This time, we keep both reports initiated. (If not, the page numbering will return to 1 after each `INITIALIZE` and there will be problems if, say, we want to produce grand totals at the end of the reports.) The following is one possible solution.

```
FD print-file ...
REPORTS ARE DETAILED-REPORT, SUMMARY-REPORT.
...
RD DETAILED-REPORT
...
RD SUMMARY-REPORT
...```
PROCEDURE DIVISION.
...
INITIATE DETAILED-REPORT, SUMMARY-REPORT
obtain first input record
PERFORM UNTIL end-of-file
  GENERATE detailed-groups ...
  IF change in AREA code
    MOVE PAGE-COUNTER IN DETAILED-REPORT
    TO PAGE-COUNTER IN SUMMARY-REPORT
    GENERATE summary-groups
    MOVE PAGE-COUNTER IN SUMMARY-REPORT
    TO PAGE-COUNTER IN DETAILED-REPORT
  END-IF
obtain next input-record
END-PERFORM
TERMINATE DETAILED-REPORT, SUMMARY-REPORT

(An alternative solution is to use an expression made up from both PAGE-COUNTERs, for example:

| 05  | COL 100   | "PAGE:".
| 05  | COL + 2   | PAGE-COUNTER IN DETAILED-REPORT +
     |   | PAGE-COUNTER IN SUMMARY-REPORT - 1.)

Because you are not doing a TERMINATE and INITIATE on each occasion that you "switch reports", they will not necessarily always resume on a fresh page. Consider each report separately and decide how you would make it resume on a fresh page if the other report were not there. For example, you might begin after each "switch" with a GENERATE of a DETAIL group with a low absolute LINE number; or you might use the NEXT GROUP NEXT PAGE clause in a dummy CONTROL HEADING; or you might simply code MOVE footing-integer TO LINE-COUNTER IN next-report-name before resuming.

5.1.4 Concurrent Reports

There are two quite unconnected cases to discuss here:

a. One report built up from several alternating RD’s.

Here, you GENERATE output via two or more RDs associated with the same report file without necessarily waiting for a page advance before switching from one report to another. It can be difficult to use this method, because each report keeps its own LINE-COUNTER and control break information. Each report will therefore act as though it alone had control of the page format. Although no serious breakdown is likely at run time, you will have to take special steps to ensure that pages terminate at the correct point. To do this, you will need to MOVE one LINE-COUNTER to another in much the same way that the PAGE-COUNTERs are dovetailed in the section on "Successive Reports" above. If you specify the same control-ids in each report, remember that each possible control break will occur in each report separately, when it is GENERATED.
b. **Distinct reports spooled to the same file.**

Here, several reports are written to the same file but are distinguished by means of the CODE (see 2.5 CODE clause). This technique was commonly used in the past when print data was spooled onto tape or disk. Several reports could be written to the same spool file by marking the print records of each report with an identifying character, characters, or *identifier* of any length. De-spooling software then passed through the spool file, each time selecting only those print records beginning with a certain identifying character. (It was possible to sort the records, but this is usually much less efficient.) As example, you might have had two reports:

```
FD PRINT-FILE ... 
  REPORTS ARE REPORT-A REPORT-B. 
... 
RD REPORT-A ... 
  CODE IS "A" ... 
*or: CODE IS mnemonic-name 
*and put in SPECIAL-NAMES: "A" IS mnemonic-name 
  01 ... 
... 
RD REPORT-B ... 
  CODE IS "B" ... 
  01 ... 
... 
```

The reports were both written to the same file, but you there was no problem in separating them as the reports eventually appeared as separate outputs, thanks to their distinguishing codes. A utility to separate and print the physical reports is not provided as a part of the report writer software, and users who developed a utility program to handle the output produced by OS/VS and DOS/VS COBOL’s built-in report writer with a CODE clause should use the same program to handle the output from report writer.

### 5.1.5 Several Outputs From the Same RD

There are two cases to be reviewed for this topic:

a. **Identical copies of the same physical report.**

Following the ANS-74/85 standards, report writer does not allow you to assign your report to more than one file. This is more restrictive than the ANS-68 standard used in OS/VS and DOS/VS COBOL which allows up to *two* files. If you require two or more identical copies of the same physical report, you should try to achieve this through Operating System commands, or you should write an **Independent Report File Handler** (see 5.3 Independent Report File Handlers) that executes a `WRITE` for each of several files for each report line.

b. **Different reports with similar formats going to different files.**

In a program with several reports, you may find that two or more report layouts are identical, or nearly so, and you naturally want to avoid duplication of code. The DUPLICATED clause is designed for this purpose. The following diagram shows a case with three files.
Although the layouts are similar, the contents of the reports may be quite different (as distinct from case a where all output is simply printed twice). The DUPLICATED clause (see 2.2 Report Files) sets up the given number of separate report control areas, each with its own PAGE-COUNTER, LINE-COUNTER, totals, control break indicators, Page Buffer, etc. They are distinguished by the value of the special register REPORT-NUMBER which is defined by report writer in every program that has a DUPLICATED clause. You may “switch” output to your selected report by storing a value from 1 to the maximum number of duplicates in REPORT-NUMBER. Output goes to only one of the reports at any given time. Although there are several separate physical files, only one SELECT clause and one FD entry are coded, and one OPEN and CLOSE is executed. Despite your own OPEN statement, a file is actually opened only when the report whose number is held in REPORT-NUMBER is first INITIATED.

If REPORT-NUMBER is zero when the INITIATE is executed, all the reports are INITIATED. The CLOSE operation (not the TERMINATEs) closes all the files that were opened. The value of REPORT-NUMBER is immaterial when you OPEN and CLOSE. You may MOVE ZERO to REPORT-NUMBER and do a single TERMINATE. This will TERMINATE all the reports that were INITIATED. If REPORT-NUMBER is not zero when a TERMINATE is executed, only the corresponding report is terminated.

The report layouts need not be identical in all respects. You may vary the entries “present” from one report to the next by using the PRESENT WHEN clause, and you may vary the contents by the same means or by using SOURCE identifiers subscripted by REPORT-NUMBER. The DETAIL groups may even be different for each of the reports, since you can of course choose at any time which DETAIL groups to GENERATE. Therefore the DUPLICATED clause may still save you coding time even if there are notable differences in the layouts of the report groups.

At run time, report writer will vary the for each of your multiple print files by appending to it the two digits 01 (for the first file) up to nn for the last file, where nn is the integer in your DUPLICATED clause. If you specify a DDname of more than six characters, characters 7 and 8 of your DDname are overwritten. For example:

```
SELECT FILEA ASSIGN TO LST DUPLICATED 3 TIMES
```

expects your DDnames to be LST01, LST02 and LST03, and

```
SELECT...ASSIGN TO LISTINGF DUPLICATED 16 TIMES
```


expects your DDnames to be LISTN01 through LISTN16.

The following sample of coding shows how these techniques are used:

```plaintext
FILE-CONTROL.
  SELECT REPORT-FILE ASSIGN TO PRINTF
  DUPLICATED 3 TIMES.

...  
FD REPORT-FILE
  ...  
  REPORT IS STOCK-REPORT.

...  
REPORT SECTION.
RD STOCK-REPORT ...
  01 STOCK-LINES TYPE DE.
    03 LINE + 2.

*The following is a subheading that is different in each report:  
  05 COL 120 PIC X(20) SOURCE WS-DESCRIPTION (REPORT-NUMBER).
*The following line is not generated in report #1:  
  03 LINE       ABSENT WHEN REPORT-NUMBER = 1.

...  
PROCEDURE DIVISION.  
...
*Report file is opened only once  
*even though it represents 3 potential files  
OPEN OUTPUT REPORT-FILE   *> intercepted by report writer
...
*If report #1 is required during this run:  
  MOVE 1 TO REPORT-NUMBER
  INITIATE STOCK-REPORT
...
*and, in general, if report #n is required:  
  MOVE n TO REPORT-NUMBER
  INITIATE STOCK-REPORT
...
*When output is sent to report #1, for example:  
  MOVE 1 TO REPORT-NUMBER
  GENERATE STOCK-LINES
...
*and, in general, when output is sent to report #n:  
  MOVE n TO REPORT-NUMBER
  GENERATE STOCK-LINES
...
*to terminate all of the 3 reports that may have been initiated:  
  MOVE 0 TO REPORT-NUMBER
  TERMINATE STOCK-REPORT
...
*or, to terminate just report #1:  
  MOVE 1 TO REPORT-NUMBER
  TERMINATE STOCK-REPORT
...  
CLOSE REPORT-FILE.
```
If you refer to **LINE-COUNTER**, **PAGE-COUNTER**, a total field, or any of the other special registers within the Report Control Area, the actual location accessed will be the one belonging to the report for which the most recent INITIATE, GENERATE, or TERMINATE was performed. For example, if you wish to reset PAGE-COUNTER in each of your reports in the multiple set, you should define a dummy DETAIL group (one having no LINES) and code the following:

```cobol
MOVE 1 TO REPORT-NUMBER
GENERATE DUMMY-GROUP    *>needed to make Report #1 current
MOVE 0 TO PAGE-COUNTER
MOVE 2 TO REPORT-NUMBER
GENERATE DUMMY-GROUP
... etc. ...
```

If you use a procedural statement, such as a **MOVE** to store a value directly into a named field in one of your REPORT SECTION lines, the new contents will take effect for all the reports of the multiple set, because the report lines produced by report writer are shared by each of them.

### 5.2 Developing User-Written Functions

#### 5.2.1 The Need for Functions

The **FUNCTION** clause, which is described from the user’s point of view under 3.7 FUNCTION clause, allows you to develop your own function routines as well as to use certain built-in functions, such as **DAY**, **MDATE**, and **TIME**. A user-written function allows you to display report data in a way peculiar to your installation.

User-written functions are especially effective when needed in many separate programs, as they provide a standard way of displaying certain codes or other items of information peculiar to the user site. Indeed a commitment to “think functionally” may greatly improve efficiency throughout a programming department. To convince yourself of this, consider the following scenario, before the introduction of functions:

Many report programs require the programmer to code a table of descriptions or names directly into the Working-Storage of the program. Perhaps at first only a few report programs of this type are written, so that, even when some extra codes are added to the table and all those programs have to be re-compiled, no one objects too strongly.

Suddenly a large number of new programs are written that require the same names. Some programmers code the table from scratch; others prefer to take a copy from a working program. Business conditions then change and many new codes are added to the table. Dozens of programs have to be changed. Any change requires retesting of whole suites of programs. Each program has its own particular way of encoding the table, so an amendment becomes a major undertaking. At the end of it, no one is absolutely sure that all the programs affected have been found and amended.

When the next major change becomes necessary, some one decides that from now on all the codes will be held in a static “table file”. New programs are guaranteed to be independent of the codes and names, but there is a penalty to be paid in each program, because of all the extra steps needed to **OPEN**, **READ** and **CLOSE** the file in the program. Later, there might be a general change from traditional files to a database, so all the programs have to be amended again.
Through the use of functions from the outset, programs can be made independent of all these changes. The program will be shorter and the programmer need not worry about how the report field is created. Control over specification and change becomes more assured.

### 5.2.2 How To Write a Function Routine

As designer of a function routine, you have a free hand to transform the data in any desired way, provided that the print data that results fits within the limits of the print field described by the programmer in the associated PICTURE clause.

Most function routines are COBOL subprograms with a standard LINKAGE SECTION, but they may also be written in any programming language that can be CALLed by COBOL. The use of COBOL normally prevents the number of parameters passed to the FUNCTION from being variable (a minor benefit). Since they are "normal" programs, function routines may use any files or databases that are available to other programs, may CALL other subprograms, and may store intermediate results in their own Working-Storage.

The number of relevant characters returned in the output (report) field may vary, thus enabling the same function routine to handle different sizes and formats, for example "date" formats with different arrangements of day, month, and year. The size and format of the user-coded parameters (if any) are usually prescribed by the function routine and cannot vary, unless the designer decides to specify an additional parameter to indicate which of a choice of formats the input is in. All parameters are passed "by reference" so, although they normally pass data only into the function routine, they may also be used to pass updated data back to the user program.

The function routine is used in the program via the FUNCTION clause, and the programmer need not be aware of the precise mode of operation of the function routine. The description of the built-in functions is also given in 3.7 FUNCTION clause. User-written functions may be similarly specified and added to this publication in an appropriate place.

The program name of the function routine should be Rnxxxxxx, where xxxxxx is the mnemonic name of the function of no more than 6 alphanumeric characters and n is the number of parameters to the function. For example, if the function is called COUNTY and takes one parameter, you must write a function module with the program-id: R1COUNTY otherwise. If the same function has a variable number of parameters, separate function routines must be written (although they may all call a common subordinate routine).

The parameters to the function routine are as follows:

- **Parameter 1:** Function Control Area, consisting of:
  - 2 bytes binary: number of user-coded parameters present in FUNCTION clause, i.e. excluding internal parameters 1 and 2;
  - 2 bytes binary: total size of field, in bytes, or, if field is edited, its equivalent de-edited size;
  - 2 bytes binary: size in bytes of parameter #1 of FUNCTION, if any
  - 2 bytes binary: size in bytes of parameter #2 of FUNCTION, etc.

- **Parameter 2:** report field; if edited, de-edited intermediate field;

Parameters 3,4... (optional) parameters as defined in FUNCTION clause.
IDENTIFICATION DIVISION.
PROGRAM-ID. R1DEPOT.

* COBOL Report Writer user function
* Converts internal depot code 'L', 'B', or 'N'
* to full name for printing in report

* Available report field sizes:
  * 4 = short form ('LNDN', 'BHAM', 'NCSL', ETC.)
  * 12 = long form ('LONDON', 'BIRMINGHAM', 'NEWCASTLE' ETC.)

* To use, write: FUNC DEPOT (DEPOT-CODE) instead of SOURCE...
* 
* This function may be converted later to hold depot names in a
* file or database table without impact on the programs.

ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.
01 WS-FIELDS.
* table of short names
  03 SHORT-NAME-TAB VALUE "LNDNBHAMNCSL????".
  05 SHORT-NAME PIC X(4) OCCURS 4.
* table of long names
  03 LONG-NAME-TAB VALUE 'LONDON      BIRMINGHAM  NEWCASTLE     UNKNOWN    '.
  05 LONG-NAME PIC X(12) OCCURS 4.
  03 W-DEPOT-NUMBER PIC S9(4) COMP.
  03 W-PARAM-ERR PIC X.

LINKAGE SECTION.
* Param (1): Function Control Area
  01 L-FUNC-CTL.
  * No. of parameters to function:
    03 L-PARAM-CNT PIC S9(4) COMP.
  * Length of output field in bytes:
    03 L-OP-LEN PIC S9(4) COMP.
  * Length of parameter in bytes:
    03 L-PM1-LEN PIC S9(4) COMP.

* Param (2): output area
  01 L-OP-FLD.
    03 L-OP-CH PIC X OCCURS 1 TO 12 DEPENDING ON L-OP-LEN.

* Param (3): input depot code
  01 L-DEPOT-CODE PIC X.
PROCEDURE DIVISION USING L-FUNC-CTL, L-OP-FLD, L-DEPOT-CODE.
  FUNCTION-ENTRY.
    PERFORM PARAMETER-CHECK
    IF W-PARAM-ERR = "N"
      PERFORM CONVERT-CODE.
  FUNCTION-EXIT.
  EXIT PROGRAM.
*Routine to check parameters:
  PARAMETER-CHECK.
    MOVE "N" TO W-PARAM-ERR
*Check correct parameter specification
  IF L-PARAM-CNT NOT = 1
    OR L-PM1-LEN NOT = 1
    DISPLAY "R1DEPOT ERROR CRW-651: INVALID PARAMETERS"
    MOVE "Y" TO W-PARAM-ERR.
*Check report field is correct size
  IF L-OP-LEN NOT = 4 AND NOT = 12
    DISPLAY "R1DEPOT ERROR CRW-652: INVALID REPORT FIELD SIZE"
    MOVE "Y" TO W-PARAM-ERR.
*Routine to convert depot code:
  CONVERT-CODE.
*Check input depot code, convert to integer 1-3, or 4 if unknown
  IF L-DEPOT-CODE = "L"
    MOVE 1 TO W-DEPOT-NUMBER
  ELSE IF L-DEPOT-CODE = "B"
    MOVE 2 TO W-DEPOT-NUMBER
  ELSE IF L-DEPOT-CODE = "N"
    MOVE 3 TO W-DEPOT-NUMBER
  ELSE MOVE 4 TO W-DEPOT-NUMBER.
  IF L-OP-LEN = 4
    MOVE SHORT-NAME (W-DEPOT-NUMBER) TO L-OP-FLD
  ELSE MOVE LONG-NAME (W-DEPOT-NUMBER) TO L-OP-FLD.
5.3 Independent Report File Handlers

5.3.1 Introduction

An Independent Report File Handler (or file handler) is a separately-compiled program routine that takes over all the functions of the normal COBOL OPEN-WRITE-CLOSE protocol for a particular report. The source program, as written, is identical in all respects to a regular batch print program, except that the SELECT...ASSIGN clause for the file to which the report is directed carries the additional sub-clause MODE IS... (see 2.2 Report Files), and that there may also be a CODE clause (see 2.5) in the RD. (The MODE sub-clause can also be "forced" by default onto any report file does not have a MODE by an external option setting - see Installation and Operation.)

File handlers monitor the results of any i/o operations they perform and pass any error codes back in the FILE STATUS field in the usual way, or, if no FILE STATUS is specified, they display a suitable message and, if the error is irrecoverable, abort the run.

New Report Writer comes with some built-in file handlers and these are listed below. Any number of others may be written by users to similar standards. File handlers can be written in COBOL or another language.

5.3.2 Supplied File Handlers

The following list explains the function of each of the built-in file handlers and gives the MODE mnemonic you require to invoke each.

| MODE PRINT | Purpose: basic printing. |
|            | This file handler performs the same basic print functions as when no file handler is used at all. Unless you already specify a MODE clause, it will be used automatically when any of the following features is used: |
|            | • PAGE BUFFER clause in SELECT |
|            | • DUPLICATED clause in SELECT |
|            | • STYLE clause |
|            | • UPON option of INITIATE |
|            | • CODEs of unequal length for RDs of same file. |
|            | Effect of CODE: Placed at start of print line, as in ANS standards. It is output after any carriage control characters. |

| MODE NOPF | Purpose: basic printing without using page feeds. |
|           | This file handler fills each page entirely using line feeds. |
|           | Effect of CODE: Placed in print line after the carriage control. |

| MODE MODL | Purpose: for use in a modular system. |
|           | This file handler may be used by more than one separately compiled program in a run unit. Each program may thus write to the same report file. See listing below. |
|           | Effect of CODE: Placed in print line after the carriage control. |
MODE CHAN

Purpose: make optimal use of any printer channels.

The channel positions are fed into the file handler at run time, as described in Installation and Operation. The file handler automatically uses channel skips to get as near as possible to each target line. The RD does not change.

Effect of CODE: Placed in print line after the carriage control.

MODE DUPL

not VSE

Purpose: emulate OS/VS and DOS/VS COBOL's ability to write to two files simultaneously.

The file handler performs the OPEN, WRITE, CLOSE as appropriate for each file in turn.

Effect of CODE: Placed in print line after the carriage control.

5.3.3 User-Written File Handlers

An Independent Report File Handler is not usually developed as part of one program, but separately so that it can be used by all the report programs in a project. Only a brief specification is normally required, similar to those in the table of built-in file handlers below. Any application will then be able to use the file handler.

The file handler has a fixed LINKAGE SECTION, and the areas and their various locations are given in COBOL format below. For use in a language other than COBOL, translate the data-names and PICTUREs as appropriate.

The program-name of the file handler is related to the name of the MODE mnemonic. Full details are given in Operation and Installation.

5.3.4 Possible Uses of a File Handler

In this section we list some possible uses to which you might put a file handler.


Provided your program's "report lines" contain no COMPUTATIONAL entries, you might adapt your file handler to produce output which is not intended for printing but instead for passing on to another program. Then, if you decide to ignore all LINE numbers, so that each "line" is another record irrespective of the "line advance", your file handler would simply do a WRITE without ADVANCING.

2. Output to Database

Your program may be required to send its printed output to a database rather than in a sequential file.

3. Private Styles

If the printer has advanced features that require special programming, you can designate certain Styles as PRIVATE. The processing of such Styles is then left to your file handler. Full details are in Installation and Operation.

5.3 – Developing an Independent Report File Handler
4. Output to Multiple Files

Your file handler may write to several files simultaneously, as outlined above (see 5.1.5 Several Outputs From the Same RD) and under MODE DUPL above.

5.3.5 Using CODE

In a "batch" report, the CODE is a "literal" prefixed to every print line. However, with a file handler you can designate the CODE clause for the passing of any additional information from the program to the file handler. Normally this is to control the output, rather than for output as part of the data. For example, given the necessary hardware and software prerequisites, you might decide to develop a file handler, and use the CODE clause, as follows:

<table>
<thead>
<tr>
<th>Your possible file handler</th>
<th>Likely use of CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microfiche output</td>
<td>Key for fiche indexing.</td>
</tr>
<tr>
<td>Remote transmission</td>
<td>Identifier for remote printer.</td>
</tr>
<tr>
<td>Spooled restart system</td>
<td>Key, page number, etc., for restart.</td>
</tr>
</tbody>
</table>

The contents of the program's CODE operand appears in the file handler as the LINKAGE SECTION item L-RCA-CODE-VAL as described later in this section.

5.3.6 Actions of an Independent Report File Handler

A file handler must perform certain mandatory housekeeping functions in order to execute correctly in combination with the report writer code. However, the method by which the report data is physically output is left entirely to the file handler.

If the file handler may be required to service more than one report file or INITIATED report simultaneously, or be required to be usable by more than one program simultaneously, it should store its intermediate results in the File Control Area or Report Control Area, where certain locations have been reserved for discretionary use by the file handler.

The following are the mandatory housekeeping actions that must be performed by every file handler. The standard locations referred to are described later in this section.

1. If the location L-FCA-ACT-IND is anything other than "0" or "9", "OPEN" the report "file". The file handler may interpret the OPEN function and the nature of the "file" in any way it wishes. The value of L-FCA-ACT-IND indicates the type of OPEN required (OUTPUT, EXTEND, or other).

2. If the location L-RCA-ACT-IND is "6", "INITIATE" the report. The file handler may perform the INITIATE action in any way it wishes.

3. If there is data to output, indicated by the field L-PRC-BYTE-CNT being non-zero, "print" the data. The exact nature of the "printing" activity is left entirely to the file handler and may differ widely from a "batch" WRITE statement. The following locations will be needed to accomplish this:

   L-RCA-VERI-POSN contains the current vertical position or zero. If it is zero the file handler should execute a "form feed". This action should be omitted if L-RCA-PAGE-LIM is zero, indicating that the report is not divided into pages.
5.3 – Developing an Independent Report File Handler

L-FCA-SUPP-PFD, if set to 1, indicates that the program has a FIRSTPAGE NO ADVANCING clause in the SELECT...ASSIGN. If so, the "form feed" is not done and instead the current position is assumed to be "line 1". It is cleared automatically by the control routine.

L-RCA-LINE-CNTR contains the target vertical position. By subtracting L-RCA-VERT-POSN from this, you obtain the distance to be advanced.

4. If the location L-RCA-ACT-IND is "8", "TERMINATE" the report. The file handler may perform the TERMINATE action in any way it wishes.

5. If the location L-FCA-ACT-IND is "9", "CLOSE" the report "file". The file handler may interpret the CLOSE function in any way it wishes.

By convention, there will never be more than one action from the possibilities OPEN, INITIATE, "print", TERMINATE, and CLOSE on any entry.

5.3.7 File Handler LINKAGE Areas

All the information required by the file handler is passed to it via optional user parameters and three standard LINKAGE areas, of which source copies are provided with this product.

1. Optional Leading Parameters: User-Defined Parameters

There may be any number of parameters at the start of the LINKAGE SECTION pre-determined by the user and specified through the USING phrase of the MODE clause. See 2.2 Report Files for a description of this phrase.

2. Parameter 1: File Control Area

The File Control Area contains information relating to the current report file. A File Control Area is set up for each report file that has a MODE clause.

<table>
<thead>
<tr>
<th>01 L-FCA-CNTRL-AREA.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length in bytes of CODEs for this file.</td>
</tr>
<tr>
<td>*= 0 if CODE clause is not present for any reports in file.</td>
</tr>
<tr>
<td>03 L-FCA-CODE-LEN PIC S9(4) COMP.</td>
</tr>
<tr>
<td>Maximum record length in bytes for File.</td>
</tr>
<tr>
<td>03 L-FCA-REC-LEN PIC S9(4) COMP.</td>
</tr>
<tr>
<td>*Action indicator byte.</td>
</tr>
<tr>
<td>*Action performed is open to interpretation by the designer.</td>
</tr>
<tr>
<td>*&quot;0&quot; = no action against file</td>
</tr>
<tr>
<td>*&quot;1&quot; = file to be opened output</td>
</tr>
<tr>
<td>*&quot;2&quot; = file to be opened extend</td>
</tr>
<tr>
<td>*&quot;5&quot; = file to be opened in irregular manner</td>
</tr>
<tr>
<td>*&quot;9&quot; = file to be closed</td>
</tr>
<tr>
<td>03 L-FCA-ACT-IND PIC X.</td>
</tr>
</tbody>
</table>
*File status indicator.
*Initial value = low-values (hex '00')
"0" = file not opened
"1" = file opened output
"2" = file opened extend
"5" = file opened in irregular manner
"9" = file closed
03 L-FCA-STAT-IND PIC X.

*Count of reports initiated for this file.
*Incremented by 1 on INITIATE, decremented by 1 on TERMINATE;
*initially zero, must never become negative;
*used to warn on CLOSE if some report(s) still initiated
03 L-FCA-REPS-INT PIC S9(4) COMP.

*MODE mnemonic name:
*up to 4 characters (padded with spaces) from the MODE clause
03 L-FCA-MODE-MNEM PIC X(4).

*File Status:
*If L-FCA-FS-IND (below) = "1", file handler control routine will
*always pass back any file status here without abandoning the run;
*Otherwise, file handler control routine will abandon the run with
*an error message if a serious error is indicated in File Status.
03 L-FCA-STATUS PIC XX.

*File Status indicator: set if SELECT has a FILE STATUS clause.
03 L-FCA-FS-IND PIC X.

*Unassigned:
03 FILLER PIC X.

*Locations reserved for communication between file handler
*and report writer:
*integer of DUPLICATED clause if present; zero if no such clause
03 L-FCA-DUP-FACT PIC S9(4) COMP.

*Recording mode = "F", "V", "U", "S" or space if not specified
*taken from RECORDING MODE clause of the FD;
*may be taken into account or ignored by the file handler
*(report record is always variable-length; (see 2.2.3))
03 L-FCA-REC-MODE PIC X.

*Indicator set = "1" if PAGE BUFFER clause present:
03 L-FCA-PBUF-IND PIC X.

*First four characters of file name:
03 L-FCA-FILE-PFIX PIC X(4).

*Locations reserved for use by file-handler:
*Address of block of DUPLICATED FCA's, if applicable
03 L-FCA-DUP-ADR PIC X(4).
*General-purpose location used as by file handler
03 L-FCA-GEN-LOCN PIC S9(4) COMP.
### 5.3 – Developing an Independent Report File Handler

#### 3. Parameter 2: Report Control Area

The Report Control Area is set up by the Precompiler for each report that is directed to a file that has a **MODE** clause. In the user program, the Report Control Area is identified by the report-name.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-RCA-CNTRL-AREA</td>
<td>PAGE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-PAGE-CNTR</td>
<td>LINE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-LINE-CNTR</td>
<td>Vertical position pointer. Shows the actual current position vertically on the page.</td>
</tr>
<tr>
<td>L-RCA-VERT-POSN</td>
<td>Line byte count override. Used internally only. Will = -1 in file handler.</td>
</tr>
<tr>
<td>L-RCA-LINE-SIZE</td>
<td>*PAGE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-PAGE-CNTR</td>
<td>*LINE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-LINE-CNTR</td>
<td>*Vertical position pointer. Shows the actual current position vertically on the page.</td>
</tr>
<tr>
<td>L-RCA-VERT-POSN</td>
<td>*Line byte count override. Used internally only. Will = -1 in file handler.</td>
</tr>
<tr>
<td>L-RCA-LINE-SIZE</td>
<td>*PAGE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-PAGE-CNTR</td>
<td>*LINE-COUNTER, referred to in program as</td>
</tr>
<tr>
<td>L-RCA-LINE-CNTR</td>
<td>*Vertical position pointer. Shows the actual current position vertically on the page.</td>
</tr>
<tr>
<td>L-RCA-VERT-POSN</td>
<td>*Line byte count override. Used internally only. Will = -1 in file handler.</td>
</tr>
</tbody>
</table>

**Example:**

```
03 L-RCA-CNTRL-AREA.
03 L-RCA-PAGE-CNTR PIC S9(9) COMP.
03 L-RCA-LINE-CNTR PIC S9(9) COMP.
03 L-RCA-VERT-POSN PIC S9(9) COMP.
03 L-RCA-LINE-SIZE PIC S9(4) COMP.
```
*Line Limit; i.e. the logical size of the report line
*from the LINE LIMIT clause in the RD, or its default:
  03 L-RCA-LINE-LMT       PIC S9(4) COMP.

*Action indicator byte:
**"0" = no change to report status
**"6" = report to be initiated
**"7" reserved
**"8" = report to be terminated
  03 L-RCA-ACT-IND        PIC X.

*Report status indicator:
**"0" = report never initiated
**"6" = report initiated
**"8" = report terminated
  03 L-RCA-STAT-IND        PIC X.

*Report-name from RD statement:
  03 L-RCA-REP-NAME        PIC X(30).

*Print record format indicator: always = "0"
  03 L-RCA-PRT-FMT         PIC X.

*Line column width override: (1 byte binary held as character)
  03 L-RCA-COL-WIDTH       PIC X.

*Page Limit from PAGE LIMIT clause RD; zero if no PAGE clause.
  03 L-RCA-PAGE-LIM        PIC S9(4) COMP.

*Length of CODE, or zero if no CODE clause.
*The length of the CODE is determined from the length of the CODE
**"literal" or, if the "identifier" form of CODE clause is used,
*from the difference between the record length of the report file
*(as given in the RECORD or BLOCK CONTAINS clause of the FD)
*and the LINE LIMIT (or its default value),
*allowing for the normal carriage control characters.
  03 L-RCA-CODE-LEN        PIC S9(4) COMP.

*Error flag: zero = no error.
*May be set by file handler to non-zero value representing a
*standard error condition for reporting via a subroutine.
*If it is given a value which does not correspond with a known
*error message, the message UNKNOWN ERROR TYPE will be output
*together with information that usually accompanies any message.
  03 L-RCA-ERR-FLG         PIC S9(4) COMP.

*Error code detected by file handler:
  03 L-RCA-FH-ERR          PIC S9(4) COMP.

*Report Number from REPORT-NUMBER location
*zero if DUPLICATED clause not in use.
  03 L-RCA-REP-NUM         PIC S9(4) COMP.

*Location reserved for future communication between file handler
*and report writer.
  03 L-RCA-COMM-AREA       PIC X(8).
**Location reserved for internal use (not by file handler):**

**Data items used by Page Buffer:**

*Current margin offset established by SET COLUMN (1 = no margin)*

| 03  | L-RCA-MARG          | PIC S9(4) COMP. |

*Forced absolute line number set up by SET LINE TO integer.*

| 03  | L-RCA-ABS-LNO       | PIC S9(4) COMP. |

*Indicator set to "H" "HOLD" status, otherwise space.*

| 03  | L-RCA-HOLD-IND      | PIC X. |

**General locations used by file handlers:**

| 03  | FILLER              | PIC X(11). |

**Data items used by Page Buffer: (not required by file handler)**

*Current physical position;*

| 03  | L-RCA-PHYS-POSN     | PIC S9(4) COMP. |

*Lowest line no. having line buffered.*

| 03  | L-RCA-STL-LOW       | PIC S9(4) COMP. |

*Highest line no. having line buffered.*

| 03  | L-RCA-STL-HIGH      | PIC S9(4) COMP. |

*Maximum value permitted for LINE-COUNTER.*

| 03  | L-RCA-MAX-LCT       | PIC S9(4) COMP. |

*Maximum physical size in bytes of "line" of data;*

*this is the largest value that can ever be held in "byte count"

*(L-PRC-BYTE-CNT) of any print line.*

*It differs from the line limit (L-RCA-LINE-LMT) in that*

*(a) the line limit is a maximum for checking purposes only and*

*may never be attained in any actual line of the report,*

*(b) lines may contain formatting characters that do not occupy*

*a column of print, e.g. start- and end-sequence of a "style" (underline, bold etc.)*  
*shift out and shift in characters used by Kanjii (DBCS)*

| 03  | L-RCA-BYTE-LMT      | PIC S9(4) COMP. |

*Number of purely formatting characters override; if non-zero, the*

*value (L-PRC-BYTE-CNT - L-RCA-FMT-CNT) overrides L-PRC-END-COL;*

*it must always be cleared by file handler.*

| 03  | L-RCA-FMT-CNT       | PIC S9(4) COMP. |

*Indicators, for general use by file handler.*

| 03  | L-RCA-WORK-IND      | PIC X OCCURS 4. |

*Code Value; zero length if no CODE clause in RD*

*Initial value set from CODE clause.*

*May be referred to in user program as CODE-VALUE [IN report-name]*

| 03  | L-RCA-CODE-VAL      | |

| 05  | L-RCA-CODE-CHA      | PIC X OCCURS 0 TO 4095 TIMES DEPENDING ON L-RCA-CODE-LEN. |

*End of report control area.*
4. **Parameter 3: Report Data**

This parameter contains the **data to be written**. For OPEN, INITIATE, TERMINATE and CLOSE operations, this parameter is a **dummy** with a value of zero in the counter L-PRC-BYTE-CNT. The number of bytes to be written may be reduced to the value in L-RCA-LINE-SIZE when it is non-negative.

<table>
<thead>
<tr>
<th>01 L-PRC-PRINT-REC.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Physical byte count, not including this 4-byte header;</em></td>
</tr>
<tr>
<td><em>will be zero when no data is to be output;</em></td>
</tr>
<tr>
<td><em>gives no. of bytes to be written, unless overridden by</em></td>
</tr>
<tr>
<td><em>L-RCA-LINE-SIZE set non-negative.</em></td>
</tr>
<tr>
<td>03 L-PRC-BYTE-CNT    PIC S9(4) COMP.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>03 L-PRC-HEAD.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Indicator bits, layout is as follows:</em></td>
</tr>
<tr>
<td><em>bit 0-3: unused</em></td>
</tr>
<tr>
<td><em>bit 4: set if data contains unresolved STYLE sequence(s)</em></td>
</tr>
<tr>
<td><em>bit 5: set if data consists only of formatting (STYLE) characters</em></td>
</tr>
<tr>
<td><em>bit 6: reserved</em></td>
</tr>
<tr>
<td><em>bit 7: set if print data must be flushed to spaces.</em></td>
</tr>
<tr>
<td><em>2nd byte is reserved.</em></td>
</tr>
<tr>
<td>05 L-PRC-HEAD-BYTES PIC S9(4) COMP.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>03 L-PRC-DATA.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Data to be written;</em></td>
</tr>
<tr>
<td>05 L-PRC-DATA-CHA PIC X</td>
</tr>
<tr>
<td>OCCURS 0 TO 256 TIMES DEPENDING ON L-PRC-BYTE-CNT.</td>
</tr>
</tbody>
</table>

5.3.8 **Sample Independent Report File Handler**

The following is the complete code of the standard **MODL** file handler (used for writing to a single report file from a modular system). This file handler is one of the simplest, and so it may be used as a model for your own user-written file handlers, which will require almost all the code that follows, except perhaps for the incrementing and decrementing of the "nest depth" (**W-OPEN-NEST**) which is peculiar to this file handler.

At least one file handler is supplied in source form with this product, giving you a machine-readable copy of most of this coding.

```cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. CRFHMODL.
*
*Independent Report Writer file handler for use in modular program*
*
*Purpose:
* Used when several separately-compiled modules need to access
* the same report file without continually opening and closing.  
* This file handler does not currently handle more than 1 file.
```
Method of use:

1. The control program does one OPEN at start of processing and one CLOSE at very end of all processing. (To satisfy the syntax it should have a minimal REPORT SECTION which is not used.)

2. Each subprogram including control module has an FD & does OPEN, some GENERATEs and CLOSE each time it is CALLED, as though it were the only module using the file.

This file handler ignores all "nested" OPENs and CLOSEs.

Typical sequence of operations:

a. Control program does OPEN OUTPUT which is actioned.

b. Subprogram does OPEN which is ignored.

c. Subprogram does INITIATE, several GENERATE's & TERMINATE.

d. Subprogram does CLOSE which is ignored.

e. Many other subprogram CALLs occur in same way.

f. Control program does CLOSE which is actioned.

Subprograms have to do the (dummy) OPEN and CLOSE as Report Writer requires that every program be logically complete.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT PRINT-FILE
* is always overwritten by name specified in program
ASSIGN TO RPFILE
FILE STATUS IS W-FCA-STATUS.

DATA DIVISION.
FILE SECTION.
FD PRINT-FILE
LABEL RECORDS STANDARD.
*THIS FILE MAY ALSO BE DESIGNATED "RECORDING MODE V"
01 F-PRINT-RECORD.
05 F-PRINT-DATA.
07 F-PRINT-CHAR PIC X OCCURS 512.

WORKING-STORAGE SECTION.
*General locations:
01 WS-GENERAL.
05 W-FEED PIC S9(4) COMP.
*This location contains the "nest" level of OPEN/CLOSE statements:
05 W-OPEN-NEST PIC S9(4) COMP VALUE ZERO.
*Pointer to record data:
05 W-PTR PIC S9(4) COMP.
*Indicator set = "Y" if a the given line of data has been printed:
05 W-DATA-PRINTED PIC X.
*File name for print file:
05 W-FILE-NAME PIC X(79).
*Indicator set = "1" if an error occurs on writing.
  05 W-PRINT-ERROR PIC X.
*File status:
  05 W-FCA-STATUS PIC XX.

*Standard Linkage:
  LINKAGE SECTION.

*Parameter 1: File Control Area
  COPY RWFCACOM.

*Parameter 2: Report Control Area
  COPY RWRCACOM.

*Parameter 3: Print Line
  COPY RWPLNCOM.

PROCEDURE DIVISION USING   L-FCA-CNTRL-AREA
                             L-RCA-CNTRL-AREA
                             L-PRC-PRINT-REC.

*  FILE-HANDLER-CONTROL SECTION.
  FHC-ENTRY.
  *Assume successful operation:
    MOVE "00" TO W-FCA-STATUS
  *Open file if indicated:
    IF   L-FCA-ACT-IND > "0" AND < "9"
      PERFORM OPEN-FILE.
  *Initiate report if indicated:
    IF   L-RCA-ACT-IND = "6"
      PERFORM INITIATE-REPORT.
  *Print data if any data present:
    IF   L-PRC-BYTE-CNT NOT = 0
      PERFORM PRINT-DATA.
  *Terminate report if indicated:
    IF   L-RCA-ACT-IND = "8"
      PERFORM TERMINATE-REPORT.
  *Close file if indicated:
    IF   L-FCA-ACT-IND = "9"
      PERFORM CLOSE-FILE.
  MOVE W-FCA-STATUS TO L-FCA-STATUS.
  FHC-EXIT.
  EXIT PROGRAM.

*Main sections to print the data line.
*
  PRINT-DATA SECTION.
  PDA-ENTRY.
  *check not too many characters for print line:
    IF   L-PRC-BYTE-CNT > 512
      DISPLAY "CRFHMODL Error: Line too long - truncated".
    MOVE "N" TO W-DATA-PRINTED
*If vertical position is zero in paged report, page feed required:
  IF   L-RCA-VERT-POSN = 0
  AND  L-RCA-PAGE-LIM NOT = 0
  AND  W-PRINT-ERROR = "0"

*If FIRST PAGE NO ADVANCING was coded in SELECT statement, 
*omit first page advance (this ind. is reset by control s/r)
  IF   L-FCA-SUPP-PFD = "1"
       MOVE 1 TO L-RCA-VERT-POSN
  ELSE PERFORM TOP-OF-PAGE.

*Now write data after advancing required distance, unless it was 
*written on line 1 when we advanced to top of page:
  IF   W-DATA-PRINTED = "N"
  AND  W-PRINT-ERROR = 0
  PERFORM AFTER-ADVANCING.

*Routine to print at top-of-page.
  TOP-OF-PAGE SECTION.
  TOP-ENTRY.
  *If LINE-COUNTER = 1 we can print data at top-of-form, 
  *otherwise we print blank lines first;
  IF   L-RCA-LINE-CNTR > 1
       MOVE SPACES TO F-PRINT-DATA
  ELSE PERFORM FILL-RECORD
       MOVE "Y" TO W-DATA-PRINTED.
       WRITE F-PRINT-RECORD AFTER ADVANCING PAGE
  *If bad File Status, set indicator to prevent further writing 
  *in case user program has FILE STATUS which is not examined.
  IF   L-FCA-STATUS NOT = "00"
       MOVE "1" TO W-PRINT-ERROR.
       MOVE 1 TO L-RCA-VERT-POSN.

*Routine to print with appropriate advance (if any)
  AFTER-ADVANCING SECTION.
  ADV-ENTRY.
  PERFORM FILL-RECORD
  SUBTRACT L-RCA-VERT-POSN FROM L-RCA-LINE-CNTR GIVING W-FEED
  WRITE F-PRINT-RECORD AFTER ADVANCING W-FEED.
  IF   L-FCA-STATUS NOT = "00"
       MOVE "1" TO W-PRINT-ERROR.

*Move data into record, allowing fir possible CODE
  FILL-RECORD SECTION.
  FLR-ENTRY.
  IF   L-RCA-CODE-LEN = 0
  AND  L-RCA-MARG NOT > 1
       MOVE L-PRC-DATA TO F-PRINT-RECORD
  ELSE MOVE 1 TO W-PTR
       IF   L-RCA-CODE-LEN > 0
            MOVE L-RCA-CODE-VAL TO F-PRINT-RECORD
            ADD L-RCA-CODE-LEN TO W-PTR
       ELSE MOVE SPACES TO F-PRINT-RECORD
       END-IF

5.3 – Developing an Independent Report File Handler
IF L-RCA-MARG > 0
   ADD L-RCA-MARG TO W-PTR
   SUBTRACT 1 FROM W-PTR
END-IF
STRING L-PRC-DATA DELIMITED BY SIZE INTO F-PRINT-RECORD
   WITH POINTER W-PTR.

*OPEN the report file
OPEN-FILE SECTION.
OPF-ENTRY.
*If OPEN not OUTPUT or EXTEND, signal error 44 and assume OUTPUT:
   IF L-FCA-ACT-IND = "5"
      MOVE 44 TO L-RCA-ERR-FLG
      MOVE "1" TO L-FCA-ACT-IND.
*Does not really OPEN the file if the OPEN is "nested".
   IF W-OPEN-NEST = 0
*Get name of print file
   IF RETURN-CODE = 0
*OPEN OUTPUT or EXTEND, depending on the Action Indicator.
      IF L-FCA-ACT-IND = "1"
         OPEN OUTPUT PRINT-FILE
      ELSE OPEN EXTEND PRINT-FILE
      END-IF
   END-IF
   IF L-FCA-STATUS NOT = "00"
      MOVE "1" TO W-PRINT-ERROR.
*Increment nest level:
      ADD 1 TO W-OPEN-NEST.

*Initiate the report
INITIATE-REPORT SECTION.
INT-ENTRY.
*In this section we do any action consistent with INITIATING
   the report. In this routine nothing need be done.

*Terminate the report
TERMINATE-REPORT SECTION.
TRM-ENTRY.
*In this section we do any action consistent with TERMINATING
   the report. In this routine nothing need be done.

*Close the report file
CLOSE-FILE SECTION.
CLF-ENTRY.
*Does not really CLOSE the file if the CLOSE is "nested".
   SUBTRACT 1 FROM W-OPEN-NEST
   IF W-OPEN-NEST = 0
      CLOSE PRINT-FILE.
      MOVE "0" TO W-PRINT-ERROR.
*End of file handler.
6

Migration from OS/VS or DOS/VS COBOL Report Writer

This part is a guide to help you to move all the Report Writer code in your program sources successfully from OS/VS COBOL or DOS/VS COBOL Report Writer to new Report Writer.
6.1 **Re-compiling OS/VS and DOS/VS COBOL Sources**

You will be able to obtain correct working versions of all your OS/VS COBOL or DOS/VS COBOL Report Writer programs using new Report Writer. The migration of most Report Writer programs require no changes to the current Report Writer code. However, new Report Writer produces more information, in the form of informational (severity "I") messages to make certain processes and assumptions clear to you. You should therefore expect a number of these severity "I" messages to be produced when you submit a typical migrated Report Writer program to the precompiler.

The precompiler is also stricter than both OS/VS COBOL and DOS/VS COBOL in its checking of your Report Writer for adherence to syntax rules and in the additional consistency checks that it performs. In some cases, the precompiler will produce warning (severity "W") - or occasionally even severity "E" or "S" messages - in a program that the previous compiler accepted without any diagnostic messages. (You may find that the situations that cause these messages recur in all the programs originally written by the same individuals, because a certain style of coding is used consistently.) However, even severity "W" messages normally do not require a change to your program source, since your program will behave at run time exactly as it did under the previous compiler.

The following sections list the messages that have occurred when a large sample of "clean" migrated Report Writer programs were precompiled, compiled and run. They are listed in order of severity. This part outlines what action, if any, you should take for each severity level. A more detailed explanation of the messages will be found in Appendix E, where all diagnostic messages are listed.

### 6.1.1 Informational (I-level)

Informational messages do not indicate a fault of any kind and do not require you to take any action. You can suppress the listing of I-level messages by specifying the option FLAG(W). If you are running under CMS or TSO, the option MSGL(3) may be used to prevent I-level messages being displayed on your screen.

The following severity-I messages may result from "clean" migrated Report Writer programs.

**RW-001-I No Report Writer data entries were found in this program.**

**Meaning** This message should be expected if your program has no REPORT SECTION. You may wish to precompile every COBOL program since you cannot easily tell which programs contain Report Writer code and which do not. You may decide to place a NORW directory at the start of your source program to inhibit the scan for non-existent Report Writer code. (See Installation and Operation.)

**Remedy** None required.
**RW-008-I FD has record definition but no RECORD CONTAINS: compiler may assume variable-length.**

**Meaning**
This message may result if your report file FD is followed by a record description and the FD entry has no RECORD CONTAINS clause. New Report Writer creates a record description after your FD to enable it to output generated Report Writer records. If no RECORD CONTAINS clause is present, it sets up the record with a length equal to the longest actual line to be written (rounded up to a multiple of 4 and incremented by 1 for any carriage control character). Since your FD already has a record description, its length is likely to be different from the length of Report Writer's generated record. If RECORDING MODE F is specified, a compiler error will result (see 6.1.3 below). If not, the result may be unexpected variable-length records.

**Remedy**
It is probable that the record description after the FD entry was originally coded under the mistaken assumption that it is required. A record description is not required after the FD for a report file, unless the program issues WRITEs independently to the same file. Check whether the record description is referred to and, if not, remove it.

If the record description is referred to, you should add a RECORD CONTAINS clause to the FD entry, specifying the length in bytes of the record description that follows the FD.

**RW-030-I PAGE LIMIT will never be reached.**

**Meaning**
This message implies that your PAGE LIMIT is larger than it need be since no report line will ever reach it. There are two cases where this may happen. Either you have a PAGE FOOTING group, using absolute LINE numbers, which does not reach the PAGE LIMIT. Or, you have no PAGE FOOTING group but the value of LAST DETAIL or FOOTING is less than the PAGE LIMIT so that body groups can never reach it. Possibly the value of PAGE LIMIT was coded under the mistaken assumption that it should represent the physical size of the printed page.

**Remedy**
None required, but to eliminate the message you may reduce the value of the PAGE LIMIT to that of the last line of the PAGE FOOTING group, or to the value of LAST DETAIL or FOOTING.

**RW-085-I New reserved word accepted as data-name.**

**Meaning**
This message appears when you use one of Report Writer's new reserved words as a SOURCE or SUM operand, for example "SOURCE FUNC" or "SUM COLS". Report Writer detects this from the context and allows the clause as you intended so that the new reserved words have as little impact as possible on migrated programs.

**Remedy**
None required, but to eliminate the message you may edit the program to change the data-name, for example from FUNC to WS-FUNC.
**RW-110-I Elementary item has no size: will not be output.**

**Meaning**
This message appears when your elementary SOURCE item has no PICTURE clause. This technique has been used frequently by programmers to define "dummy" entries in a "totals-only" report, so that there will be a SOURCE identifier to correspond with every SUM identifier. For example:

```
01 DUMMY-GROUP     TYPE DETAIL.
   03 LINE PLUS 1.
   05 SOURCE AMOUNT.
```

**Remedy**
No action is required, but if you wish to eliminate the message, simply attach a PICTURE to the entry:

```
01 DUMMY-GROUP     TYPE DETAIL.
   03 LINE PLUS 1.
   05 PIC Z(6)9   SOURCE AMOUNT.
```

**RW-146-I No GENERATE issued for this DETAIL.**

**Meaning**
The DETAIL group referred is probably a dummy group, coded in a "totals-only" report, as illustrated above under message RW-110. Without the SOURCE clauses in the dummy group, OS/VS COBOL and DOS/VS COBOL will not SUM these fields.

**Remedy**
None required. New Report Writer does not require dummy groups in this situation, because it does not require a SOURCE to correlate with each SUM. So, if this is the only purpose of the dummy group, you may eliminate this message by removing the dummy group.

**RW-161-I SUM will be totalled UPON generation of xx due to SOURCE SUM correlation.**  or  
**RW-162-I SUM will be totalled also UPON generation of xx.**

**Meaning**
One of these messages should appear for each SUM identifier in your program which is not the name of another REPORT SECTION item. See Appendix E under these messages for additional commentary.

**Remedy**
None required.

**RW-200-I MODE PRNT has been assumed for file due to CODE clause.**

**Meaning**
This message will appear if a report file has more than one RD, and they have CODE clauses with "literals" which are not all of the same length (counting an absent CODE clause as a "zero-length literal"). Clearly, this situation could not be implemented easily by means of normal WRITE statements, so the precompiler uses an independent report file handler. The PRNT file handler, which produces a similar output to basic COBOL WRITExes, is used if no other file handler is specified (in a MODE clause or by a parameter to the precompiler). For more information about file handlers, see 2.2 Report Files.
**Remedy** None required. However, if the FD is followed by a record description, message RW-180-E will also appear (see below). You might consider removing all CODE clauses from the program and assigning each affected report to a separate print file instead, thus avoiding the need to run the utility print program which makes multiple passes through your print file.

**RW-213-I Value xx assumed for LINE LIMIT.**

**Meaning** This message will appear for each RD in your program. It shows the rightmost position permitted for any character in a report line.

**Remedy** None required.

### 6.1.2 Warning (W-level)

Warning messages indicate that the code found contains an infringement of the syntax, but that the program will execute as intended. In general, you need not correct W-level violations to ensure that the program will be compatible with OS/Vs COBOL or DOS/Vs COBOL. It is possible for a program source which produces no messages under the previous compiler to produce several W-level messages when precompiled.

The following W-level messages may result from "clean" migrated Report Writer programs. A simple change is suggested for each case, should you prefer to alter the source to make it clearer and easier to maintain, and to conform to the ANS-74/85 Report Writer standard.

**RW-019-W Report xx in FD has no REPORT SECTION entry.**

**Meaning** This message indicates that a REPORT clause in an FD refers to a report-name that has no RD entry in the REPORT SECTION. OS/Vs COBOL, DOS/Vs COBOL and the precompiler all ignore the report-name.

**Remedy** No action is required but if you want to remove this message, delete the unused report-name from your program.

**RW-031-W PAGE LIMIT increased to value of LAST DETAIL or FOOTING.**

**Meaning** This message results when the PAGE LIMIT is less than the value of LAST DETAIL or FOOTING, for example:

<table>
<thead>
<tr>
<th>RD</th>
<th>REPORT-ONE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PAGE LIMIT 50 LINES</td>
</tr>
<tr>
<td></td>
<td>LAST DETAIL 54</td>
</tr>
<tr>
<td></td>
<td>FOOTING 56.</td>
</tr>
</tbody>
</table>

**Remedy** No action is required since both OS/Vs COBOL, DOS/Vs COBOL and the precompiler all override the inadequate PAGE LIMIT. If you want to avoid this message, increase the PAGE LIMIT integer to the value of LAST DETAIL or, if present, FOOTING:

<table>
<thead>
<tr>
<th>RD</th>
<th>REPORT-ONE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PAGE LIMIT 56 LINES</td>
</tr>
<tr>
<td></td>
<td>LAST DETAIL 54</td>
</tr>
<tr>
<td></td>
<td>FOOTING 56.</td>
</tr>
</tbody>
</table>
RW-051-W duplicated CONTROL: ignored.

**Meaning**  This message implies that your report description lists the same control field twice, for example:

```
RD  REPORT-ONE
    PAGE LIMIT 60 LINES
    CONTROLS ARE ACCOUNT-CODE
          ACCOUNT-CODE
          DISTRICT-CODE.
```

**Remedy**  No action is required but if you want to remove this message you should simply remove the second occurrence of the control, since it serves no purpose.

RW-064-W LINE entries nested: previous LINE assumed level xx.

**Meaning**  This message is issued when two LINE clauses are at different levels. The commonest instance is where a group with more than one LINE clause has the first LINE clause at the 01-level:

```
01  TYPE REPORT FOOTING LINE NEXT PAGE.
    03  COLUMN 1 ...
    03  LINE IS PLUS 1.
    05  COLUMN 1 ...
```

**Remedy**  No action is required because OS/VS COBOL, DOS/VS COBOL and the precompiler allow this situation. If you want to remove this message, restructure the code as follows:

```
01  TYPE REPORT FOOTING.
    03  LINE NEXT PAGE.
    05  COLUMN 1 ...
    03  LINE IS PLUS 1.
    05  COLUMN 1 ...
```

RW-070-W COLUMN entries following LINE assumed to be subordinate to it.

**Meaning**  This message will appear if a COLUMN entry is not at a level lower than the preceding LINE entry, as in the following example:

```
01  TYPE PAGE HEADING.
    03  LINE 1 COLUMN 1 ...
    03  COLUMN 21 ...
```

The second COLUMN clause is not strictly subordinate to the LINE clause.

**Remedy**  No action is required because OS/VS COBOL, DOS/VS COBOL and new Report Writer allow this arrangement of clauses. If you wish to eliminate this message, you may restructure the code as follows:

```
01  TYPE PAGE HEADING.
    03  LINE 1.
    05  COLUMN 1 ...
    05  COLUMN 21 ...
```
**RW-072-W** Recurrence of same absolute LINE merged with preceding.

**Meaning**
This message will appear if the same absolute LINE clause is repeated in successive entries with different COLUMN clauses, for example:

```
01  TYPE PAGE HEADING.
   03  LINE 1  COLUMN 10 ...
   03  LINE 1  COLUMN 30 ...
```

This is a loose interpretation of the ANS 68 standard allowed by the older compilers.

**Remedy**
No action is required because OS/VS COBOL, DOS/VS COBOL and new Report Writer allow this arrangement of clauses. If you wish to eliminate this message, you may restructure the code as follows:

```
01  TYPE PAGE HEADING.
   03  LINE 1.
      05  COLUMN 10 ...
      05  COLUMN 30 ...
```

**RW-096-W** LINE clauses in group will cause it to extend beyond bottom limit.

**Meaning**
This message will appear if a report group ends too far down the page. For example:

```
RD  REPORT-ONE
    PAGE LIMIT IS 60 LINES
    FIRST DETAIL 5
    FOOTING 58.

* 01  TYPE REPORT HEADING.
    03  LINE 1 ...
       05  COLUMN 1 ...
    03  LINE PLUS 64 ...
       05  COLUMN 1 ...

* 01  TYPE PAGE HEADING.
    03  LINE 1 ...
       05  COLUMN 1 ...
    03  LINE PLUS 4 ...
       05  COLUMN 1 ...
```

Both report groups in this example will invoke this message. The REPORT HEADING stretches down to beyond the PAGE LIMIT (the absolute limit on all report line positions). The PAGE HEADING encroaches into the FIRST DETAIL position. (Note, if the second line of the PAGE HEADING is an absolute line such as LINE 5, instead of the equivalent LINE PLUS 4, OS/VS COBOL and DOS/VS COBOL do give an error message.)
**Remedy**  
No action is required because OS/VS COBOL, DOS/VS COBOL and the precompiler will all accept the REPORT HEADING and PAGE HEADING as described and will begin the DETAIL groups immediately after the PAGE HEADING. To clear these conditions, you should aim to adjust the PAGE LIMIT sub-clauses so that they truly represent the regions of your page, without affecting any other part of the report. If you alter the PAGE LIMIT, and require complete compatibility, it is advisable to include both a LAST DETAIL and a FOOTING clause to ensure that your DETAIL and CH/CF groups come down to the same lowest position as before. The above code can now be rewritten as follows:

```cobol
RD  REPORT-ONE
    PAGE LIMIT IS 60 LINES
    FIRST DETAIL 6
    LAST DETAIL 58
    FOOTING 58.
```

**RW-106-W** This RH group will have own page: NEXTGROUP NEXT PAGE assumed.

**Meaning**  
This message appears if your REPORT HEADING group has no NEXT GROUP NEXT PAGE clause (which usually means that the first PAGE HEADING should follow it on the same page) but there is no room to print the REPORT HEADING above the first PAGE HEADING between HEADING and FIRST DETAIL, as in this example:

```cobol
RD  REPORT-ONE
    PAGE LIMIT IS 60 LINES
    HEADING 1
    FIRST DETAIL 4.
    *
    01  TYPE IS REPORT HEADING.
        03  LINE 1.
            05  COLUMN 1    PIC  X(40)   SOURCE REPORT-NAME.
        03  LINE PLUS 2.
            05  COLUMN 1    PIC  X(40)   VALUE ALL "*".
    *
    01  TYPE IS PAGE HEADING.
        03  LINE PLUS 1.
            05  COLUMN 1    PIC  X(40)   SOURCE COMPANY-NAME.
```

**Remedy**  
No action is required because OS/VS COBOL and DOS/VS COBOL Report Writer also place the REPORT HEADING on a page by itself and your results will be identical. If you wish the REPORT HEADING to temporarily push down the PAGE HEADING, omit the FIRST DETAIL clause. If you wish to avoid this message, add the NEXT GROUP NEXT PAGE clause:

```cobol
01  TYPE IS REPORT HEADING   NEXT GROUP NEXT PAGE.
```
**Meaning**

This message appears if your REPORT FOOTING group begins with an absolute LINE without a NEXT PAGE phrase (which usually means that it should follow the last PAGE FOOTING on the same page) but the LINE number is not greater than the last LINE position of the PAGE FOOTING, as in this example:

```
RD REPORT-ONE
  PAGE LIMIT IS 60 LINES
  LAST DETAIL 54
  FOOTING 56.

  01 TYPE IS PAGE FOOTING.
  03 LINE 58.
    05 COLUMN 1 PIC X(10) VALUE "REPORT ABC".

  01 TYPE IS REPORT FOOTING.
  03 LINE 1.
    05 COLUMN 1 PIC X(13) VALUE "END OF REPORT".
```

**Remedy**

No action is required because OS/VS COBOL and DOS/VS COBOL Report Writer also place the REPORT FOOTING on a page by itself and your results will be identical. If you wish to avoid this message, add a NEXT PAGE phrase to the LINE clause of the REPORT FOOTING:

```
01 TYPE IS REPORT FOOTING.
  03 LINE 1 ON NEXT PAGE.
    05 COLUMN 1 PIC X(13) VALUE "END OF REPORT".
```

**Meaning**

These messages may appear if your original program has no INITIATE or TERMINATE for a report. Your program may have operated successfully under the previous compiler without one or both of these statements. ANS standards require that every Report Writer program must perform an INITIATE at the start and a TERMINATE at the end of each report. If the INITIATE is not performed, a run time error message 14 will be issued from your program and the INITIATE will be implicitly performed. If the TERMINATE is not performed, the last set of CONTROL FOOTING groups, the last PAGE FOOTING and the REPORT FOOTING, as applicable, will be lost from your output.

**Remedy**

If the INITIATE is missing, it is advisable to add it to the source. If in doubt, insert it immediately after the OPEN for the report file. If the TERMINATE is missing and you are satisfied that it was not omitted deliberately to prevent the final groups being printed (mentioned in the preceding paragraph), add it to your source. If in doubt, insert it immediately before the CLOSE for the report file. For further information, see 4.1.1 Sequence of Operations.
RW-151-W Superfluous period: ignored.

Meaning  
OS/VS COBOL and DOS/VS COBOL are tolerant of superfluous period characters coming at the end of an entry, for example:

```
05 COLUMN 23 PIC Z(6)9 SOURCE AMOUNT. . .
```

Remedy  
No action is required, but if you want to remove this message, simply remove any extra periods appearing after the first period.

RW-163-W Item not in REPORT SECTION is accumulated on each GENERATE.

Meaning  
This message should never appear in a correct and logical OS/VS COBOL or DOS/VS COBOL program. It suggests a serious discrepancy in your original program. The message indicates that your program has a SUM clause whose operand is not a SOURCE in a DETAIL group or another SUM in your REPORT SECTION. This condition is not flagged by the older compilers but it will not perform any adding and will produce a value of zero in the report. The precompiler, on the other hand, processes the SUM clause by accumulating the field on each GENERATE.

Remedy  
If the report is producing a zero value and you are satisfied with it, you may substitute VALUE ZERO for the SUM clause. If you decide that the field should be accumulated, you should decide whether to allow it to accumulate on each GENERATE. If you decide that it should be accumulated only on the GENERATE of certain DETAIL groups, add the phrase

```
UPON detail-group-name(s)...
```

to the SUM clause.

RW-250-W Item overlaps or is to left of item in same line.

Meaning  
This message appears when the COLUMN entries within a line are not in ascending sequence, or, taking into account their sizes, are found to overlap. OS/VS COBOL and DOS/VS COBOL allow COLUMN entries to appear in any order and to overlap to any extent. It simply stores them into a line initially set to spaces in the order they were coded. Each entry may have any starting COLUMN and byte length. No check is performed as to whether some data has already been stored in any of the target positions. Here is an example:

```
03 LINE PLUS 1.
 05 COLUMN 40 PIC X(22) VALUE "TOTAL COST = ".
 05 COLUMN 54 PIC Z(5)9 SOURCE WS-TOTAL-COST.
 05 COLUMN 34 PIC Z(5)9 SOURCE WS-MONTHLY-COST.
 05 COLUMN 24 PIC X(10) SOURCE MONTH-NAME.
```

Remedy  
If you are satisfied with your current version of the program, then you need not make any change. In the case above, the first field is longer than it need be. The second field overwrites the excess spaces so no harm results. The third field has its COLUMN number "out of sequence" but it does not overlap any other field. It may therefore be left as it is.
Since overlapping has serious implications, you may prefer to remove it. In the case above, the PICTURE may be shortened or simply removed. It is possible that an unintended overlap will become more serious with time, especially when larger values need those higher-order digit positions that might be overlaid by another field. If you cannot rearrange the COLUMN numbers or reduce the PICTURE sizes, try a “staggered” layout, such as the following:

```
$99 $99 $99 $99 $99 - LINE PLUS 1
$99 $99 $99 $99 - LINE PLUS 1
```

If your fields are simply out of order, rearranging them by ascending COLUMN number will make it easier for you to detect genuine cases of overlap. (If the last entry in the example had been mistyped as COLUMN 25, it would have overwritten the top digit of the monetary value on its right!)

### 6.1.3 More Severe (E- and S-level) Messages

**IGYPA3107-S “UNSTRING INTO” identifier <name (usage)> was invalid ...**

**Meaning**
Depending on which COBOL compiler you are using, one of these messages will appear when your program has a CONTROL data item which is not USAGE DISPLAY. The VS COBOL II error can occur only when the option NOXCAL is in effect. If the PRTX option is in effect, this message will have been placed on the affected CONTROL clause.

**Remedy**
This condition is corrected by REDEFIning your CONTROL data item as PIC X(...) and using the data-name of the redefinition instead. An equivalent method is to make your CONTROL item a group-level item:

```
05 ACCOUNT-NO-X.
  07 ACCOUNT-NO        PIC 9(5) COMP-3.
... CONTROL IS ACCOUNT-NO-X...
```

With VS COBOL II, you may avoid the problem by specifying the (default) option XCAL.

**IGYPS2052-S A “RECORDING MODE” of F was specified for file ...**

**Meaning**
This compiler error message appears as a result of the precompiler message RW-008-I (see 6.1.1 above).

**Remedy**
As under RW-008-I (see 6.1.1 above).

**RW-149-E Report assigned to more than one FD: file handler DUPL required.**

**Meaning**
Defining the same report-name in the REPORT clause of more than one FD is allowed in new Report Writer, as in OS/VS COBOL and DOS/VS COBOL, but for quite different reasons. OS/VS COBOL allows a report to be written to up to two files at the same time, but new Report Writer follows the Codasyl standard by requiring you to select only a single file to be written to at INITIATE time. Thus the following is allowed by both:
but new Report Writer requires you to code UPON FILE-A or UPON FILE-B after INITIATE ANNUAL-SUMMARY.

Remedy

You can still write to two files simultaneously if you need to but this must be achieved using a file handler that performs a separate WRITE to each file. See 5.1 Multiple Reports. The file handler is supplied with all versions of the run time library. You should now add a MODE DUPL clause to your SELECT statement as follows:

```
FILE-CONTROL.
  SELECT FILE-A ASSIGN TO DDname MODE DUPL.
FD FILE-A ...
  REPORT IS ANNUAL-SUMMARY.
```

Meaning

OS/VS COBOL and DOS/VS COBOL allow entries below report group level to have any level-numbers from 02 to 49. The precompiler performs the same strict checks on level-numbers that you would expect in any other DATA DIVISION section. In the following case, for example:

```
  01  TYPE PH.
    07  LINE 2.
    03  COLUMN 1   PIC Z(5)9   SOURCE AMOUNT-1.
    06  COLUMN 11  PIC Z(5)9   SOURCE AMOUNT-2.
```

you will see the following diagnostic messages:

RW-020-S Group item has elementary clauses: ignored.

RW-069-W COLUMN should be subordinate to LINE: LINE + 0 assumed.

It is essential to remove the error condition by renumbering the levels so that:

- LINE clauses are subordinate to (01-level) report groups,
- COLUMN clauses are subordinate to LINES.

The above example, when corrected, should appear as:

```
  01  TYPE PH.
    03  LINE 2.
    05  COLUMN 1   PIC Z(5)9   SOURCE AMOUNT-1.
    05  COLUMN 11  PIC Z(5)9   SOURCE AMOUNT-2.
```

Meaning

Your FD has more than one RD with "CODEs" of unequal size and the corresponding FD is followed by an 01-level record description. (See RW-200-I above). The precompiler implements the CODE clause in this case through a file handler that does not permit access to a separate file record.
A record description is not required after an FD that has a REPORT(S) clause, unless your program does a WRITE independently to the same report file. If your program does not do a WRITE to the same file, the entire record description is redundant and can be removed. If your program does do a WRITE to the same file, you should convert the WRITE to a GENERATE, if necessary by defining a new RD for the same file containing no PAGE clause and a simple DETAIL group for each special record format written to the file.

It is possible for your program to cause other severe diagnostics to appear. Some may be through the unexpected tolerance of the previous compiler of some erroneous construct. It should be clear from the message what you need to do to correct the error. The descriptions of the messages in Appendix E will help you, and you may also refer to the main sections describing the clauses.

### 6.2 Other Considerations

The following situation does not result in a diagnostic message and you should be aware of it, so that you will recognize the circumstances if it occurs in any program.

#### 6.2.1 Long Control Fields Truncated

**Meaning**  
If NOXCAL is the option in use, there is a limit to the length of the saved control fields. This limit, which is 80 bytes as supplied, may be set to any value up to 256 bytes using the CTRLEN option. If your program contains control items longer than this limit, the effect will be an obvious truncation in the contents of the control fields whenever they are printed in your report. When the XCAL option is used, there is no such restriction.

**Remedy**  
To correct this problem, if VS COBOL II is in use, alter your choice of option from NOXCAL to XCAL. Or, for either compiler, set the CTRLEN option to a size that is likely to be larger than any control used in any candidate program. If you are in doubt, a maximum value of 256 will not be harmful, albeit a little inefficient.

### 6.3 Physical Comparison of Report Writer Output

Once you have corrected any syntactic discrepancies in your original Report Writer code, your new Report Writer code should now produce a layout that is visibly identical to the output you obtained under the previous compiler’s built-in Report Writer.

There is however one known difference that may be apparent at the physical level, for example if you use a utility program to compare the print files produced by the two Report Writer systems: the OS/VS COBOL and DOS/VS COBOL Report Writer prints a blank line at the top of each page, even if there is data on LINE 1, whereas new Report Writer, like basic COBOL, implements the latter by a single WRITE at top-of-page.
If you require the print records to be identical in physical arrangement to that produced by your original programs (for example if you use special de-spooling software that reads your print file and expects the first line of data on each page to be blank), you may achieve this by adding the clause `MODE IS PRNT` to the SELECT statement of your report file, since the PRNT file handler is designed to emulate the previous compilers' Report Writer output routine (see 5.3.2 Supplied File Handlers).

### 6.4 Unreachable Code

The precompiler generates statements to perform certain extra checks that may prove to be unnecessary in some programs and may therefore show up as code that "can never be executed" with the OPTIMIZE option of the IBM COBOL compiler and be deleted. There are sound reasons for all these cases and they do not indicate any fault in the code generation. For example, the precompiler will generate a "branch around" before the block of PERFORMed generated procedures in case there should be a "fall-through" from the preceding paragraph. Naturally, the precompiler does not examine the structure of the rest of the program to ascertain whether such a fall-through is actually possible, since this would mean duplicating a task for which the compiler itself is best suited, at the cost of a prohibitive overhead. Other extra checks which may prove unnecessary are:

- SIZE ERROR checks during the accumulation of SUM fields.
- Checks on whether a report has been INITIATEd.
Appendices
Appendix A

List of Post-1968 Extensions

This section lists the extensions to the ANSI 1968 COBOL report writer standard that are included in COBOL Report Writer. The extensions are marked as follows:

- **ANS-68** features "held over" from the ANS-68 standard;
- **IBM** IBM’s extensions to its ANS-68 implementation in OS/VS and DOS/VS COBOL;
- **ANS-74** changes introduced in the ANSI 1974 standard;
- **ANS-85** changes introduced in the ANSI 1985 standard;
- **Codasyl** Codasyl extensions beyond the ANS-85 standard - potential features in a future ANS COBOL standard.

Unmarked extensions are those introduced by SPC Systems. Not all the changes found in ANS-74 are listed here because the changes that represent restrictions to ANS-68 have not been implemented. COBOL Report Writer is thus an optimal merging or best of all worlds from the three standards.

⇒ Items marked in this way are recent additions, new to this release.

1. **FILE-CONTROL and FILE SECTION**

   - **MODE** clause to enable processing by an Independent Report File Handler.
   - **PAGE BUFFER** clause to allow each page to be held in memory in order to set up an irregular format.
     ⇒ **RANDOM PAGE** clause to allow the page’s current line and column to be repositioned like a "cursor".
   - **DUPLICATED** clause to reduce coding when program has more than one report with a similar layout.
   - **(ANS-68)** Report file **FD** may be followed by a record description and may be written to independently.
   - **(ANS-85)** **FD** for report file may have a **GLOBAL** clause and/or an **EXTERNAL** clause.
   - **FIRSTPAGE NO ADVANCING** clause, preventing initial page advance.
     ⇒ **STYLE** clause to facilitate special effects in output device.
     ⇒ **REPORTS ARE ALL** option to assign all reports to a single file.
     ⇒ **(IBM)** A report may be written to up to two files simultaneously.

2. **PAGE LIMIT Clause**

   - **PAGE LIMIT** phrases become clauses in their own right and **PAGE LIMIT** may be last.
   - **DETAIL** in **FIRST/ LAST** may be abbreviated as **DE**
• **FIRST BODY GROUP** alternative spelling for FIRST DETAIL.

• **LAST DETAIL / DETAIL OR CONTROL HEADING** alternative spellings for LAST DETAIL.

• **LAST CF / CONTROL FOOTING / BODY GROUP** alternative spellings for FOOTING.

• **FIRST DETAIL** defaults to line following PAGE HEADING.

• **LAST DETAIL** defaults to line preceding PAGE FOOTING.

• **+/PLUS** form of FOOTING.

• **FOOTING** defaults to line before PAGE FOOTING group, or to PAGE LIMIT.

• Identifier operand of **LAST DETAIL**.

• (ANS-74) Word **LINE** or **LINES** not required.

3. Rest of RD

• **REPORT** as an alternative word for **FINAL**

• **LINE LIMIT** clause.

• **OVERFLOW** clause to check arithmetic expressions for size error.

• **SUM OVERFLOW** clause to check totals for size error.

• **ALLOW (NO) SOURCE SUM CORR** clause to enable/disable SOURCE SUM correlation.

• (IBM) CONTROL operand **FINAL** is assumed if not declared.

• (IBM) Optional **WITH** before **CODE**.

• (ANS-68) **SUM** ming of non-REPORT SECTION item takes place when DETAIL generated with same item as a SOURCE (SOURCE SUM correlation).

• (ANS-68) Control data-names may be **subscripted**.

• (ANS-68) Control fields may be referenced in any group and, at CONTROL FOOTING time, **pre-break** values are supplied.

• (ANS-74) **Literal** form of **CODE** clause.

• (Codasyl) Optional word **IS** with **CODE**.

• (Codasyl) **CODE** may be of any length.

• (Codasyl) **Identifier** form of **CODE** clause.

• (ANS-85) **GLOBAL** clause, enabling the report, or its groups, to be accessed from within a contained program.

⇒ **STYLE** clause to facilitate special effects in output device.

4. Report Groups (General)

• **PLUS** may be written as + whenever used.

• Report Groups may have any number of group or elementary levels.

• (ANS-68) Report group may consist of elementary 01-level entry only.

• (ANS-74) The default qualifier for **PAGE-COUNTER** and **LINE-COUNTER** in the REPORT SECTION is the current report.
• The default qualifier for \texttt{PAGE-COUNTER} and \texttt{LINE-COUNTER} in a \texttt{USE BEFORE REPORTING} section is the report containing the group referred to.

5. \textbf{TYPE Clause}

• Word \texttt{TYPE} optional.
• \texttt{TYPE DETAIL} assumed if no \texttt{TYPE} clause.
• Control-name need not follow \texttt{CH} if only one control present, or \texttt{CF}.
• \texttt{OR PAGE} option of \texttt{CONTROL HEADING} for outputting group after page advance even if no control break.

• \texttt{(Codasyl} ("ON")) Optional word \texttt{FOR} or \texttt{ON} after \texttt{CH} and \texttt{CF}.
  \begin{itemize}
  \item \texttt{CF} may be used for more than one level.
  \item \texttt{CF FOR ALL} meaning \texttt{CF} for all levels of control.
  \end{itemize}

6. \textbf{LINE Clause}

• \texttt{LINE} alone means \texttt{LINE + 1}.
• \texttt{Absolute} \texttt{LINE} may follow \texttt{relative} \texttt{LINE} provided first \texttt{LINE} is absolute.
• \texttt{NEXTPAGE} phrase allowed on \texttt{LINEs} other than first in RH and RF.
• \texttt{LINE} may have no subordinate \texttt{COLUMNS}, thus producing blank line.
• Multiple form to allow several \texttt{LINEs} to be defined in one entry.
• \texttt{(ANS-68)} \texttt{LINE} may be coded as though subordinate to another \texttt{LINE} (although a Warning is issued).
• \texttt{(ANS-68)} Relative \texttt{LINE} allowed at start of \texttt{PAGE FOOTING}.
• \texttt{(ANS-74)} \texttt{LINE} integer \texttt{ON NEXTPAGE} option.
• \texttt{(ANS-74)} \texttt{LINE NEXTPAGE} not allowed in types \texttt{PH}, or \texttt{PF}. (Legacy compilers do not diagnose this but the code fails in consequence.)
  \begin{itemize}
  \item \texttt{(IBM)} Same \texttt{absolute} \texttt{LINE} number may be repeated with each \texttt{COLUMN}.
  \end{itemize}

7. \textbf{COLUMN Clause}

• \texttt{COLUMN} may be shortened to \texttt{COL}.
• \texttt{COLUMN} alone means \texttt{COLUMN + 1}.
• \texttt{COLUMN + 1} assumed by default for any elementary item beneath \texttt{LINE} level with no name, if \texttt{NOOSVS} in effect.
• \texttt{RIGHT}, \texttt{CENTER}, and \texttt{LEFT} phrases.
• Multiple form to allow several \texttt{ITEMs} to be defined in one entry.
• \texttt{(ANS-68)} \texttt{COLUMN} may be coded at same level as preceding \texttt{LINE} (although a Warning is issued).
• \texttt{(Codasyl)} Relative form (+ or PLUS).
8. **SOURCE Clause**
   - SOURCE keyword **optional**.
   - Arithmetic expression form.
   - Multiple form to allow several items to be defined in one entry.
   - *(ANS-68)* CURRENT-DATE and TIME-OF-DAY may be used as identifiers.
   - *(ANS-85)* identifier may be reference modified.

9. **VALUE Clause**
   - VALUE keyword **optional**.
   - PICTURE clause optional with non-numeric literals.
   - Multiple form to allow several items to be defined in one entry.

10. **SUM Clause**
    - Optional word **OF** after SUM.
    - Cross-foot and roll-forward SUM of **SOURCE** and **VALUE** entries allowed.
    - **Totaling** of values in printed **tables** along any axis.
    - SUM allowed in **any** **TYPE** of group.
    - SUM may be used as **term in expression**.
    - Arithmetic expression allowed as operand.
    - Multiple form to total another multiple entry.
    - *(ANS-68)** not in **OS/VS or DOS/VS COBOL** UPON may refer to DETAIL in another report.
    - *(ANS-68)** allowed but not implemented by **OS/VS or DOS/VS COBOL** SUM may refer to REPORT SECTION data-name in a **different report**.
    - *(ANS-74)** More than one **SUM** clause may be coded in the same entry.
    - *(ANS-74)** If **SOURCE** SUM correlation **not** in effect, SUM adds all operands on each **GENERATE**
      ⇒ *(IBM)* Data-name of SUM entry may be re-used in different reports and is implicitly qualified by report-name.

11. **PICTURE Clause**
    - "<" (left-shift) symbol for variable-length fields.
      ⇒ ">" symbol as optional terminator for variable-length part.
      ⇒ General insertion characters using quotes in picture-string. *(This was long since considered by Codasyl and discarded.)*
      ⇒ *(IBM)* PICTURE symbol **A** may be used even with non-alphabetic literal.
12. **NEXT GROUP Clause**

- **NEXT BODY GROUP / NEXT DE OR CH GROUP / NEXT DETAIL OR CONTROL HEADING GROUP** as alternative spellings.
- **(Codasyl)** Optional word **ON** before NEXT PAGE.

13. **New Clauses**

- **PRESENT / ABSENT [J UST] AFTER PAGE / control / PAGE OR control** as more general alternative to GROUP INDICATE.
- Multiple-choice **PRESENT WHEN** entry to select one from a set of SOURCE / VALUE terms.
  ⇒ Special condition **CONTROL IS control-id** for use with PRESENT WHEN and multiple CF group.
- **OCCURS** for repetition with **DEPENDING ON**... and **STEP** phrases.
- **VARYING** clause to enable data-names to vary over a range of values during processing of a repeating item.
- **REPEATED** clause for side-by-side presentation of body groups.
- **GROUP LIMIT** clause to give lower limit to body groups.
- **ROUNDED** phrase for SUM / SOURCE entries.
- **FUNCTION** clause for invocation of built-in or user-written routine for special-format displays.
  ⇒ New **FUNCTIONs**: **CTIME, MONTH, MOVE, YDATE, RYDATE, STATE, STATEF, ZIP**.
- **COUNT** clause, similar to SUM but adding 1 per occurrence.
- **(partly Codasyl)** **PRESENT / ABSENT WHEN** clause to select / deselect any items, lines or groups.
  ⇒ **MULTIPLE PAGE** clause for spreading a group over several pages.
  ⇒ **WRAP** clause for automatic continuation on a new line.
  ⇒ **STYLE** clause to facilitate special effects in output device.

14. **PROCEDURE DIVISION**

- **SET PAGE TO HOLD / RELEASE** statement to invoke / release Page Buffer.
- **SET LINE** and **SET COLUMN** statements to move vertically and horizontally within the Page Buffer or Random Page.
- **(IBM)** A Declarative procedure may refer to (for example PERFORM) a non-Declarative procedure.
- Adding to **LINE-COUNTER** creates an additional gap of that size.
- A REPORT SECTION data-name may be the target field of a PROCEDURE DIVISION statement.
- **(IBM)** **MOVE 1 TO PRINT-SWITCH**
- **(ANS-74)** **SUPPRESS PRINTING** statement, equivalent to **MOVE 1 TO PRINT-SWITCH**.
15. Other Features

- **(ANS-85)** *USE GLOBAL BEFORE REPORTING* form of directive.
  ⇒ *(Codasyl)* *INITIATE UPON* file-name.

- Independent Report File Handlers, built-in or user-written, for directing report writer output to a special device or to user's own de-spooling software.
  ⇒ Report writer keywords are only **locally reserved**.
- **SIGN**, **BLANK WHEN ZERO**, and **JUSTIFIED** allowed at group level.
  ⇒ New form of **SIGN** clause for user-defined treatment of negatives.
- **(ANS-68)** With summary reporting, any number of DETAIL groups may be present.
- **(ANS-85)** REPORT SECTION and report writer statements may be used in **nested programs**.
- **(ANS-85)** **Lower-case** valid in all report writer formats.
- **(ANS-85)** New features incidentally affecting report writer, such as: reference modification, "\(\geq\)" and "\(<\)\(\leq\)" operators, subscripting to 7 levels, relative subscripting.
- **(ANS-85)** **REPLACE** statement may affect report writer code.
  ⇒ **Hexadecimal** Literals in REPORT SECTION.
- **Symbolic Characters** in REPORT SECTION.
  ⇒ **Double Byte Character Set (USAGE DISPLAY-1)** in REPORT SECTION.
  ⇒ Report writer data-names may be **DBCS**.
  ⇒ **Fips Flagging** capability.
- Ability to choose only **ANS standard report writer subset**.

16. General COBOL Features

A number of additional features provided by the precompiler apply to any part of the COBOL source, rather than just to COBOL Report Writer. They are therefore listed here in some detail:

a. In-line Comments

The two-character combination "\(*>\)" indicates that the rest of the source line is to be treated as a comment, for example:

```
05 WS-EOF      PIC 9.       *> end-of-file indicator
...           MOVE 1 TO WS-EOF    *> set end-of-file indicator
```

b. Wild Cards in COPY

The two-character combination "\(?\)\(?\)" may be coded (within **pseudotext brackets ==...==**) as part of the text to be replaced in a **COPY...REPLACING** or **REPLACE** statement. It causes a successful match with any COBOL word,
or a (non-null) part of a word if coded as such. The "??" pair may also appear in the replacement text, in which case it copies unchanged the fragment of text that was matched with the corresponding "??" in the text being replaced.

This sample replaces a certain parameter in each CALL:

```
REPLACE ==CALL ?? USING W-PARAM-1== BY
==CALL ?? USING W-PARAM-2==.
```

This sample changes all words in the source library member beginning with IN- to begin instead with OUT:

```
COPY MEMBER1 REPLACING ==IN-??== BY ==OUT-??==.
```

17. Precompiler’s Tolerance of Other COBOL Constructs

When using the precompiler in tandem with other precompilers or preprocessors, it is essential to know how the precompiler will react to non-ANS COBOL features in the COBOL source. The precompiler has been designed in general to accept embedded CICS, IMS and DB2 commands and tolerate other alien extensions that might reasonably be expected. In particular, it will accept:

a. The **EXEC ... END-EXEC** construct. The text from EXEC through END-EXEC is copied intact, enabling **CICS** and **database** commands to be embedded in a COBOL Report Writer source.

b. **Unrecognized DATA DIVISION SECTIONs**. Any such SECTION is transcribed intact.

c. **Non-standard characters in the continuation column.** Any character other than “D”, “-”, “*”, “/”, and **space** in column 7 is treated as though it were a “*” (comment) character. Such lines will be copied intact unless they immediately precede a report writer construct such as REPORT SECTION that would be removed by the precompiler.

d. **Unrecognized PROCEDURE DIVISION statements.** Any unrecognized word found in the PROCEDURE DIVISION will be copied intact unless it **immediately** follows one of the report writer commands such as INITIATE, when it is expected to be the (first) operand of the statement.
Appendix B
List of New Reserved Words

The following list shows the new reserved words used in the COBOL Report Writer syntax. Except for SUPPRESS PRINTING which is in the 1974 and 1985 ANS standards, these words are new to all three standards. However, with the exception of DATA-SUB-1/2/3/4, LINE-LIMIT, REPORT-NUMBER, and REPEATED-COUNTER they are **locally reserved only** when the OSVS option is in effect. This means in general that the user may use these names as data-names, file names, etc., provided that, if they are used in the REPORT SECTION, it is only with care to avoid ambiguity. For example, SOURCE IS COLS and SOURCE IS STEP are permitted, but the multiple form SOURCES ARE STEP, COLS would not be correctly parsed. SOURCE IS BUFFER, DUPLICATED would be permitted as these keywords normally appear only in the SELECT clause.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSENT</td>
<td>PRESENT/ABSENT WHEN Clause</td>
</tr>
<tr>
<td>BATCH</td>
<td>MODE Clause</td>
</tr>
<tr>
<td>BODY</td>
<td>FIRST/LAST DETAIL, NEXT GROUP Clause</td>
</tr>
<tr>
<td>BUFFER</td>
<td>PAGE BUFFER Clause</td>
</tr>
<tr>
<td>CENTER, CENTRE</td>
<td>COLUMN Clause</td>
</tr>
<tr>
<td>CODE-VALUE</td>
<td>CODE Clause</td>
</tr>
<tr>
<td>COL</td>
<td>COLUMN Clause</td>
</tr>
<tr>
<td>COLS</td>
<td>COLUMN Clause</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>COLUMN Clause</td>
</tr>
<tr>
<td>DATA-SUB-1/2/3/4</td>
<td>OCCURS Clause</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>PRESENT / ABSENT WHEN Clause</td>
</tr>
<tr>
<td>DEPTH</td>
<td>OCCURS Clause</td>
</tr>
<tr>
<td>DUPLICATED</td>
<td>DUPLICATED Clause</td>
</tr>
<tr>
<td>FUNC</td>
<td>FUNCTION Clause</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>FUNCTION Clause</td>
</tr>
<tr>
<td>HOLD</td>
<td>SET Statement</td>
</tr>
<tr>
<td>LINE-LIMIT</td>
<td>LINE LIMIT Clause</td>
</tr>
<tr>
<td>NEW</td>
<td>PRESENT AFTER Clause</td>
</tr>
<tr>
<td>NONE</td>
<td>SOURCE Clause</td>
</tr>
<tr>
<td>NUMBERS</td>
<td>COLUMN Clause, LINE Clause</td>
</tr>
<tr>
<td>PRESENT</td>
<td>PRESENT AFTER Clause, PRESENT WHEN Clause</td>
</tr>
<tr>
<td>PRINTING</td>
<td>SUPPRESS PRINTING Statement</td>
</tr>
<tr>
<td>REPEATED</td>
<td>REPEATED Clause</td>
</tr>
<tr>
<td>REPEATED-COUNTER</td>
<td>REPEATED Clause</td>
</tr>
<tr>
<td>REPLACE</td>
<td>OVERFLOW Clause</td>
</tr>
<tr>
<td>REPORT-NUMBER</td>
<td>DUPLICATED Clause</td>
</tr>
<tr>
<td>SOURCES</td>
<td>SOURCE Clause</td>
</tr>
<tr>
<td>STEP</td>
<td>OCCURS Clause</td>
</tr>
<tr>
<td>SUPPRESS</td>
<td>SUPPRESS PRINTING Statement</td>
</tr>
<tr>
<td>UNLESS</td>
<td>PRESENT WHEN Clause</td>
</tr>
<tr>
<td>WIDTH</td>
<td>OCCURS Clause and REPEATED Clause</td>
</tr>
<tr>
<td>WRAP</td>
<td>WRAP Clause</td>
</tr>
</tbody>
</table>

Words beginning with the prefix **R- -** (note the double hyphen) must also be avoided because the precompiler uses this prefix for its own generated data and procedure names.
Appendix C

Summary of Formats

a. SELECT...ASSIGN clauses

```
Format

►►SELECT    OPTIONAL  file-name►

ASSIGN TO assignment-name RESERVE integer

MODE IS BATCH mnemonic-name

DUPLECTED integer TIMES

WITH PAGE BUFFER

WITH RANDOM PAGE

TYPE IS DEFERRED

IS NONE device-name DEFERRED

FIRST PAGE NO ADVANCING WITH

other standard clauses
```

b. FILE SECTION entries

```
Format

FD file-name IS EXTERNAL IS GLOBAL ►

REPORT IS report-name REPORTS ARE ALL

(continued over)
```
**c. REPORT SECTION entries**

Format

```
REPORT SECTION. report-description-entry
```

where `report-description` is defined as:

Format

```
RD report-name IS GLOBAL LINE LIMIT IS integer IDENTIFIER
HEADING IS integer FIRST DETAIL IS integer
LAST DETAIL OR CONTROL HEADING IS integer IDENTIFIER
LAST CONTROL FOOTING OR CONTROL FOOTING IS integer PLUS integer
PAGE LIMIT IS LINE LIMITS ARE LINES
```
d. Report-Group-Entry

Format (continued)

*CONTROL IS* REPORT FINAL

*CODE IS* literal ALLOW SOURCE SUM CORR

*STYLE IS* style-name

OVERFLOW PROCEDURE IS OMITTED STANDARD REPLACE BY literal STOP literal

SUM OVERFLOW PROCEDURE IS OMITTED

STOP literal

report-group-entry

Format

*level-number* - *data-name*

*TYPE IS* REPORT HEADING RH

PAGE HEADING PH

CONTROL HEADING CH

ON control-id FOR OR PAGE

DETAIL DE

CONTROL FOOTING CF

ON control-id FOR ALL

PAGE FOOTING PF

REPORT FOOTING RP

(Continued over)
Format (continued)

- **FUNCTION** IS function-name

  ( parameter )

- **SUM OF** report-section-data-name

  identifier

  expression

  **UPON** group-name

- **COUNT OF** report-section-dn

  **RESET ON** control-id

  **ROUNDED**

- **PRESENT WHEN** condition

  **CONTROL IS** control-id

  control-id OR PAGE

  PAGE OR control-id

  PAGE

  control-id

- **PRESENT JUST AFTER** NEW PAGE

  GROUP INDICATE

  multiple-choice-form - see below

- **SIGN IS**

  **TRAILING**

  **SEPARATE CHARACTER**

  **LEADING** literal

  **TRAILING** literal

- **BLANK WHEN ZERO**

  **DISPLAY**

  **JUSTIFIED**

  **RIGHT**

  **JUST**

  **DISPLAY-1**

- **OCCURS** integer TO integer TIMES

  **DEPENDING ON** identifier

  expression

  **STEP**

  **WIDTH**

  **DEPTH**

  **LINE**

  **COLUMNS**

  **COLS**

(continued over)
### Format (continued)

```
VARYING data-name FROM expression-1 BY expression-2
```

```
MULTIPLE PAGE
```

```
NO
```

```
WITH NO WRAP WRAP
```

```
AFTER COLUMN integer identifier
```

```
TO COLUMN integer identifier STEP integer LINES
```

```
STYLE IS style-name WHEN condition
```

#### e. Multiple-Choice Form

```
SOURCE IS expression
```

```
SOURCE IS ARE
```

```
VALUE IS literal
```

```
FUNCTION IS function-name
```

```
FUNCTION VALUES ARE
```

```
FUNCTION IS function-name ( parameter )
```

#### f. Additional PROCEDURE DIVISION statements

```
USE GLOBAL
```

```
BEFORE REPORTING report-group-name.
```

Appendix D

Glossary

Absolute Positioning

This describes a LINE or COLUMN clause with an integer operand not preceded by + or PLUS. Absolute positioning places the line or field at a fixed vertical or horizontal position relative to the edge of the page.

Axes of Summing

Four axes are possible: group of LINES, LINE, group of COLUMNS, and COLUMN. When a field is the SUM of another REPORT SECTION field that repeats (has an OCCURS, or multiple LINES or COLUMNS), totalling may take place along any or all of these axes, depending on whether or not the SUM also repeats along the axes.

Body Group

A CONTROL HEADING, DETAIL, or CONTROL FOOTING report group. They are so named because they appear in the “body” of the page, that is, between any PAGE HEADING and PAGE FOOTING groups that may also be present.

CODE

A value that is passed to the output routine but does not ultimately appear in the report. It was originally meant for separating several reports written to the same spool file, but may now also be used for any control information for a basic file that is to be processed by special software downstream, or one controlled by an Independent Report File Handler.

Conditional Field

A field that is qualified by a PRESENT/ABSENT WHEN/AFTER clause or an OCCURS ... DEPENDING with minimum zero, and may not therefore always appear in the report.

Control

A field (control field) represented by an identifier (control identifier), defined in a SECTION other than the REPORT SECTION, whose value is tested by report writer on each GENERATE to establish whether there has been a change in value since the previous GENERATE for the report.

Control Break

A change in the value of a control from one GENERATE to the next. Control breaks may occur at one of several levels, depending on how many controls are listed in the CONTROL clause, and their hierarchy, that is, the major-to-minor order in which they are listed, which is also the order of testing.
Control Characters

Characters which do not appear in the output as printed data but instead influence the way the data is presented. They are often referred to as escape sequences since Escape is frequently the first character.

Cross-footing

Summing from a REPORT SECTION field into another field in the same group.

Declarative SECTION

A SECTION, preceded by a USE [GLOBAL] BEFORE REPORTING statement, that will be executed implicitly just before a certain report group is produced.

Dummy Report Group

A report group that is used not for producing output, but for triggering particular features, such as a NEXT GROUP clause or a Declarative SECTION. It usually has no LINE clauses (and hence no COLUMN clauses either).

DUPLICATED file

A file that has a DUPLICATED clause. This defines a number of separate copies of all the report’s control registers and enables several different physical reports to be produced from one Report Description.

Entry

An element of the DATA DIVISION of a source program beginning with a level-number (or a level-indicator like RD, FD, etc.), followed by optional clauses, and ending in a period.

File Handler

Short for Independent Report File Handler (see below).

FINAL control

An object of the CONTROL and TYPE clauses and the RESET phrase that describes the top-level control, which may be used to produce a CONTROL FOOTING that encompasses the whole report, such as for grand totals. It may also produce a major report-encompassing CONTROL HEADING. The alternative (preferred) name is REPORT.

Form Feed

This is used in this publication to denote the physical action when a printer skips forward to the top of a new page. On a line printer this used to be referred to as a "skip to channel 1". A form feed is used in “batch” printing to execute a page advance but an Independent Report File Handler may use a different method.
Function

A built-in or user-written subroutine that is automatically invoked when the programmer uses a FUNCTION clause. Functions may produce any special format defined by the user or supplied as standard.

GLOBAL Report

A report defined with a GLOBAL phrase in the RD. Such a report is available, together with its special registers and sum-counters, to any program nested within the one in which the RD is defined.

GROUP INDICATE

An older term, used in current ANS standards, for the simplest type of PRESENT AFTER clause.

Independent Report File Handler

A supplied or user-written subroutine that intercepts all the output from the report before it is written and handles it in its own way. It is invoked automatically when the user codes a MODE sub-clause in the SELECT...ASSIGN.

Multiple CONTROL FOOTING

A group headed by the clause TYPE CF FOR control-1 control-2... or CF FOR ALL. The group is then used as a CONTROL FOOTING for each (or ALL) of the controls, thus avoiding the need to code a similar report group at several levels.

Multiple Form of Clauses

A LINE, COLUMN, SOURCE, or VALUE clause with several operands. This enables several different LINEs or COLUMN fields to be represented in a single entry. Such a single entry may also be referred to in a single SUM clause with a great saving in program code.

Multiple-choice Entry

A series of SOURCE, VALUE, or FUNCTION clauses, each followed by a [PRESENT] WHEN / AFTER clause, all written in the same elementary entry. The first choice that is "present" becomes the effective value. WHEN OTHER may be used for the "catchall".

Non-REPORT SECTION SUM

A SUM clause referring to an identifier that is not defined in the REPORT SECTION. Adding takes place according to SOURCE SUM correlation or on execution of each GENERATE.
Page Advance

This is an automatic operation that takes place when a body group that is about to be printed cannot fit entirely on the current page. It outputs a PAGE FOOTING group (if defined), increments PAGE-COUNTER, advances to the top of a new page, and outputs a PAGE HEADING group (if defined).

Page Buffer

An area defined when a file is defined with a MODE clause and a WITH PAGE BUFFER clause. The Page Buffer enables report data to be displayed on the page in random order, using the SET PAGE STATUS, SET LINE, and SET COLUMN statements.

Page-fit Test

A test performed automatically to ensure that the whole of a body group can be fitted on the current page. If not, a page advance is executed and the whole of the body group appears on the next page.

Precompiler

The translation phase of the COBOL Report Writer software that converts report writer clauses and statements into procedural ("vanilla") COBOL before they are compiled.

Relative Positioning

This applies to a LINE or COLUMN clause with a + or PLUS before the integer. (LINE and COLUMN with no operand imply +1 and are therefore also relative.) Relative positioning places the line or field at a distance relative to the preceding line or field.

Report

A report is any human-readable set of data lines produced by any program.

Report Description

The RD entry for a report, plus all the Report Group Descriptions that follow the RD.

Report Description Entry

The full name for the RD entry (not including the report groups that follow it).

Report File

A file defined by means of an FD entry that has a REPORT clause. Any of a wide range of physical realizations of report output that may be produced using report writer.
Report Group
A contiguous set of lines produced in one operation. Also used, informally, to mean a Report Group Description.

Report Group Description
The set of entries, beginning with an 01-level entry, that describe a report group.

REPORT SECTION SUM
A SUM clause referring to a data-name that is defined in the REPORT SECTION. The adding takes place by rolling forward or cross-footing.

Rolling Forward
Summing from a REPORT SECTION field into REPORT SECTION field in a different group, such as from a lower-level CF group into a higher-level CF. (Distinguished from Cross-Footing.)

SOURCE SUM Correlation
An option, for compatibility with ANS-68 report writer, whereby for every non-REPORT SECTION SUM operand a check is made to see in which DETAIL groups, if any, the same item is a SOURCE. Adding then takes place only for those operands whose referencing DETAIL group was GENERATED. (In the ANS-74 and ANS-85 standards, by contrast, a GENERATE causes all non-REPORT SECTION SUM operands to be accumulated into the totals that reference them.)

Special Register
A location such as PAGE-COUNTER, LINE-COUNTER, and CODE-VALUE that is defined automatically by report writer, rather than by the programmer, and may be accessed, usually under special conditions, to control the production of the report.

Style
A special property given to a file, report, line or elementary field that causes it to make use of a special effect available from the output device, such as UNDERLINE, COURIER, or LANDSCAPE. STYLEs may be implemented in a number of different ways, all of them completely transparent to the programmer.

Subtotaling
An ANS report writer term for SUMming of a non-REPORT SECTION item.

Sum Counter
The ANS report writer term for total field, not used in this publication since "COUNT" now has a special significance as a clause.
Summary Reporting

The action that takes place when a GENERATE report-name statement is executed. CONTROL HEADING or CONTROL FOOTING groups are the only body groups that can be presented.

Total Field

An internal register set up implicitly for each entry that has a SUM or COUNT clause. The total field is incremented by the value indicated in the SUM operand until it is output, whereupon it is stored in the report line and reset to zero. Its PICTURE is similar to the explicit PICTURE of the same entry, but without any editing characters.

TYPE

This term has two meanings in report writer. The TYPE clause at the 01-level states how the report group is used in the report, for example TYPE PAGE HEADING. In the SELECT...ASSIGN clause, the TYPE clause indicates the physical device to which the output is to be sent.

Unprintable Item

An elementary item that has no COLUMN clause. It does not appear in the report line but may be totalled like any other item.

Variable-length Field

A report item that has either a "<" PICTURE symbol or a multiple-choice VALUE with no PICTURE and different-length "literals".

Variable-position Field

A report item whose horizontal position may be different from one appearance to the next because it contains a relative COLUMN clause and follows either a conditional or variable-length field, or another variable-position field.
Appendix E

Precompiler Messages

The following messages may be issued during precompilation under various circumstances. Most are self-explanatory, but an additional explanation is given below for each of them. The severity level of each message is also shown. The meanings of these are as follows:

**Severe (S)** Implies a serious violation of the rules of syntax or usage, such that the object program is not reliable - perhaps even incomplete - and should not be run.

**Error (E)** Implies a violation of the rules of syntax or usage, but such that the resultant program will be executable, although the results will not necessarily be those expected.

**Warning (W)** Implies a less severe violation of the rules, or a situation where a change to the code is preferable, although the program will execute as intended.

**Informational (I)** Is used for a confirmatory or informational message and does not imply any violation on behalf of the user.

Return codes are issued by the precompiler to indicate the most severe level of error. They are listed in *Installation and Operation*.

**Messages**

<table>
<thead>
<tr>
<th>Ident.-Sev.</th>
<th>Message and Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW-001-I</td>
<td><strong>No Report Writer data entries were found in this program.</strong></td>
</tr>
<tr>
<td></td>
<td>The precompiler found no REPORT SECTION. This is not an error, as the program need not contain report writer statements.</td>
</tr>
<tr>
<td>RW-002-S</td>
<td>Clause xx not allowed in this context: ignored.</td>
</tr>
<tr>
<td></td>
<td>The given clause should not be coded in this type of entry.</td>
</tr>
<tr>
<td>RW-003-S</td>
<td><strong>Unrecognized item xx: discarded.</strong></td>
</tr>
<tr>
<td></td>
<td>The given word was found when a new clause keyword was expected.</td>
</tr>
<tr>
<td>RW-004-S</td>
<td><strong>Report Writer statements in FILE-CONTROL/FILE SECTION but no REPORT SECTION.</strong></td>
</tr>
<tr>
<td></td>
<td>The program contains a report writer clause in a SELECT...ASSIGN or FD entry but there is no REPORT SECTION.</td>
</tr>
<tr>
<td>RW-007-S</td>
<td><strong>No END DECLARATIVES found.</strong></td>
</tr>
<tr>
<td></td>
<td>While processing the DECLARATIVES portion, the precompiler encountered the end of the source before END DECLARATIVES.</td>
</tr>
</tbody>
</table>
FD has record definition but no RECORD CONTAINS: compiler may assume variable-length.

If there is an 01-level entry following the FD for a report file, the record description generated by the precompiler may disagree with it in length, causing the compiler to assume that the file is RECORDING MODE V. If the 01-level entry is not used in the program, it should be removed.

Word xx expected here: assumed.

The named word is compulsory for this clause.

No period after REPORT SECTION: assumed.

REPORT SECTION should be followed by a period ("." ) character.

RD absent or not in A-margin: assumed.

This fault may be the consequence of a fault in the coding of the REPORT SECTION header.

Clause xx not permitted in RD statement: ignored.

The precompiler is still scanning the RD statement but has found a clause that cannot be used there. It is likely that a period has been omitted.

Clause/phrase xx not allowed in REPORT SECTION: discarded.

The named keyword is a recognized COBOL keyword but cannot be used in the REPORT SECTION.

No report-name follows RD.

A period or a keyword follows immediately after RD. Report writer constructs a name in order to continue scanning.

This report-name has already been defined.

Each report-name can follow only one RD.

REPORT SECTION absent/misspelt: assumed.

An RD entry has been found without a correct REPORT SECTION header preceding it.

Report xx in FD has no REPORT SECTION entry.

A report-name has been declared in a REPORT clause of an FD but there is no RD entry for the report-name. This situation is allowed by OS/VS and DOS/VS COBOL. The superfluous report-name should be removed, together with the clause, if it is its only operand.

Group item has elementary clauses: ignored.

This message appears as a result of an illogical sequence of level-numbers. For example, you may have coded:

```
03 LINE 1 COL 20 ...  
05 COL 30 ...  
```
The first COLUMN clause is at a group level, but COLUMN must always be at the elementary level. This example should be re-coded:

```
03  LINE 1.
05  COL 20  ...
05  COL 30  ...
```

See also Part 6.

**RW-021-E**  
Clause xx empty: ignored.  
This may be the result of using a reserved word as a data item, for example, CONTROLS ARE PAGE.

**RW-022-S**  
Clause xx cannot be repeated in entry: previous occurrence discarded.  
This normally happens as the result of a missing period.

**RW-023-S**  
Invalid CODE Clause: ignored.  
The CODE clause is incorrect.

**RW-024-S**  
CODE too long for record: ignored.  
The CODE value has too many characters and, taking into account the size of the longest print line (and possible carriage control character), would cause the record length implied by the RECORD CONTAINS clause to be exceeded.

**RW-026-S**  
FINAL/REPORT control not highest: ignored.  
The keyword FINAL or REPORT, if included in the CONTROLS clause, must be the first (or only) operand.

**RW-027-S**  
Invalid control identifier.  
This implies that an identifier used as a CONTROL is improperly formed.

**RW-028-E**  
No integer after PAGE LIMIT: assuming 60.  
The integer was probably mistyped.

**RW-029-S**  
PAGE LIMIT not in range 1 to 9999: assuming 60.  
The integer of PAGE LIMIT is not feasible.

**RW-030-I**  
PAGE LIMIT will never be reached.  
This message is issued when there is no PAGE FOOTING and the [LAST CONTROL] FOOTING value is less than the PAGE LIMIT, or when there is a PAGE FOOTING but it does not reach the PAGE LIMIT.

**RW-031-W**  
PAGE LIMIT increased to value of LAST DETAIL or FOOTING.  
The PAGE LIMIT must not be less than the value of LAST DETAIL or [LAST CONTROL] FOOTING. It has been adjusted up to the higher value.
Length xx assumed for CODE identifier.

If the identifier form of the CODE clause is used, the number of characters to be assigned to the CODE is calculated by subtracting the LINE LIMIT (and the size of any carriage control character) from the record size given in the RECORD or BLOCK CONTAINS clause, rather than from the size of the identifier. This message confirms the calculated length.

Values of PAGE sub-clauses invalid or not in sequence.

The integers of the HEADING, FIRST DETAIL, LAST DETAIL, and LAST CONTROL FOOTING sub-clauses and the PAGE LIMIT should be in non-descending order.

Size of CODE differs from other reports going to same file.

This situation is tolerated but, since the CODE literal is placed at the front of each output record, the output may be difficult to interpret.

Standard OS/VS code literal is normally length 1.

However with report writer, CODE values may be of any length.

Invalid level number: assuming xx.

This usually happens as the result of other faults in the code.

LINAGE not allowed with REPORT(S) clause in FD.

The LINAGE and REPORT(S) clauses are mutually exclusive. If you are using report writer, do not code LINAGE or any of its phrases.

Report has no groups.

This is usually the result of the absence of a 01 level-number after the RD entry.

First level no. after RD not 01.

A value of 01 is assumed.

Group-name not unique: name discarded.

Each report group-name should be unique within the RD.

Invalid TYPE: assuming DETAIL.

The keyword after TYPE is not one of the standard names.

Already had TYPE xx: assuming DETAIL.

Two non-body groups of the same TYPE were found in one RD.

Clause xx allowed only at level 01: discarded.

The clauses TYPE, NEXT GROUP, GROUP LIMIT and REPEATED can be coded only at the 01 level.

TYPE xx not allowed without PAGE: assuming DETAIL.

TYPE PH and PF are allowed only if there is a PAGE clause in the RD.
Control in TYPE clause not declared: assuming DETAIL.
TYPE CH or CF should be followed by FINAL/REPORT or the name of one of the identifiers listed in the CONTROL(S) clause of the RD.

Already had CH/CF for this control: assuming DETAIL
Each control identifier can appear in only one CH and/or one CF group (including multiple CFs).

Duplicated control: ignored.
The same control name cannot be used twice in a CONTROL clause. (There is no restriction, however, on using the same field under a redefined name.)

No TYPE: assuming DETAIL.
The TYPE clause may be omitted, in which case DETAIL is assumed.

Absolute LINE not allowed in unpaged report: PLUS form assumed.
Without a legal PAGE clause, every LINE must be relative (LINE + or PLUS).

LINE value should be non-negative integer: ignored.
The integer of the LINE clause is not numeric or is negative.

NEXTPAGE option of LINE in unpaged report: discarded.
LINE [integer] ON NEXT PAGE is not allowed if there is no PAGE clause in the RD.

PLUS not allowed with NEXTPAGE: ignored.
The form LINE IS PLUS/+ integer ON NEXT PAGE is not allowed.

NEXTPAGE allowed only in first line of this group.
The LINE IS [integer] ON NEXT PAGE clause can only appear as the first LINE clause of a group, except in a TYPE RH or RF, or a MULTIPLE PAGE group.

Integer in NEXTPAGE below heading value.
In the LINE IS integer ON NEXT PAGE clause, the integer must not be less than the HEADING integer (default value 1).

Absolute line follows relative line.
This is not allowed if the first LINE of the group is relative, except in a MULTIPLE PAGE group.

Line position precedes previous line.
All absolute LINE numbers must be in strictly ascending order (taking into account possible ABSENT lines), except in a MULTIPLE PAGE group.

LINE clause subordinate to another LINE: ignored.
This condition arises when the OSVS option is not in effect and the LINE clauses in a report group are at different levels, for example:
It may be rewritten as follows:

```cobol
01 ... TYPE DE.
   03 LINE NEXT PAGE.
      05 COLUMN 20 ...
   03 LINE PLUS 1.
      05 COLUMN 20 ...
```

This structure is accepted as written by OS/VS COBOL, DOS/VS COBOL, and by this implementation, provided the OSVS option is in effect, in which case a Warning RW-064 appears.

RW-064-W  **LINE entries nested: previous LINE assumed level xx.**

See the comments under RW-063 above.

RW-066-S  **COLUMN value should be positive integer <999: ignored.**

The integer of the COLUMN clause is not feasible.

RW-069-W  **COLUMN should be subordinate to LINE: LINE +0 assumed.**

This message is issued when a COLUMN clause is found but there is no LINE clause containing it, for example:

```cobol
01 TYPE IS PH.
   03 COLUMN 30 ...
```

The same message appears if the OSVS option is not in effect and the COLUMN clause is coded at the same level as the preceding LINE clause (or higher), for example:

```cobol
03 LINE 1 COLUMN 20 ...
   03 COLUMN 30 ...
```

Here, the second COLUMN clause is not subordinate to any LINE clause. A LINE PLUS ZERO clause is assumed in default. This means that an unnecessary second record will be written so that the item will be printed in the position defined by the preceding LINE. This structure should be re-coded as follows:

```cobol
03 LINE 1.
   05 COLUMN 20 ...
   05 COLUMN 30 ...
```

If the OSVS option is in effect, Warning message RW-070 appears instead and the additional LINE PLUS ZERO is not generated.

RW-070-W  **COLUMN entries following LINE assumed to be subordinate to it.**

See under RW-069 above. See also Part 6.
RW-071-S **NEXTGROUP value should be positive integer: discarded.**
The integer following NEXT GROUP is not numeric or is negative or zero.

RW-072-W **Recurrence of same absolute LINE merged with preceding.**
This message may be result from the following type of construct when the option OSVS is in effect:

```
03 LINE 1 COLUMN 10 ...
03 LINE 1 COLUMN 30 ...
```
The two COLUMN entries should be brought together under a single LINE 1 entry, but OS/VS and DOS/VS COBOL allow this construct, as does the precompiler also.

RW-074-S **NEXTGROUP absolute or NEXTPAGE in unpaged report: PLUS assumed.**
The forms NEXT GROUP IS integer and NEXT GROUP NEXT PAGE are not allowed unless there is a PAGE clause in the RD.

RW-075-S **Invalid PICTURE: PICTURE X substituted.**
The PICTURE clause has an invalid symbol, or an invalid combination of symbols or lacks a compulsory symbol.

RW-076-S **PICTURE too long: truncated to 32 symbols.**
There is a limit of 32 characters on a PICTURE string.

RW-081-S **Invalid literal: clause or phrase ignored.**
A literal is expected here but the item found is not correct.

RW-082-S **Invalid integer: clause or phrase ignored.**
An integer is expected here but the item found is not correct.

RW-083-W **Hexadecimal value defined as printable text rather than by STYLE clause.**
Although hexadecimal values are allowed in the REPORT SECTION, their usual purpose is to put control characters in the print data. A more system-independent method is to use the STYLE clause.

RW-084-S **Invalid identifier or expression: clause or phrase ignored.**
An identifier or expression is expected here but the item found is not correct.

RW-085-I **New reserved word accepted as data-name.**
Clauses such as SOURCE IS STEP or SUM OF FUNC are recognized as correct even though STEP and FUNC are report writer keywords. This allow older code to continue to function without change.
CF group refers to identifier that is not a CONTROL; after-break value may be used.

You have specified an identifier in a SOURCE clause within a CONTROL FOOTING group and the identifier is not a CONTROL or a SUM total field. This message reminds you that only CONTROL items are restored to their before-the-break values during the processing of CONTROL FOOTING groups. If the data item referred to normally changes at control-break time, you will obtain its after-the-break value (unless the item is a redefinition of, or subordinate to, a CONTROL item) and this may not be what you intended. To obtain the before-the-break value, you could define a Declarative section for the CONTROL HEADING. If the data item does not change at control-break time, this message should be ignored.

Control in RESET phrase not declared: discarded.

Except for FINAL or REPORT, the identifier specified in the RESET phrase must be declared in the CONTROL clause.

RESET at lower level than associated SUM clause.

If SUM and RESET are used in a TYPE CF group, the control used must be at the same level or a higher level than that of this CF group.

Elementary item must have data-name or SOURCE etc.

Every elementary item should have either SOURCE, or VALUE, or SUM/COUNT, or FUNCTION clause, or should carry a data-name.

LINE...NEXTPAGE not allowed in this type of group.

It is not allowed in a TYPE PH or RH group.

First LINE number will cause group to be positioned higher than top limit.

The first LINE number of the group is such that the group will start higher (earlier) than its highest permissible position on the page.

LINE clauses in group will cause it to extend beyond bottom limit.

The last LINE number of the group is such that the group will end lower than its lowest permissible position on the page. See also Part 6.

PRESENT should not precede WHEN/UNLESS if used as qualifier for previous clause.

The WHEN phrase of the STYLE clause is not normally preceded by PRESENT. If you have written something like:

```
05 COL 1 VALUE "Payment overdue" STYLE UNDERLINE
       PRESENT WHEN W-DATE < W-TODAY.
```

note that the condition immediately following the STYLE clause is assumed to qualify just the STYLE clause, so this should be re-written as:

```
05 COL 1 STYLE UNDERLINE VALUE "Payment overdue"
       PRESENT WHEN W-DATE < W-TODAY.
```
RW-099-E  **ABSENT not allowed before qualifying condition.**

The STYLE clause cannot be immediately followed by an ABSENT clause and the order of the clauses should therefore be changed.

RW-100-W  **WHEN OTHER allowed only with multiple choice.**

The clause [PRESENT] WHEN OTHER cannot be used in isolation. That is, there must be at least one [PRESENT] WHEN condition clause in the same entry.

RW-106-W  **This RH group will have own page: NEXTGROUP NEXTPAGE assumed.**

If NEXT GROUP NEXT PAGE is not used in an RH group, report writer will try to fit the group on the first page above the PH group. If that cannot be done, this message is issued.

RW-107-W  **This RF group will have own page: NEXTPAGE assumed.**

If LINE...NEXT PAGE is not used in an RF group, report writer will try to fit the RF group on the last page below the PF group. If that cannot be done, this message is issued.

RW-110-I  **Elementary item has no size: will not be output.**

This message occurs when the PICTURE clause is omitted from an elementary item and there is no way of telling its size, such as from a VALUE "literal" as, for example:

| 05 COLUMN 21 | SOURCE IS WS-PAYMENT. |

A PICTURE clause is always required at the elementary level except (a) with a simple VALUE "literal". However, OS/VS and DOS/VS COBOL allow the PICTURE to be omitted and treats the field as unprintable, so this technique was often used in the past to enable an item to be used in a SUM when it was not also being used as a SOURCE.

RW-112-W  **Absolute NEXTGROUP position not beyond last line position.**

The integer of a NEXT GROUP integer clause in a TYPE RH or PF group must be greater than the last line position in the group.

RW-113-W  **NEXTGROUP position out of range.**

Either the current group is a REPORT HEADING and the NEXT GROUP is beyond the FIRST DETAIL, or the current group is a PAGE FOOTING and the NEXT GROUP is beyond the PAGE LIMIT.

RW-114-W  **Absolute NEXTGROUP not in range FIRSTDETAIL to FOOTING.**

The integer of a NEXT GROUP integer clause in a body group must be in that range.

RW-115-S  **NEXTGROUP not allowed in types PH and RF: ignored.**

To provide spacing following a Page Heading, the FIRST DETAIL clause may be used.
**SUM clause has nonnumeric PICTURE: ignored.**
The PICTURE clause was probably mistyped.

**Group-name in UPON phrase is unknown or ambiguous.**
An UPON phrase is followed by a name that is either not a group-name or requires qualification.

**UPON group not type DETAIL**
It is not possible to SUM ... UPON a group that is not TYPE DETAIL. If the field is a SOURCE in the group, you may instead give a data-name to the entry and SUM that data-name.

**Rolling forward of SUM from higher to lower CF group not allowed.**
An attempt is being made to SUM a field in a higher CONTROL FOOTING and print the total in a lower CONTROL FOOTING.

**Report-name qualifier required: IN xx assumed.**
The same group-name is in use in more than one report. The name in the first report is arbitrarily chosen.

**Qualifying report-name invalid.**
The only qualifier accepted following a sum counter, group-name or special register is the report-name itself.

**Name following GENERATE not report or detail group.**
The name following GENERATE is probably misspelt.

**Name following INITIATE/TERMINATE not report: ignored.**
The name following INITIATE or TERMINATE is probably misspelt.

**Cannot do summary reporting without CONTROLS clause.**
This may be issued if GENERATE report-name is coded erroneously instead of GENERATE detail-name.

**SUPPRESS allowed only in DECLARATIVES: ignored.**
A SUPPRESS keyword has been found outside the DECLARATIVES part of the program.

**Invalid operand after USE BEFORE REPORTING: section discarded.**
The directive USE BEFORE REPORTING must be followed by the name of a report group.

**“xx”.. not allowed in Report Writer DECLARATIVES.**
This is issued if an INITIATE, GENERATE, or TERMINATE is found in the DECLARATIVES part of the program.

**Report Writer verb has no operands: ignored.**
The verb INITIATE, GENERATE, or TERMINATE is followed immediately by a period, another verb, or the end of the source.
Appendix E: Precompiler Messages

RW-138-S  **Group already has DECLARATIVES section.**

There can only be one DECLARATIVES section for any given report group.

RW-139-W  **USE AFTER EXCEPTION/ERROR DECLARATIVE not used by file-handler.**

Independent Report File Handlers cannot access the USE AFTER EXCEPTION or USE AFTER ERROR DECLARATIVE section for the file they control. This message warns that the section will have no effect. A FILE STATUS should be created and tested after each operation instead.

RW-142-W  **No INITIATE statement found for this report.**

Check that INITIALIZE was not coded by mistake instead of INITIATE. This message and the next are issued only if there is a GENERATE statement referring to the report.

RW-143-W  **No TERMINATE statement found for this report.**

See notes under RW-142 above.

RW-144-W  **MODE more than 4 characters: excess ignored.**

Report writer generates the name CRFHxxxx for the name of the Independent Report File Handler, so 4 is the limit.

RW-145-E  **MODE not allowed where record descriptions follow FD: discarded.**

A report file that is written using an Independent Report File Handler cannot process any records other than those implicitly defined in the REPORT SECTION. If the (01-level) record description is not referred to (via a WRITE statement) anywhere in the program, it should be omitted. Otherwise, a way should be found to write the record by means of additional code in the REPORT SECTION. The file handler itself may also be enhanced to write special records to the print file.

RW-146-I  **No GENERATE issued for this DETAIL.**

There is no GENERATE statement in the PROCEDURE DIVISION referring to this DETAIL group-name.

RW-148-S  **Invalid UPON file-name.**

The file-name specified in the INITIATE...UPON is not defined in the FILE SECTION or it is not a report file or its REPORT(S) clause does not include the report-name (or one of the report-names) given in the INITIATE.

RW-149-E  **Report assigned to more than one FD: file handler DUPL required.**

OS/VS and DOS/VS COBOL allowed a report-name to be defined in the REPORT(S) clause of up to two FD’s. This property is now emulated using the DUPL file handler. You should code **MODE DUPL** in the SELECT ... ASSIGN statement for both files, or use the precompiler’s **FMODE** option.
Superfluous period: ignored.
This message will appear if an excess period appears between the level-
number and the last clause in the entry, for example:

05 COLUMN 21. PIC Z(5)9. SOURCE IS WS-INCOME.

Period missing: assumed.
This message will appear if no period appears at the end of an entry
before the next level-number or heading. Occasionally, this may be the
result of coding a superfluous numeric literal that is wrongly assumed to
be a new level-number.

SUM will be totalled UPON generation of xx due to SOURCE SUM
correlation.
In an original report writer program that has more than one DETAIL
group, this message should be expected for every SUM operand which is
not the name of another REPORT SECTION item. This message confirms
that SOURCE SUM correlation is in effect (one of the supplied defaults
which is essential when an original OS/VS or DOS/VS COBOL source is
processed). Report writer has searched the SOURCE clauses in all the
DETAIL groups for the same identifier as that specified in the SUM clause
and has found it in DETAIL xx. It will perform the adding when, and only
when, DETAIL xx is GENERATED. This message tells you exactly when
adding of the SUM operand takes place.

SUM will be totalled also UPON generation of xx.
This is used in combination with message RW-161. It indicates that the
SUM operand was found in more than one DETAIL group. Adding will
take place when either the DETAIL named in message RW-161-I or the
DETAIL named in this message is GENERATED.

Item not in REPORT SECTION is accumulated on every GENERATE.
The SUM clause referenced has an identifier from a SECTION other than
REPORT SECTION, there are more than one DETAIL groups and either
there is no SOURCE SUM correlation or the same identifier does not
appear as a SOURCE in any DETAIL of the report. This message reminds
you that adding will take place on the execution of a GENERATE for any
DETAIL in the same report, in accordance with the ANS-85 standard. This
message should not appear for any program written originally for OS/VS
or DOS/VS COBOL, as the previous compiler would not have generated
any adding for the SUM entry in these circumstances and a zero value
should have resulted.

Sum counter may be accumulated without being cleared.
There is a RESET phrase for a total field that is also used for rolling forward
to another SUM. Some values will therefore be rolled forward several
times into the same total, which is therefore suspect.
Appendix E: Precompiler Messages

RW-166-W **UPON phrase not allowed with SUM of Report Section entry.**

It is not permissible to use the UPON phrase if the item added is a REPORT SECTION data-name. REPORT SECTION items are added in whenever their group is produced.

RW-167-S **REPORT SECTION entry referred to by SUM must be numeric.**

Only a numeric entry may be the object of a SUM. However, a COUNT clause may be used to count the occurrences of any entry.

RW-168-S **RESET without SUM or COUNT clause: discarded.**

The RESET phrase has no meaning except in conjunction with a SUM or COUNT clause.

RW-171-S **Invalid SIGN clause: discarded.**

The SIGN clause is malformed.

RW-172-S **BLANK WHEN ZERO not applicable: discarded.**

BLANK WHEN ZERO can only be associated with numeric items.

RW-173-E **Entry contains both DBCS and non-DBCS fields.**

If the entry has a multiple VALUE clause, either all the literals must be DBCS or all must be non-DBCS.

RW-174-E **DISPLAY-1 group must contain only DBCS fields.**

If DISPLAY-1 is coded on a group level, every subordinate entry must be DBCS, that is, it must have a DBCS PICTURE ("G" symbol) or a DBCS literal VALUE, or both.

RW-180-E **CODE not allowed in RD where a record description follows FD.**

The report associated with this file has a CODE clause, but the FD is followed by an 01-level record description. Since all the records written to the file should have the additional "CODE" characters attached, it is not permissible to WRITE records independently of report writer. If the 01-level record description is not referred to, it should be deleted.

RW-181-E **PAGE BUFFER/DUPLICATED not allowed in RD whose FD has a record description.**

The PAGE BUFFER and DUPLICATED clauses imply a special organization of the report file or its records and it is not permissible to WRITE records independently of report writer. If the 01-level record description is not referred to, it should be deleted.

RW-184-W **Some SELECT clauses will be ignored by file handler and may be invalid.**

The MODE clause causes the report to be output using an **Independent Report File Handler**. The phrases **RESERVE integer AREA(S), PADDING CHARACTER, RECORD DELIMITER, and PASSWORD of the SELECT...ASSIGN clause** are not processed by the file handler and are treated as documentary only.
Some FD clauses will be ignored by file handler and may be invalid.

The MODE clause causes the report to be output using an Independent Report File Handler. The FD phrases CODE-SET, RECORD IS VARYING..., BLOCK CONTAINS integer RECORDS, LABEL RECORD(S) IS/ARE data-name, and VALUE OF... are not processed by the file handler and are treated as documentary only.

"xx" does not conform to chosen language level.

The precompiler has been installed to accept only a subset of the available language and the specified item is not in that subset.


This message appears as a result of the FIPS option to flag nonstandard IBM and SPC extensions to the current standard.


This message appears as a result of the FIPS option to flag standard elements above a specified level.


This message appears as a result of the FIPS option to flag all obsolete elements.

MODE PRNT has been assumed for file due to CODE clause.

If a file has more than one report whose RD's do not all have a CODE of equal length, the file can only be processed by an Independent Report File Handler. The file is therefore treated as though MODE PRNT had been coded.

Scanning to next recognizable period or keyword.

This message is issued after a more serious error to document the precompiler's error-recovery processing.

No associated FD: MODE PRNT assumed.

Each RD's report-name must appear in the REPORT clause of an FD as well as in the RD. The precompiler supplies a file by default and, for simplicity, implements it via the PRNT file handler.

REPORT SECTION nest level must not exceed 23.

This is the limit to the number of different levels of nesting allowed using level-numbers in the REPORT SECTION.

Data-name accessible in REPORT SECTION only.

This message is issued when an entry has a data-name that the precompiler cannot retain when generating the intermediate data description. Possible reasons for this are: (a) the LINE is subject to an OCCURS clause and, as lines are generated only once, the dimensions of the data-name would be insufficient; (b) the item has a PICTURE clause with an "<" symbol and this would be incomprehensible to basic COBOL.
PLUS integer allowed only with FOOTING: ignored.
The HEADING, FIRST DETAIL and LAST DETAIL clauses have no PLUS integer form.

OCCURS clause not allowed at level 01.
Perhaps a REPEATED clause was intended. This is used at level 01 to print several body groups across the page.

LINE LIMIT invalid: clause ignored.
The LINE LIMIT clause is malformed.

LINE LIMIT too high for record/CODE.
The LINE LIMIT exceeds the maximum number of bytes available in the print line, taking into account an existing CODE.

CODE identifier form not allowed without LINE LIMIT.
If the identifier form of the CODE clause is used, a LINE LIMIT clause must be present.

Value xx assumed for LINE LIMIT.
This message is always issued if there is no LINE LIMIT clause in an RD. It gives the rightmost value allowed for any COLUMN and enables report writer to ensure that none of your print lines will be truncated because they are wider than one printer's width. This default value is calculated from the RECORD CONTAINS or BLOCK CONTAINS clause in the corresponding FD (allowing for the carriage control character), or, failing that from the installed default (see Installation and Operation). It should be checked to ensure that the value is as expected. (See LINE LIMIT clause.)

SOURCE/ SUM will show same value in each repetition.
An OCCURS clause or multiple LINES or COLUMNS clause is in effect but the same SOURCE or SUM value will be placed in each occurrence because there are no subscripts to be varied, or there are subscripts but no VARYING clause.

More than one unconditional SOURCE etc.: previous discarded.
More than one SOURCE, VALUE clause, etc. has been found and there are no associated [PRESENT] WHEN condition clauses to indicate a multiple-choice entry.

Condition has unpaired left parenthesis or reserved word.
A major keyword, or a period, or the end of the source was found before the expected closing parenthesis (")") while scanning a condition.

Item xx found out of context and ignored.
The given item was not expected in this context.

Illegal OCCURS integer: clause discarded.
An integer of this OCCURS clause was non-numeric.
OCCURS...TO maximum must exceed minimum: TO phrase discarded.
The second integer in an OCCURS...TO clause must be greater than the first integer.

Minimum OCCURS integer negative: clause ignored.
The first integer in an OCCURS...TO clause must be zero or greater than zero.

OCCURS integer must be positive: clause discarded.
The integer in a basic OCCURS clause or the second integer in an OCCURS...TO clause should be greater than zero.

Invalid identifier/expression in DEPENDING ON: phrase discarded.
This may be due to the use of a SUM or COUNT term in the expression.

OCCURS...TO format allowed only with DEPENDING.
A DEPENDING ON... phrase was expected following the OCCURS...TO clause. The first integer is discarded.

Depth of repetition exceeds 4: clause discarded.
Only up to four nested levels of OCCURS or multiple LINES or COLUMNS are allowed.

More than 8 levels of OCCURS or PRESENT etc.
Only up to eight levels of PRESENT WHEN, PRESENT AFTER, or GROUP INDICATE clauses, or repetition are allowed, including up to 4 levels of nested OCCURS, multiple LINES, or multiple COLUMNS.

Invalid FUNCTION: clause ignored.
This is issued if the function name is malformed.

No closing parenthesis in FUNCTION: assumed.
A major keyword, or period, or end of source was found before the closing parenthesis following the parameters to a FUNCTION clause.

Summed REPORT SECTION item must have SOURCE, SUM or VALUE.
A SUM was found of a REPORT SECTION item, but the item has no SOURCE, SUM, or VALUE clause.

Misuse of < symbol in PICTURE: ignored.
The "<" symbol may only be used immediately preceding a PICTURE symbol representing "data" as opposed to an insertion character.

Report has > 1 CONTROL, so control-id must be specified in TYPE CH.
TYPE CH must be followed by one of the identifiers (or REPORT / FINAL) listed in the CONTROL(S) clause.
GROUP INDICATE/PRESENT AFTER not allowed in RH or RF groups.
A TYPE REPORT HEADING or REPORT FOOTING group cannot have a GROUP INDICATE clause or a PRESENT AFTER... clause.

Undeclared control in GROUP INDICATE/PRESENT AFTER: discarded.
All control names must first be declared in a CONTROL clause.

Invalid GROUP INDICATE syntax: default format assumed.
The ON/FOR phrase is incomprehensible, so the simple (no-operand) GROUP INDICATE format is assumed.

PAGE option used in unpaged report: PAGE discarded.
There is no (valid) PAGE clause in the RD and hence none of the options using the keyword PAGE can be used.

REPEATED invalid in dummy or multiple-page group.
A group with a REPEATED clause must have at least one LINE and must not have a MULTIPLE PAGE clause.

Invalid REPEATED syntax: clause ignored.
The REPEATED clause is malformed.

Too many repeats for available LINE LIMIT.
The LINE LIMIT cannot accommodate the number of repetitions given in the TIMES phrase. If an EVERY/WIDTH phrase is given, the spacing given is too wide, or, if not, the desired number of repetitions cannot fit, even at the closest possible spacing.

REPEATED allowed only in body groups: clause ignored.
The REPEATED clause can be used only in a TYPE CH, DE, or CF group.

REPEATED must have EVERY or TIMES: ignored.
Either the TIMES phrase or the EVERY/WIDTH phrase or both must be present in the REPEATED clause.

Repeats of this group will overlap.
Running the program will result in encroachment between successive side-by-side groups with unpredictable results.

Invalid condition: clause ignored.
The condition following a PRESENT/ABSENT clause was syntactically invalid.

Superfluous PRESENT/GROUP INDICATE/ERROR found: discarded.
This fault is usually the result of a missing period.

GROUP INDICATE/PRESENT AFTER cannot be part of multiple choice.
The only conditional clauses allowed in a multiple choice entry must be of the form [PRESENT] WHEN condition.
GROUP IND./PRESENT AFTER in CONTROL group must refer to higher level.

If a PRESENT AFTER control [OR PAGE] clause is written in a CH or CF group, the control referred to must be at a higher level than that of the group.

Item overlaps or is to left of item in same line.

This message is issued if two unconditional horizontal items overlap or if their COLUMN numbers are not in ascending sequence. It will also appear in cases where a conditional item could overlap an unconditional one. The item will be accepted as defined. If it does not overlap any of the column positions of a previous field in the line, there will be no problem at run time. Otherwise, the earlier field will be overlaid, in whole or in part, by the new item.

Column overlap may occur in this line.

This message is issued if some of the items in the report line have PRESENT WHEN clauses without which they would overlap. The precompiler assumes that the programmer has taken action to ensure that the conditions are mutually exclusive.

Final COLUMN position may exceed line limit.

This message is issued if some of the items in the report line have PRESENT WHEN clauses without which they would exceed the size of the line.

Abs. line follows relative with no unconditional abs. lines.

An absolute LINE clause (no +) cannot follow a relative LINE (with +) unless there is an unconditional absolute LINE earlier in the group.

REPEATED absolute group must have at least 1 unconditional LINE.

It is not possible for all the LINE entries in a group with a REPEATED clause to have a PRESENT/ABSENT/GROUP INDICATE clause, unless they are all relative (with PLUS or +).

Nonnumeric SOURCE or non-REPORT SECTION SUM cannot be ROUNDED.

ROUNDED can only be used with (a) a numeric SOURCE or (b) a SUM clause that refers to another REPORT SECTION item.

Final column position in line exceeds limit.

The rightmost column of this line exceeds the value of LINE LIMIT. The line will be truncated.

Line overlap: LINE PLUS 1 assumed.

This message will be issued if two unconditional absolute lines overlap. It will also appear in cases where a conditional absolute line could overlap an unconditional one.
RW-259-I  **Line overlap may occur in this group.**

This message is issued if some of the lines in the group have PRESENT WHEN clauses without which they would overlap. The precompiler assumes that the programmer has taken care to ensure that the conditions are mutually exclusive.

RW-260-I  **This group may extend beyond lower limit.**

This message is issued if some of the lines in the group have PRESENT WHEN clauses, which, if all present, would exceed the maximum size of the group as defined in the PAGE LIMIT sub-clauses or the GROUP LIMIT clause.

RW-261-S  **Circular reference of sum totals: no summing.**

This message is issued if circular combinations of cross-foot SUM's are used, such as:

<table>
<thead>
<tr>
<th>A SUM A</th>
<th>B SUM A</th>
</tr>
</thead>
<tbody>
<tr>
<td>or A SUM B</td>
<td>B SUM A</td>
</tr>
<tr>
<td>or A SUM C</td>
<td>C SUM A</td>
</tr>
</tbody>
</table>

RW-262-E  **Number of occurrences of addend and SUM field differ at level xx.**

This message is issued in the following case: a SUM clause is subject to repetition (OCCURS or multiple LINES / COLUMNS); an item referred to in the SUM clause is a REPORT SECTION item that also has repetition. After matching the axis of each repetition (LINE against LINE, COLUMN against COLUMN, group of LINES against group of LINES and group of COLUMNS against group of COLUMNS), it is found that the numbers of occurrences along the matching axis are not the same.

RW-263-S  **Repetition affecting SUM field has no equivalent for summand.**

This message is issued when a SUM clause has repetition (OCCURS clause or multiple LINES / COLUMNS) and an item referred to in the SUM clause is a REPORT SECTION item but has no repetition along the axis of repetition of the SUM. Briefly: a multiple SUM cannot be manufactured from a single original value.

RW-265-W  **UPON phrase required in running total in type DETAIL.**

This message is issued if a SUM clause is found in a DETAIL group and the item summed is not in the REPORT SECTION. The precompiler cannot ascertain when adding is to take place unless an UPON phrase is included.

RW-266-E  **Rolling forward not allowed between these types.**

It is not permitted to roll forward into a REPORT HEADING group or from a REPORT FOOTING group.

RW-267-S  **No subscripts allowed on SUM of REPORT SECTION entry.**

The SUM identifier must be without subscripts when it refers to an entry in the REPORT SECTION.

RW-270-S  **SUM OVERFLOW clause is invalid: discarded.**

The SUM OVERFLOW clause is malformed.
Leftmost COLUMN position less than one: COLUMN PLUS 1 assumed.

This message is associated with the RIGHT and CENTER forms of the COLUMN clause.

Invalid MODE clause: discarded.

The MODE clause is malformed. (The mode name should not be in quotes or apostrophes.)

Invalid or duplicated phrase after MODE: discarded.

One of the phrases or clauses USING, WITH PAGE BUFFER, or DUPLICATED is either malformed or occurs more than once.

Invalid DUPLICATED integer: phrase discarded.

No valid integer follows the DUPLICATED keyword.

MODE xx has been assumed for file due to clauses in FILE-CONTROL entry.

To implement certain features, such as PAGE BUFFER and DUPLICATED, the independent report file handler protocol is required, rather than direct "WRITEs". If there is not already a MODE clause, report writer handles the file as though a MODE clause had been coded.

SET statement has invalid syntax: discarded.

The SET PAGE/LINE/COLUMN statement is malformed.

SET statement not allowed with report that has no page buffer.

The SET PAGE/LINE/COLUMN statement cannot be used unless the corresponding SELECT...ASSIGN clause has a WITH PAGE BUFFER clause.

Integer of SET statement is out of range.

An attempt is being made to SET the LINE or COLUMN outside the dimensions of the page.

RESET not allowed in multiple CF group.

RESET cannot be used with a SUM or COUNT clause in a multiple CONTROL FOOTING because several levels of totalling are implied.

TYPE CF with no operands assumed to be multiple CF for all controls.

If TYPE CONTROL FOOTING is coded with no control-id operand following, it is taken to mean CONTROL FOOTING FOR ALL.

Mis-use of CONTROL IS control-id condition.

The special condition CONTROL IS control-id can only be used in a multiple TYPE CONTROL FOOTING that has that control-id as one of its operands.
**RW-291-W**  
**UPON phrase refers to current group.**  
The operand used in the UPON phrase should refer to the name of a different group.

**RW-293-W**  
**SUM of item with GROUP INDICATE/ PRESENT AFTER may be incorrect.**  
If the item quoted in the SUM clause is subject to one of these clauses, it cannot be reliably totalled, since report writer will, on principle, not total fields that ordinarily will not be Present.

**RW-294-W**  
**Summed REPORT SECTION name xx is ambiguous: first occurrence used.**  
Either (a) the given data-name, the operand of a SUM clause, is either defined in more than one place within the current report (in which case the names should be made different) or (b) the given data-name is **not** defined in the same report but **is** defined in more than one different report (in which case report-name qualifiers should be used).

**RW-295-I**  
**Maximum number of repetitions will be xx.**  
In the absence of a TIMES phrase, report writer has calculated from the rest of the EVERY/WIDTH phrase of the REPEATED clause and the LINE LIMIT that this many repetitions of the group can be arranged side-by-side.

**RW-296-I**  
**Distance between REPEATED groups assumed to be xx COLUMNS.**  
In the absence of an EVERY/WIDTH phrase, report writer has calculated from the TIMES phrase and the LINE LIMIT that the distance between corresponding repetitions will be that many columns.

**RW-297-W**  
**Absolute LINES subject to REPEATED or WRAP cannot all be conditional.**  
For the REPEATED or WRAP clause to work correctly, report writer must know at precompilation time whether the group will be absolute or relative. If all the absolute LINE clauses have a condition attached, this cannot be done.

**RW-301-W**  
**Multiple LINE or COLUMN should not be blank.**  
An entry containing a LINE clause with more than one operand should be followed by at least one subordinate COLUMN entry. An entry containing a COLUMN clause with more than one operand should also contain a SOURCE, SUM, VALUE, or FUNCTION clause.

**RW-302-S**  
**Number of SOURCE items etc. does not match number of repetitions.**  
With a multiple SOURCE or VALUE, there must be an equal number of repetitions either as an OCCURS or as a multiple LINES or COLUMNS clause.

**RW-303-S**  
**OCCURS not allowed in entry with multiple LINE/ COLUMN.**  
It is not permissible to combine an OCCURS and a LINE or COLUMN with multiple operands in the same entry.
**STEP/WIDTH/DEPTH allowed only with OCCURS.**
The STEP/WIDTH/DEPTH clause cannot be coded unless there is a valid OCCURS clause in the same entry.

**STEP/WIDTH/DEPTH not allowed in blank item.**
The STEP/WIDTH/DEPTH clause is not allowed in an elementary entry that has no COLUMN clause (actual or implied).

**STEP/WIDTH/DEPTH less than size of field: \( xx \) assumed.**
The distance given in the STEP/WIDTH/DEPTH clause must be at least the size (horizontal or vertical) of the item it applies to.

**Wrong axis implied by STEP/WIDTH/DEPTH.**
The keyword LINES may only be used in the vertical direction and COLUMNS in the horizontal direction.

**Scope of OCCURS includes absolute entry: minimum STEP (\( xx \)) assumed.**
An OCCURS clause includes within its scope an absolute entry (LINE or COLUMN without "+"). The value given is the minimum distance that may be assumed between adjacent entries.

**Mult. SOURCE/VALUE not subject to fixed OCCURS or mult. LINE/COLUMN.**
A multiple SOURCE or VALUE clause must be associated with either OCCURS without DEPENDING or a multiple LINES or a multiple COLUMNS.

**SOURCE NONE allowed only in multiple entry.**
SOURCE NONE can be used only as part of a multiple-operand SOURCE clause.

**Multiple LINE numbers must increase: PLUS 1 assumed as necessary.**
In a multiple LINES clause, absolute line numbers must be strictly increasing.

**REPORTS ALL already encountered: ignored.**
REPORTS ARE ALL can only be coded once per program.

**ALL should be only REPORT(S) operand: will be applied to unassigned reports.**
REPORTS ARE ALL must be coded without any report-names and must be the only REPORT(S) clause in the program.

**Invalid operand for VARYING.**
An operand following VARYING is not one of the permitted forms.

**VARYING allowed only with OCCURS or multiple LINE/COLUMN.**
The VARYING clause can be used only if there is an OCCURS clause or a multiple LINES or COLUMNS clause in the same entry.
VARYING data-name duplicated.
This message is issued if the same data-name has been used twice in the same VARYING clause.

Invalid VARYING data-name.
The names DATA-SUB-1/2/3/4 cannot be used as VARYING operands.

VARYING data-name already in use in an enclosing entry.
It is not permissible to use the same data-name for two nested VARYING clauses.

REPORT SECTION item cannot appear in expression in SUM clause.
The use of SUM expression is permitted only if all the items in the expression lie outside the REPORT SECTION.

Expression summed cannot contain SUM: parentheses required.
Expressions of the type SUM A + SUM B are not allowed because the SUM operand binds the entire arithmetical expression. For example, SUM A + B means SUM (A + B) not (SUM A) + B. The expression SUM A + SUM B must therefore be written (SUM A) + (SUM B).

Only SOURCE expression can contain SUM.
The SUM or COUNT term cannot be used in a DEPENDING ON expression, as a parameter to FUNCTION, or as part of a conditional expression.

SUM not allowed in multiple-choice entry.
The SUM clause or SUM term cannot be used in a multiple-choice entry. The code should be rewritten using separate entries.

SUM in lone DETAIL not allowed except as cross-foot.
If the report contains only one DETAIL group, that group cannot contain any SUM clauses except for cross-footing (totalling items within the same group).

COUNT does not refer to REPORT SECTION entry.
Only an entry defined in the REPORT SECTION can be counted.

Data-name of VARYING referred to in FROM or BY expression.
The FROM or BY expression of a VARYING clause refers to its own data-name. This may lead to unpredictable results.

COUNT cannot be followed by expression or literal.
Only simple data-names can be used in the COUNT clause.

RESET cannot be used with SUM/COUNT expression.
RESET is only allowed with the basic SUM or COUNT clause, not when they are a term in an expression.
Multiple-choice entry not allowed at LINE level.
The LINE clause cannot be written at the same level as COLUMN when the entry is a multiple-choice entry. They should be written at different levels.

Printer TYPE xx not recognized: default TYPE assumed.
The Printer Description File for the given "printer type" in the SELECT clause cannot be found.

STYLE xx is already in effect.
The given style is unnecessary since it is already in effect having been defined at a higher level.

Unknown STYLE name "xx": changed to NORMAL.
The given style cannot be found in the Printer Description File.

No code defined for this STYLE clause: changed to NORMAL.
This particular combination of styles is not permitted (although each style-name is valid).

Contradictory combination of STYLEs: changed to NORMAL.
You have coded two or more styles in combination that belong to the same mutually-exclusive class.

STYLE not allowed in unprintable entry.
An group entry with a STYLE clause must define at least one line or print column. (However, you can code STYLE in an elementary dummy entry.)

STYLE within multiple-choice selection cannot have condition.
A construct of the form:

```cobol
05 COL 1 VALUE "ONE" WHEN FLD-A = 1
   VALUE "TWO" WHEN FLD-A = 2
   STYLE UNDERLINE WHEN FLD-B = 1.
```

should be re-coded as:

```cobol
05 STYLE UNDERLINE WHEN FLD-B = 1.
07 COL 1 VALUE "ONE" WHEN FLD-A = 1
   VALUE "TWO" WHEN FLD-A = 2.
```

xx clause has invalid syntax: ignored.
The given clause is malformed.

PRESENT AFTER clause has duplicated phrase: discarded.
One of the options following PRESENT AFTER is repeated.

PAGE and JUST not allowed in PRESENT AFTER in PH or PF group.
The clause PRESENT JUST AFTER [NEW] PAGE can appear only in a body group.
RW-344-I  **LAST DETAIL assumed to be xx.**  
This informational message is issued if LAST DETAIL is not specified and its assumed value is not the same as PAGE LIMIT.

RW-345-W  **LAST BODY GROUP present with FOOTING.**  
LAST BODY GROUP means the same as FOOTING, so one is superfluous.

RW-346-S  **PAGE option allowed only with TYPE CH.**  
This refers to the OR PAGE phrase allowed only with a TYPE CH group.

RW-347-I  **LAST BODY GROUP assumed to be xx.**  
This message appears when no LAST BODY GROUP (or [LAST CONTROL] FOOTING) clause is specified and its value is inferred from the PAGE LIMIT clause and the size of any PAGE FOOTING group.

RW-348-I  **NEXT GROUP syntax inappropriate for this TYPE of group.**  
This is issued if NEXT DE OR CH GROUP is written in a TYPE DE or CH group or if NEXT BODY GROUP is written in a TYPE CF group, both of which imply a misunderstanding of the operation of the NEXT GROUP clause.

RW-349-S  **COLUMN RIGHT/CENTER must have absolute value: changed to COLUMN PLUS 1.**  
COLUMN RIGHT and COLUMN CENTER have no relative form.

RW-350-S  **MULTIPLE PAGE not allowed in type PH, PF or CH with OR PAGE.**  
These three group types must, by definition, be confined to one page and therefore cannot have a MULTIPLE PAGE clause.

RW-351-S  **(NO) MULTIPLE PAGE must have at least 2 subordinate lines.**  
A single line clearly cannot span two pages, so this clause is permitted only for a group of LINE entries, or a multiple LINES or LINE with OCCURS.

RW-352-S  **Clause "xx" not allowed in unpaged report.**  
A PAGE clause must be present in the RD if certain clauses, such as GROUP LIMIT are used.

RW-353-S  **GROUP LIMIT too low or too high for this group.**  
The GROUP LIMIT cannot be less than (that is, positionally higher than) the normal highest (earliest) position for the group, nor greater than (that is, positionally lower than) the normal latest position.

RW-354-S  **GROUP LIMIT allowed only in body group.**  
GROUP LIMIT cannot be used except with a TYPE CH, DE, or CF group.

RW-355-E  **NO MULTIPLE PAGE can be nested only within MULTIPLE PAGE.**  
No other nested combinations of these clause are permitted.
Some styles cannot be output directly and will be DEFERRED.
The precompiler may be unable to generate code to resolved certain styles directly, especially when they require “overprinting” of part of a line, or “shadow printing” of characters. In such cases, the precompiler defers the resolution of the codes to execution time and your program will therefore require the Printer Description File and some additional run time routines whenever it executes.

STYLE not allowed in individual multiple-choice entry.
A STYLE clause cannot be coded between the SOURCE or VALUE clause and the WHEN phrase of a multiple-choice entry. It can only apply to the complete entry.

Condition on STYLE not permitted in FD or RD.
Only an unconditional STYLE clause is allowed for FD or RD entries. The required effect must be implemented in the report groups.

Printer file was absent at precompile time and DEFERRED was assumed.
The default Printer Description File could not be found.

MODE PRNT has been assumed for file because of unresolved STYLE.
A file handler interface is required because certain STYLE clauses cannot be resolved at precompilation time but require a run time component.

STYLE may need to be on each LINE to take effect in page buffer.
If the PAGE BUFFER feature is in effect, it is inadvisable to code the STYLE clause at a level between RD and LINE. Take the following construct:

```
03  STYLE HIGHLIGHT.
05  LINE 1 ...
05  LINE 2 ...
```

The ending control sequence for HIGHLIGHT is not stored until the end of line 2. If any data is placed on the right of line 1 in the page buffer, it may unintentionally also have the HIGHLIGHT property. The sample should therefore be re-coded:

```
03  LINE 1    STYLE HIGHLIGHT...
03  LINE 2    STYLE HIGHLIGHT...
```

TYPE DEFERRED not allowed where record description follows FD.
Since a deferred printer requires a file handler, there cannot be any (01-level) record descriptions following the FD for the file (compare error no. RW-145-E).

WRAP/NO WRAP cannot be duplicated or nested.
The only nesting allowed for this clause is a single NO WRAP nested within a single WRAP.
RW-382-S  **WRAP cannot be used below LINE level.**
Only NO WRAP can be used at any level below that of LINE.

RW-383-E  **NO WRAP must be subordinate to WRAP.**
Unlike NO MULTIPLE PAGE, NO WRAP cannot be coded independently.

RW-384-E  **NO WRAP must encompass more than one elementary item.**
Since an elementary item cannot be split by the WRAP feature, NO WRAP is redundant unless it encloses more than one elementary printable item, or a multiple COLUMNS, or a COLUMN with OCCURS.

RW-385-S  **WRAP COLUMN integer should be between 1 and LINE LIMIT.**
The integer following the TO or AFTER phrase clearly must stay within the normal limits for the report.

RW-870-S  **Library member not found.**
The name specified in a COPY statement could not be found in SYSIBM or the indicated library.

RW-871-E  **REPLACING phrase has invalid syntax: discarded.**
The REPLACING phrase will be ignored and the COPY processed with no replacements.

RW-872-E  **Word "BY" not found: no replacements done.**
The end of a token (identifier, literal, pseudo-text, etc.) was reached in a COPY...REPLACING directive and the expected word BY was not found.

RW-873-E  **Too many levels of nesting of COPY.**
This message indicates that a COPY statement has been found within the text of another COPY member and the number of levels to which this is permitted has been exceeded. (See Installation and Operation for details.)

RW-874-E  **Pseudotext invalid or too long: no replacements done.**
Either the pseudotext on the left of BY is empty (that is: == ==) or it is longer than the maximum (normally 512 bytes).

RW-875-W  **No items were replaced during COPY.**
No match was made against the REPLACING phrase of the COPY statement during the COPY.

RW-876-E  **Invalid COPY syntax: passed to compiler unchanged.**
The word COPY is not followed by a word that could be a member-name.

RW-877-W  **Pseudotext brackets assumed as required by REPLACE.**
The format of the REPLACE statement requires that each operand should be within pseudotext brackets (== ==). These were assumed for this REPLACE statement.
BW-880-S **BASIS not allowed: passed to compiler but outcome unpredictable.**

The BASIS directive is not supported by the precompiler unless it is the first COBOL statement in the source. The results are unpredictable as the precompiler expects a complete source program.

BW-881-W **CBL/PROCESS statement passed to compiler but not actioned.**

CBL or PROCESS directives are accepted by the precompiler but not actioned by it. Since they may alter the compiler options which the precompiler is unaware of, the results may be unpredictable. See also Part 6.

BW-882-W **nn sequence errors were found in the program.**

This message may appear if the SEQUENCE option is set.
Index

Click on Page Numbers

A

abbreviated forms 30
ABSENT WHEN/AFTER clause - see PRESENT WHEN/AFTER
absolute form of COLUMN 11, 88
LINE 10, 105
LINE and COLUMN 122
NEXT GROUP 116
ADD statement, equivalence to ADV option 45, 48
ALL form of VALUE literal 196
ALL option of TYPE CF 189
ALLOW SOURCE SUM CORR clause 18, 53, 182, 185
alternating page formats 238
ANS-68 features summary 279
standard 3
ANS-74 differences 279
ANS-85 changes to FD 44
differences 279
GLOBAL clause in RD 51
USE GLOBAL statement 234
arithmetic expressions - see expressions averages, calculating using COUNT 94
axes of summing 26, 173

B

BLANK WHEN ZERO clause 86
BLOCK CONTAINS clause 45, 54
body groups definition 11
introduction 7
see also CH, DE, CF groups
boustrophedon sequence 201
BY phrase of VARYING 200

C

carriage control character 45, 55
CENTER option of COLUMN 11, 32
CF groups 7, 14, 16, 32, 57, 118, 190
CH groups 7, 33, 59, 190
with GROUP LIMIT 103
channels, use of in file handler 248
classes, of STYLE 164
CLOSE statement 16, 219, 228
implicit on STOP RUN and CANCEL 46
Coddasyl, extensions to language 279
CODE clause 18, 45, 54
with concurrent reports 241
with file handler 248
CODE-VALUE special register 55
COD - see COLUMN
COLUMN clause 10, 87
with OCCURS 120
column totals 26, 174
COLUMN-COUNTER special register 87, 93
compatibility with OS/VS COBOL, see also last part of each section of Parts 2, 3, and 4
concurrent reports 47, 240
conditions 13, 28, 139
consecutive reports 47
contained programs, with GLOBAL report 51
control breaks 15, 19
CONTROL clause 9, 19, 56
CONTROL FOOTING - see CF groups
CONTROL HEADING - see CH groups
CONTROL phrase of PRESENT WHEN 139
control-id, definition 56
CONTROLS clause - see CONTROL clause
controls with PRESENT AFTER 134
controls with SOURCE 160
COPY statement 284
correlation - see SOURCE SUM correlation
COUNT clause 94
COUNT, as term 27
cross-footing form of SUM 170, 181, 214, 216
CTIME function 98
cumulative totals- see RESET phrase
CURRENCY SIGN phrase 129
CURRENT-DATE special register 97, 160
customization - see Installation and Operation

D

DATA DIVISION - see REPORT SECTION
data-name
of report entry 9, 14
of VARYING 199
referred by SUM 167
database
commands 285
in FUNCTION 245
input from 16, 60, 232
output to 249
DATE function 34, 98
DAY function 98
DAYSIN function 98
DBCS 45, 82, 129, 195, 196
DE groups 7, 190
DECLARATIVES 127, 230
use of total fields 181
default
for PAGE sub-clauses 71
for TYPE 188
DEFERRED option with STYLEs 163
DEPENDING ON phrase 23, 122, 124
of OCCURS 23, 122, 124
of OCCURS with PRESENT WHEN 146
DEPTH - see STEP
DETAIL - see DE groups
diagnostics - see messages
DISPLAY-1 - see USAGE
DOS/VS COBOL – see OS/VS COBOL
Double-Byte Character Set - see DBCS
dummy COLUMN 143
dummy entries, with FUNCTION 96
dummy groups 119, 155
for subtotalling 183
DUPUCATED clause 47, 241

F
FD entry 14, 43, 50
fiche - see microfiche
File Control Area 251
file handler - see Independent Report File Handlers
FILE SECTION 43
FILE STATUS clause 45
FILE-CONTROL paragraph 47
FINAL control 56, 59
Fips flagging option 284
FIRST DETAIL clause 8, 69, 70
FIRST PAGE NO ADVANCING clause of FD 49
FOOTING sub-clause - see LAST CF
FUNCTION clause 34, 96
developing a Function 244
sample routine 246

G
GENERATE statement 15, 59, 213
effect on REPEATED 153
GLOBAL
in USE BEFORE REPORTING 231, 234
report files 46, 49
report groups 213
reports 18, 51, 219, 228
Glossary
GROUP INDICATE 133
GROUP LIMIT clause 32, 103

H
HEADING sub-clause 68, 70, 109
hexadecimal literals 196
hierarchy of controls 19, 59
HOLD status - see SET statement
horizontal repetition - see COLUMN, OCCURS

I
IBM extensions, list 279
identifier form of CODE 55
in-line comments 284
optional entries - see PRESENT
OS/390 - see Installation and Operation
OS/VS COBOL
comparisons of formats:
  FILE SECTION and RD
  PROCEDURE DIVISION
  Report Groups
compatibility
  also at end of each section
migration from
OSVS precompiler option
and COLUMN clause
and SOURCE SUM correlation
OTHER option of PRESENT WHEN
OVERFLOW clause

page advance processing - see page-fit test
Page Buffer feature
PAGE clause - see PAGE LIMIT clause
PAGE FOOTING - see PF groups
PAGE FOOTING totals
PAGE HEADING - see PH groups
PAGE LIMIT clause
  PAGE option
    of CH group
    of PRESENT AFTER
    page, regions of
    PAGE-COUNTER
page-fit test
parameters to file handler
parameter to FUNCTION
PF groups
  PH groups
  PICTURE clause
  with VALUE
PLUS - see relative forms
pre-break values of controls
precision of totals
precompiler
Preface
PRESENT AFTER clause
PRESENT WHEN clause
  effect on SUM
  OTHER option
  using total fields
  with OCCURS
PRINT-SWITCH

Printer Description File
PRNT file handler
procedural references to Report Writer data-names
PROCEDURE DIVISION
purpose of Report Writer

Q
qualification
  of CODE-VALUE
  of control-id
  of DETAIL group name
  of LINE-COUNTER
  of PAGE-COUNTER
  of SET statement
  of SOURCE operand
  of total fields
  of UPON operand
  using control-id

R
railroad track, explanation
RANDOM PAGE feature
RD entry
RDATE function
RECORD CONTAINS clause
record description, after FD
RECORDING MODE
  with a file handler
reference modification
relative form of
  COLUMN
  LINE
  NEXT GROUP
relative subscriptsing
REPLACE - see SET
REPEATED clause
repetition - see OCCURS, REPEATED, multiple forms
REPLACE BY option of OVERFLOW
REPLACE statement
REPLACING option of COPY
REPORT as CONTROL operand
Report Control Area
report files
REPORT FOOTING - see RF groups
Report Group Descriptions
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>report groups (see also TYPE clause)</td>
<td>4, 5</td>
</tr>
<tr>
<td>coding rules</td>
<td>82</td>
</tr>
<tr>
<td>definition</td>
<td>77</td>
</tr>
<tr>
<td>REPORT HEADING - see RH groups</td>
<td></td>
</tr>
<tr>
<td>REPORT option of PRESENT AFTER</td>
<td>136</td>
</tr>
<tr>
<td>REPORT SECTION</td>
<td>8, 50</td>
</tr>
<tr>
<td>REPORT SECTION SUM</td>
<td>169</td>
</tr>
<tr>
<td>report-name - choice of</td>
<td></td>
</tr>
<tr>
<td>report-name – definition</td>
<td>45</td>
</tr>
<tr>
<td>REPORT-NUMBER</td>
<td>36, 241</td>
</tr>
<tr>
<td>reports - introduction to</td>
<td>4</td>
</tr>
<tr>
<td>REPORTS clause of FD</td>
<td>14, 18, 45</td>
</tr>
<tr>
<td>reserved words</td>
<td>287</td>
</tr>
<tr>
<td>RESET phrase of SUM</td>
<td>168, 181, 186</td>
</tr>
<tr>
<td>RF groups</td>
<td>7, 16, 119, 190</td>
</tr>
<tr>
<td>use for writing trailer</td>
<td>45</td>
</tr>
<tr>
<td>RH groups</td>
<td>7, 114, 119, 190</td>
</tr>
<tr>
<td>use for writing header</td>
<td>45</td>
</tr>
<tr>
<td>RIGHT option of COLUMN</td>
<td>11, 32</td>
</tr>
<tr>
<td>RMDATE function</td>
<td>100</td>
</tr>
<tr>
<td>rolling forward of totals</td>
<td>90, 94, 170, 175, 181, 214</td>
</tr>
<tr>
<td>ROUNDED phrase</td>
<td>12, 159</td>
</tr>
<tr>
<td>row totals</td>
<td>26, 174</td>
</tr>
<tr>
<td>run time messages</td>
<td>63, 66, 143, 152, 172, 181, 215, 224, 229</td>
</tr>
<tr>
<td>RYDATE function</td>
<td>100</td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SELECT...ASSIGN clause</td>
<td>14, 44</td>
</tr>
<tr>
<td>SET page buffer statements</td>
<td>36, 221</td>
</tr>
<tr>
<td>shorter forms of syntax</td>
<td>30</td>
</tr>
<tr>
<td>SIGN clause</td>
<td>156</td>
</tr>
<tr>
<td>size errors - see OVERFLOW clause</td>
<td></td>
</tr>
<tr>
<td>sneaking columns</td>
<td>224</td>
</tr>
<tr>
<td>snapshots of total field</td>
<td>178</td>
</tr>
<tr>
<td>sorting, via COBOL SORT</td>
<td>57</td>
</tr>
<tr>
<td>SOURCE clause</td>
<td>12, 158</td>
</tr>
<tr>
<td>as subject of SUM</td>
<td>167</td>
</tr>
<tr>
<td>OVERFLOW checks for expressions</td>
<td>66</td>
</tr>
<tr>
<td>referring to total field</td>
<td>178</td>
</tr>
<tr>
<td>SOURCE SUM correlation</td>
<td>53, 90, 172, 182, 183, 185, 214</td>
</tr>
<tr>
<td>SPECIAL- NAMES paragraph</td>
<td></td>
</tr>
<tr>
<td>spooling report data</td>
<td>18, 54, 241, 250</td>
</tr>
<tr>
<td>squeeze symbol &quot;&lt;&quot;</td>
<td>31, 131</td>
</tr>
<tr>
<td>STATE function</td>
<td>101</td>
</tr>
<tr>
<td>STATEF function</td>
<td>101</td>
</tr>
<tr>
<td>STEP phrase</td>
<td>23, 121</td>
</tr>
<tr>
<td>STIME function</td>
<td>101</td>
</tr>
<tr>
<td>STYLE clause</td>
<td>163, 196</td>
</tr>
<tr>
<td>at FD level</td>
<td>47</td>
</tr>
<tr>
<td>at RD level</td>
<td>52</td>
</tr>
<tr>
<td>subheadings</td>
<td>135</td>
</tr>
<tr>
<td>subscripts - see VARYING</td>
<td></td>
</tr>
<tr>
<td>subtotalling</td>
<td>60, 172, 178, 181, 182, 184, 214</td>
</tr>
<tr>
<td>SUM</td>
<td></td>
</tr>
<tr>
<td>as a term</td>
<td>27, 167</td>
</tr>
<tr>
<td>clause</td>
<td>14, 18, 22, 26, 167</td>
</tr>
<tr>
<td>from a different report</td>
<td>175</td>
</tr>
<tr>
<td>use of total fields</td>
<td>176</td>
</tr>
<tr>
<td>with PRESENT WHEN</td>
<td>143, 175, 180</td>
</tr>
<tr>
<td>SUM OVERFLOW clause</td>
<td>18, 65, 172</td>
</tr>
<tr>
<td>sum-counter - see total fields</td>
<td></td>
</tr>
<tr>
<td>summary reporting</td>
<td>6, 19, 155, 182, 183, 189, 213</td>
</tr>
<tr>
<td>SUPPRESS PRINTING statement</td>
<td>227</td>
</tr>
<tr>
<td>symbolic characters, as VALUE</td>
<td>196</td>
</tr>
<tr>
<td>Syntax summary</td>
<td>289</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>tables - see OCCURS, SUM</td>
<td></td>
</tr>
<tr>
<td>TERMINATE statement</td>
<td>16, 228</td>
</tr>
<tr>
<td>with multiple reports</td>
<td>237</td>
</tr>
<tr>
<td>with total fields</td>
<td>181</td>
</tr>
<tr>
<td>TIME function</td>
<td>34, 101</td>
</tr>
<tr>
<td>TIME-OF-DAY special register</td>
<td>160</td>
</tr>
<tr>
<td>total fields</td>
<td>169</td>
</tr>
<tr>
<td>total fields, uses</td>
<td>176</td>
</tr>
<tr>
<td>totalling - see SUM</td>
<td></td>
</tr>
<tr>
<td>totals only reports</td>
<td>183</td>
</tr>
<tr>
<td>truncation</td>
<td></td>
</tr>
<tr>
<td>of expressions</td>
<td>66</td>
</tr>
<tr>
<td>of PICTURE</td>
<td>158</td>
</tr>
<tr>
<td>of line</td>
<td>144</td>
</tr>
<tr>
<td>of records</td>
<td>48</td>
</tr>
<tr>
<td>of totals</td>
<td>66, 169, 177</td>
</tr>
<tr>
<td>TYPE clause</td>
<td>6, 10, 20, 51, 188</td>
</tr>
<tr>
<td>TYPE clause of SELECT</td>
<td>46</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>UNLESS phrase of PRESENT</td>
<td>140</td>
</tr>
<tr>
<td>unprintable fields</td>
<td>90, 147, 164, 175, 177, 180</td>
</tr>
<tr>
<td>UPON phrase of INITIATE</td>
<td>45, 219</td>
</tr>
<tr>
<td>UPON phrase of SUM</td>
<td>168, 172, 184, 185, 216</td>
</tr>
<tr>
<td>USAGE clause</td>
<td>195</td>
</tr>
</tbody>
</table>
USE BEFORE REPORTING directive 127, 181, 211, 213, 217, 227, 230
User-written extensions - see Independent Report File Handlers, FUNCTION clause USING phrase of MODE 47

V
VALUE clause 12, 196
as subject of SUM 169
variable number of repetitions 23
variable-length fields 31, 32, 86, 89, 93, 131
variable-length records 48
variable-position fields 64
VARYING clause 24, 199
verbs used in Report Writer 4, 211
vertical repetition - see LINE, OCCURS 47
VM - see Installation and Operation

W
WIDTH - see REPEATED clause, STEP 284
wild cards, in COPY statement 284
WITH CODE - see CODE
WITH PAGE BUFFER - see Page Buffer feature
WITH RANDOM PAGE - see RANDOM PAGE feature
WRAP clause 34, 63, 203
WRITE statements, explicit 45

Y
YDATE function 101

Z
zero divide - see OVERFLOW 102
ZIP function 102

etc
3800 - see laser printer 31, 131
< and > PICTURE symbols 31, 131