

Compatibility With Built-In COBOL-IT Report Writer

Apart from a few insubstantial differences, listed and explained in Part 6, COBOL-IT 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-IT 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.

1.1.2 Gentle Introduction

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-IT 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.

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

1.1.5 Other Features

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 *PICTURE*s. 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) .
```

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
```



```
RD CLUB-EXPENDITURE
PAGE LIMIT 60 FIRST BODY GROUP 3 LINE LIMIT 132
CONTROL IS SPORT.

...
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 "====".
```

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:

```
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.

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:

```
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.
```

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

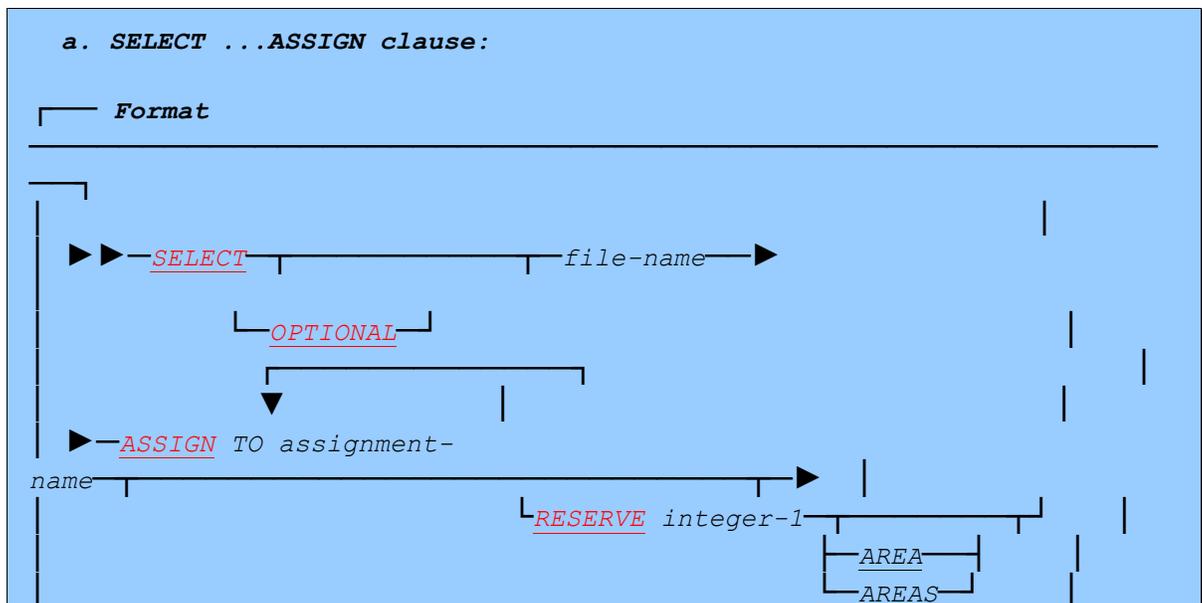
OVERFLOW	Specifies action to be taken when expression or SUM overflows	no	
ALLOW SOURCE SUM CORR	Selects ANS-85 or ANS-68 rules for SUMming	no	

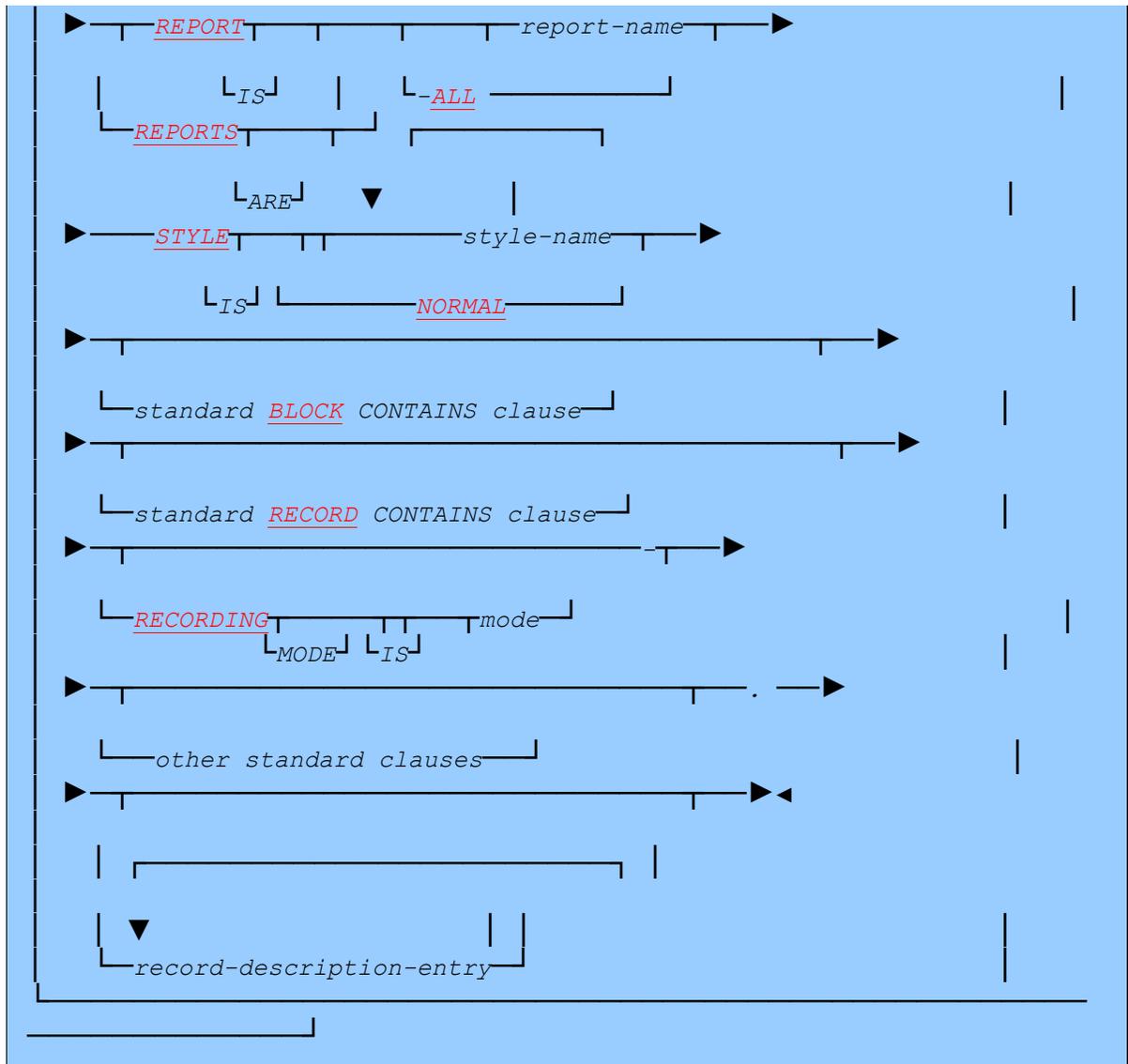
1.2.2 Report Files

COBOL-IT 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.





Select and FD: Coding Rules

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.

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`.

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).

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.

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**.

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](#)).

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.

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.

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.

FILE-CONTROL and FD: Operation

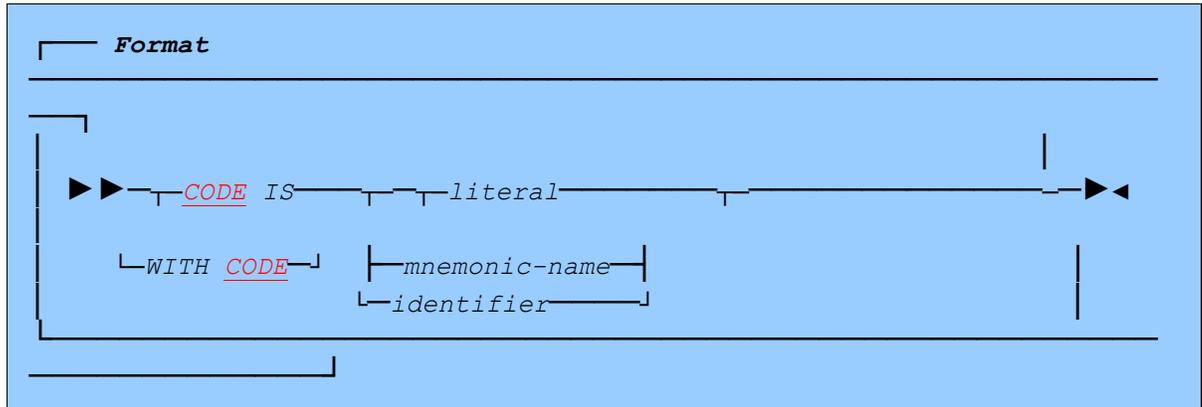
If you specify **more** than one report-name in the same **REPORTS ARE** clause, you will be able to generate report data either consecutively or concurrently for the same file. (**REPORT IS** and **REPORTS ARE** are interchangeable, however many report-names follow.) Tips on creating more than one report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

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*.

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

1.2.5 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.



CODE Clause: Coding Rules

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

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.

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

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.

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

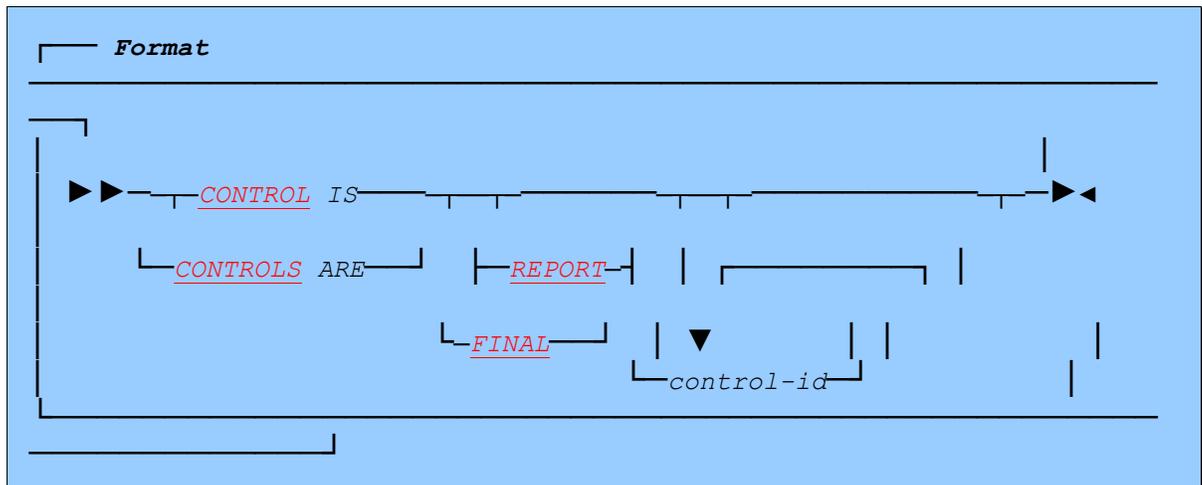
literal IS mnemonic-name

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.

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.

1.2.6 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*).



CONTROL Clause: Coding Rules

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.

`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`.

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.

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.

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 vary the number of REPEATED groups that may be placed side-by-side (see 3.19 **REPEATED clause**),
as one means of adjusting the *right margin* when the WRAP clause is used to produce *line wrap round* (see 3.28 **WRAP clause**),
to check for (illegal) line overflow in variable-position report fields when the WRAP clause is **not** used.

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*).

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**).

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.

Compatibility

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.



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 **hhmmssff**, where ff 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 - hhmsstt

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

05	COL 31	PIC <X(9)	FUNC DAY.
05	COL +2	PIC <99/<99/9(4)	FUNC MDATE.
05	COL 51	PIC 99,99,99	FUNC TIME.

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.



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 S9(9) COMP-3. This is then output in the form nnnnn-*nnnn*, but the final -*nnnn* is left blank if the last four digits are 9999.

Compatibility

Only new Report Writer provides FUNCTION as an independent clause.

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.

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:

A multiple LINES clause is **functionally equivalent** to a **LINE** with an **OCCURS** clause; for example:

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.

A **simple** (single-operand) VALUE, SOURCE, SUM, or FUNCTION is **repeated** in every occurrence of the line.

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.

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.

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

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

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.

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*:

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.

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 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 <9(4) because a value of all zeros is reproduced as a zero).

Here are some examples of the "<" symbol:

a. If PICTURE **\$9999.99** gives you the result:

\$0000.50

then PICTURE **\$9<999.<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

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:

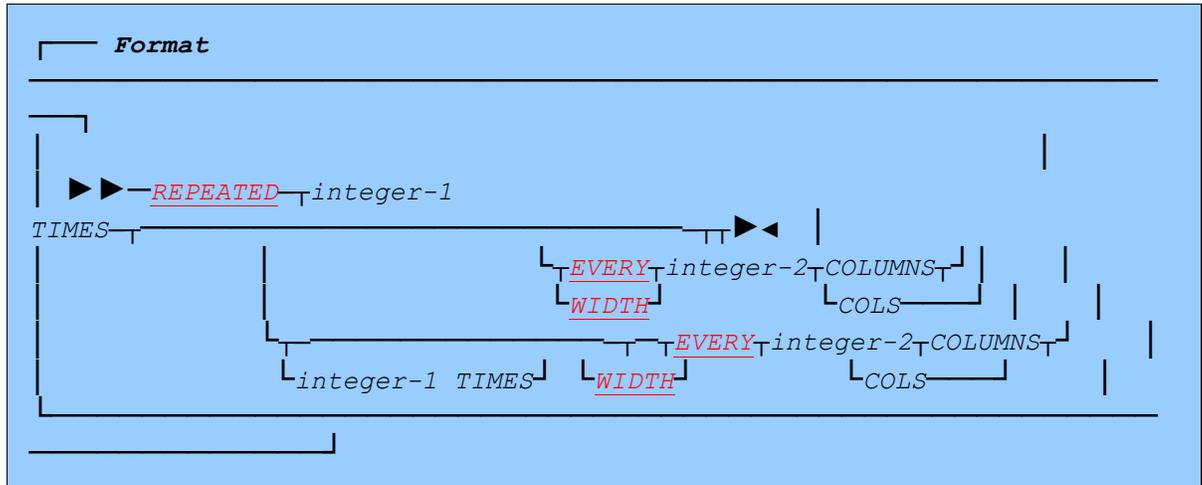
1

respectively

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

1.3.19 REPEATED clause

The REPEATED clause arranges body groups **side-by-side** across the page.



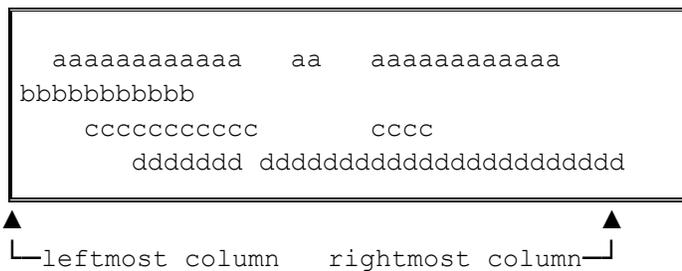
REPEATED Clause: Coding Rules

Write this clause at the 01-level of body groups (*DETAIL* or *CH/CF*) only.

You must code **either** the *TIMES* phrase **or** the *EVERY/WIDTH* phrase, or (preferably) both. *EVERY* and *WIDTH* have the same meaning.

Code **only the left-hand report group**. Report writer will automatically offset successive groups to the right of the left-hand group.

If you use the **EVERY/WIDTH** phrase, draw an imaginary "**smallest box**" around your group:



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.

**TERMINATE report
CLOSE all report files and input files**

See 2.6 **CONTROL clause**, and 4.2 **GENERATE statement** for further details.

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:

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**).

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.

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:

```
03 LINE OCCURS 4.
04 OCCURS 3.
05 COLS +2, +1 PIC... SOURCES ARE .....
```

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).

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:

```
03 LINE 2 STEP 1 OCCURS 3.
05 COL 2 STEP 5 OCCURS 4 PIC XXX SOURCES ARE
MONTH-NAME(1) MONTH-NAME(2) MONTH-NAME(3) MONTH-NAME(4)
MONTH-NAME(8) MONTH-NAME(6) MONTH-NAME(7) MONTH-NAME(5)
MONTH-NAME(9) MONTH-NAME(10) MONTH-NAME(11) MONTH-NAME(12).
```

will result in:

JAN	FEB	MAR	APR
AUG	JUN	JUL	MAY
SEP	OCT	NOV	DEC

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**.

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).

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:

```
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.

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

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:

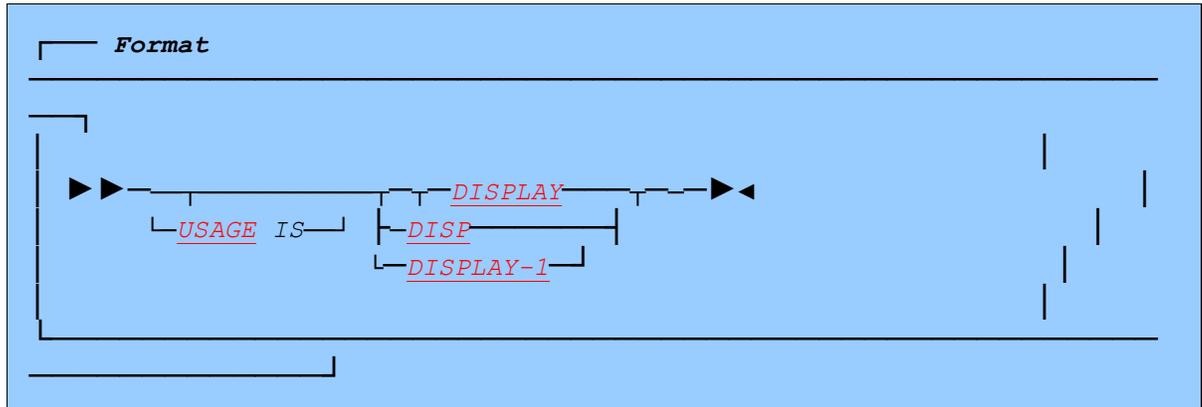
```
03 COL 20 PIC ZZZ9.99 SUM OF ACC-BAL RESET ON BRANCH-CODE.
```

you ensure that the total field is not cleared until a new BRANCH-CODE is reached.

Here is an example of a cumulative total:

1.3.25 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.



USAGE Clause: Coding Rules

The USAGE clause may be coded at any level, but **no** item may be subject to **both** USAGE DISPLAY **and** USAGE DISPLAY-1.

DISP is synonymous with **DISPLAY**.

Only *non-DBCS* items may be subject to **USAGE DISPLAY** and only *DBCS* items may be subject to **USAGE DISPLAY-1**.

No other forms of the USAGE clause are permitted in the REPORT SECTION.

USAGE Clause: Operation

DISPLAY is retained for consistency with basic COBOL but it is never required.

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.

of alignment. The **STYLE** clause is designed specifically for this purpose and has none of these drawbacks. (See 3.22 **STYLE clause**.)

VALUE Clause: Operation

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.

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.

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**.)

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:

If you wish to place no value in a particular occurrence, you may simply code a space character: "".

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.

All the literals must be either *DBCS* or *non-DBCS*.

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 .....
```

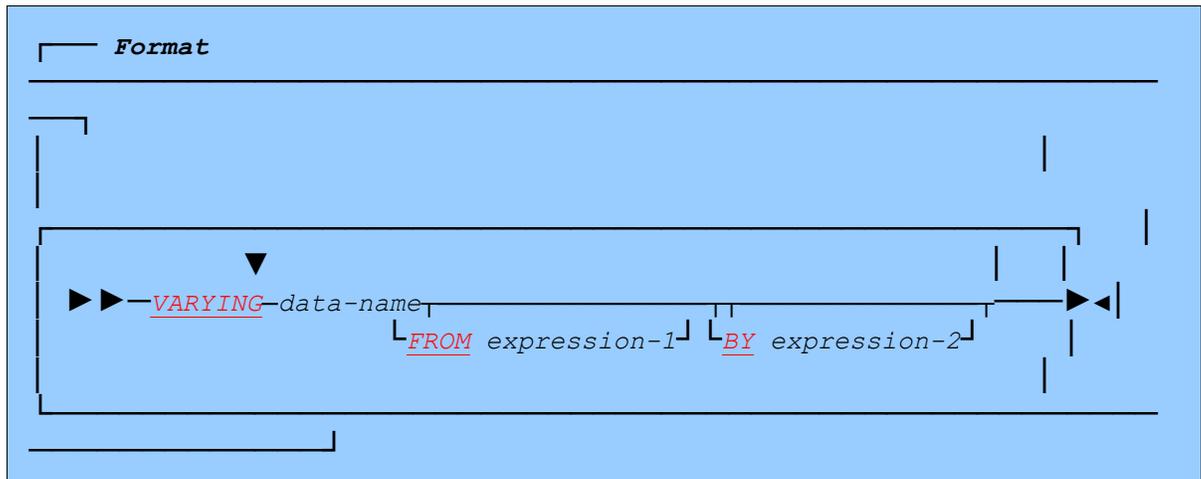
the number of literals should be either 4 (just the inner dimension), 12 (the product of the inner two dimensions), or 24 (all the dimensions).

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.      _____ will result in:
```


1.3.27 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.



VARYING Clause: Coding Rules

You may write **any number** of different *data-name* operands in this clause, each with an optional associated **FROM** and **BY** phrase.

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).

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.)

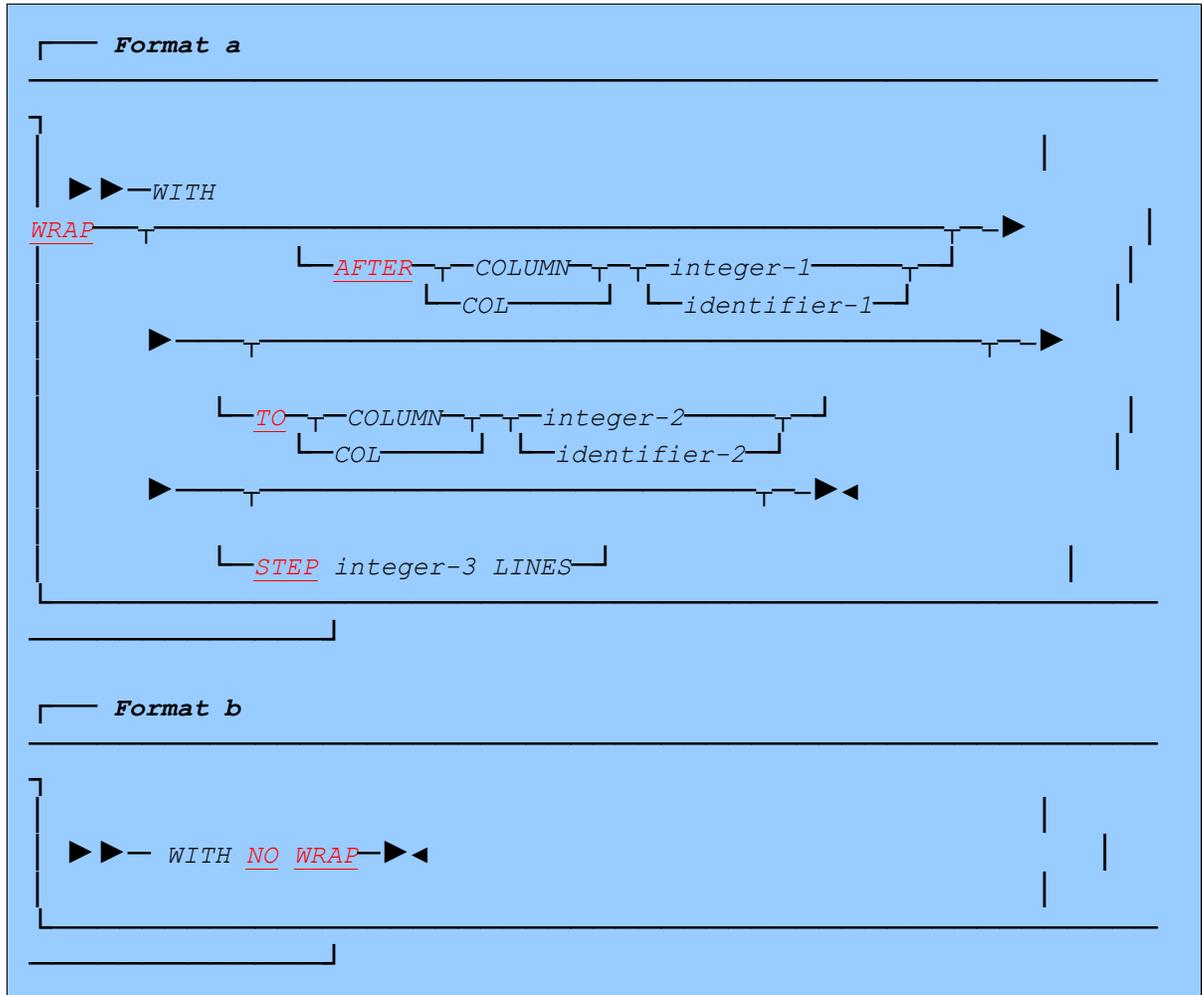
You **can re-use the same data-names** in different VARYING clauses, provided that you do not do this when the clauses are **nested** (enclosed one within the other). For example, you could write **VARYING R-LINE** on each repeating LINE, and **VARYING R-COL** on each repeating COLUMN, throughout your program.

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.)

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-

1.3.28 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.



WRAP Clause: Coding Rules

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.

1.4.1 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.

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 |

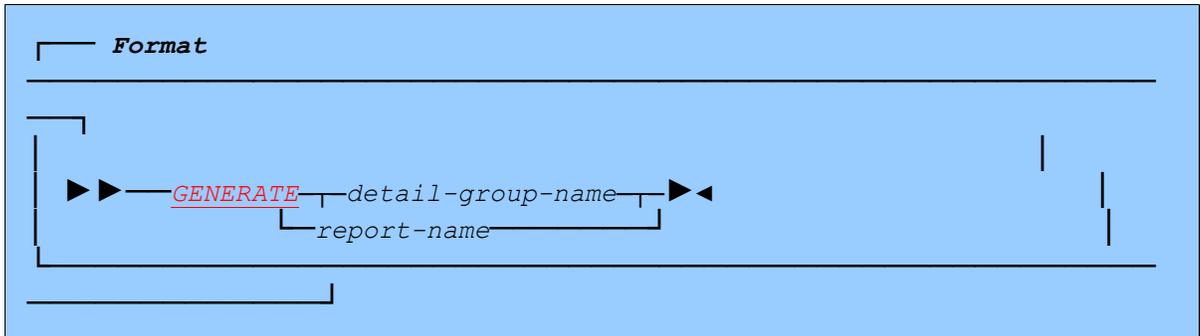
with the following basic plan for the PROCEDURE DIVISION:

```

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
    
```


1.4.2 GENERATE statement

The **GENERATE** statement is COBOL-IT 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.



GENERATE Statement: Coding Rules

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**.

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.

GENERATE must **not** appear in a **USE BEFORE REPORTING directive** Declarative.

GENERATE Statement: Operation

The **GENERATE** statement causes report writer to perform three main actions in an average report:

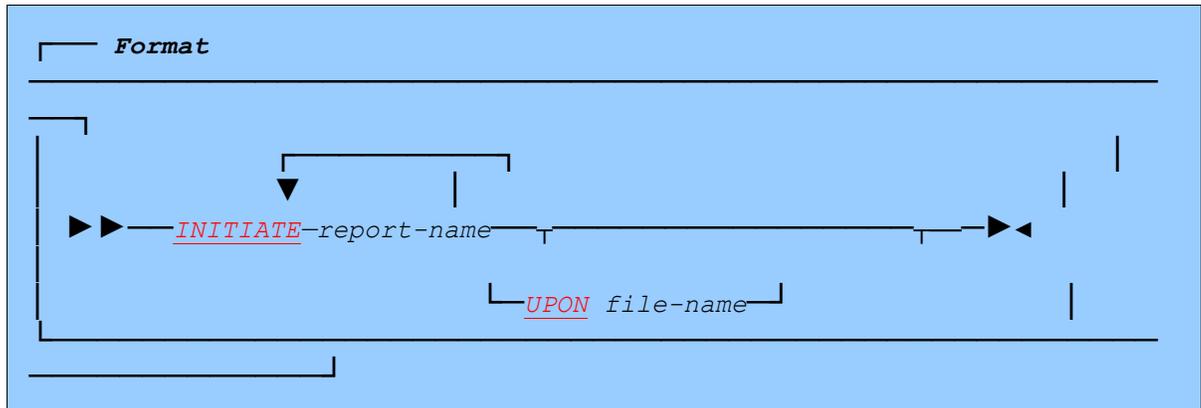
- It tests for control breaks, producing **CONTROL FOOTING** and **HEADING** groups where necessary,

- 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**),

- It generates each line in the **DETAIL** group, unless you are doing summary reporting (**GENERATE report-name**).

1.4.3 INITIATE statement

The **INITIATE** statement must be the first report writer statement to be executed for a report.



INITIATE Statement: Coding Rules

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.)

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.

INITIATE must **not** appear in a **USE BEFORE REPORTING directive** *Declarative*.

INITIATE Statement: Operation

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.

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.

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.

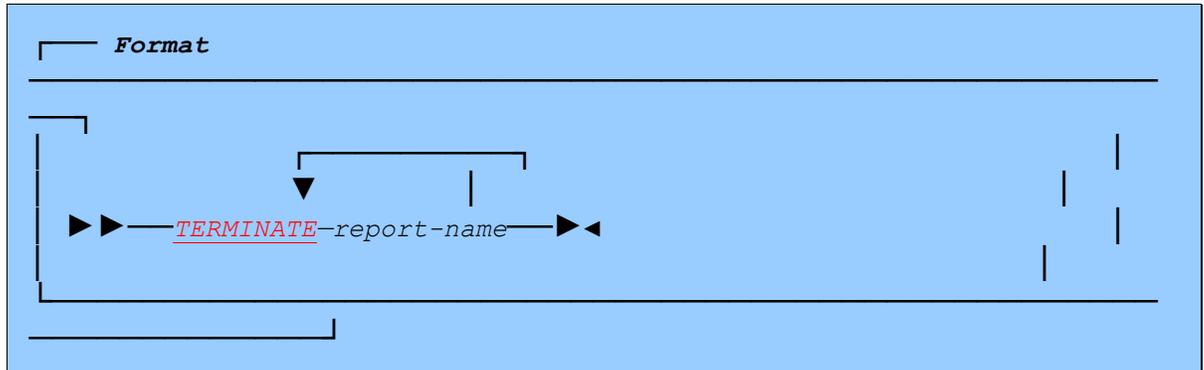
If an **UPON** phrase is present, the report will be written only to the file specified.

INITIATE Processing Cycle

The following is a more thorough description of each stage in the execution of an **INITIATE** statement:

1.4.6 TERMINATE statement

The TERMINATE must be the **last** report writer statement to be executed for each report.



TERMINATE Statement: Coding Rules

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.)

TERMINATE must **not** appear in a **USE BEFORE REPORTING directive** *Declarative*.

TERMINATE Statement: Operation

A **TERMINATE** must be executed for every report that has been INITIATED before the final close-down of the program.

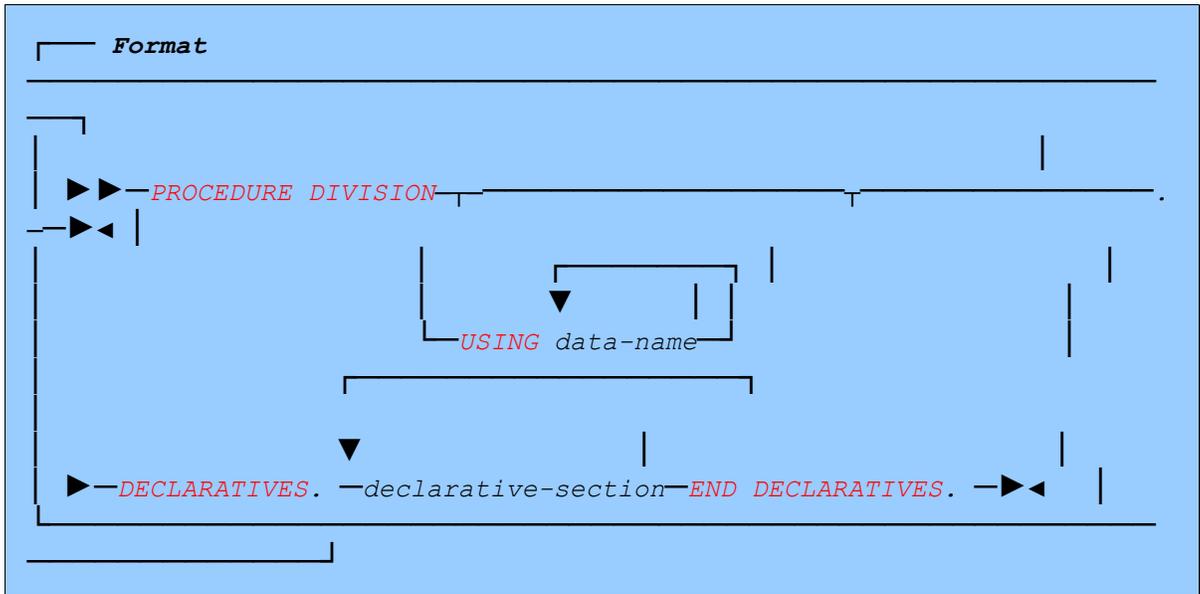
The TERMINATE statement clears any **pending REPEATED groups** or *Page Buffer* contents. It also outputs any final **CONTROL FOOTING, PAGE FOOTING** and **REPORT FOOTING** groups that may be required at the end of the report. It then returns the report to an "uninitiated" state. **PAGE-COUNTER** and **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**).

A separate, subsequent **CLOSE** should be executed for the associated report file. TERMINATE does **not** CLOSE the file.

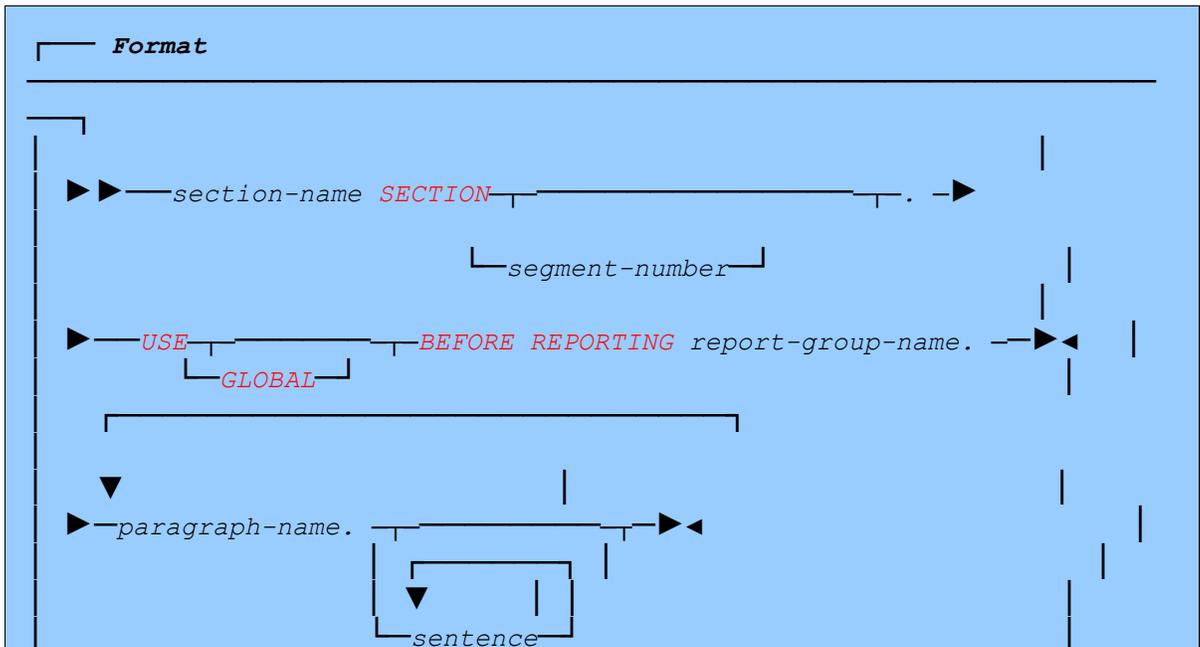
If a TERMINATE is executed without **any** GENERATE statements being executed for the report since the INITIATE was executed, **no** output at all is produced. If you wish to ensure that at least the REPORT HEADING and REPORT FOOTING groups appear, you should in this case GENERATE a **blank DETAIL group** before the TERMINATE.

1.4.7 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.



where *declarative-section* is defined as:



*

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.

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

1.5.3 Independent Report File Handlers

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.

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 PRNT

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.

Appendix A

List of Post-1968 Extensions

This section lists the extensions to the ANSI 1968 COBOL-IT Report Writer standard that are included in COBOL-IT 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-IT 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.

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.
- ¶ **FIRST PAGE 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.

PAGE LIMIT Clause

- (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.

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-IT Report Writer. They are therefore listed here in some detail:

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
```

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-??==.
```


Index

Click on Page Numbers

A

abbreviated forms	35
ABSENT WHEN/AFTER clause - see <i>PRESENT WHEN/AFTER</i>	
absolute form of	
COLUMN	11, 105
LINE	11, 125
LINE and COLUMN	144
NEXT GROUP	137
ADD statement, equivalence to	196
ADV option	54, 57
ALL form of VALUE literal	223
ALL option of TYPE CF	215
ALLOW SOURCE SUM CORR clause	19, 63 , 206, 210
alternating page formats	268
ANS-68	
features summary	315
standard	3
ANS-74 differences	315
ANS-85	
changes to FD	52
differences	315
GLOBAL clause in RD	61
USE GLOBAL statement	263
arithmetic expressions – see <i>expressions</i>	
averages, calculating using COUNT	112
axes of summing	30, 197

B

BLANK WHEN ZERO clause	103
BLOCK CONTAINS clause	54, 65
body groups	
definition	12
introduction	7
see also <i>CH, DE, CF groups</i>	
boustrophedon sequence	228
BY phrase of VARYING	227

C

carriage control character	54, 66
CENTER option of COLUMN	11, 37

CF groups	7, 15, 17, 37, 69, 139, 216	
CH groups	7, 38, 71, 216	
with GROUP LIMIT		122
channels, use of in file handler		281
classes, of STYLE		187
CLOSE statement	17, 246, 257	
implicit on STOP RUN and CANCEL		55
Codasyl, extensions to language		315
CODE clause	19, 54, 65	
with concurrent reports		271
with file handler		281
CODE-VALUE special register		66
COL - see <i>COLUMN</i>		
COLUMN clause	10, 104	
with OCCURS		142
column totals	29, 198	
COLUMN-COUNTER special register	104, 110	
compatibility with OS/VS COBOL, see also last part of each section of Parts 2, 3, and 4		4
concurrent reports	56, 270	
conditions	14, 32, 162	
consecutive reports		56
contained programs, with GLOBAL		
report		61
control breaks	16, 20, 71 , 240	
CONTROL clause	9, 20, 68	
CONTROL FOOTING - see <i>CF groups</i>		
CONTROL HEADING - see <i>CH groups</i>		
CONTROL phrase of PRESENT WHEN		162
control-id, definition		68
CONTROLS clause - see <i>CONTROL clause</i>		
controls with PRESENT AFTER		157
controls with SOURCE		183
COPY statement		321
correlation - see <i>SOURCE SUM correlation</i>		
COUNT clause		112
COUNT, as term		31
cross-footing form of SUM	194, 206, 241, 243	
CTIME function		116
cumulative totals- see <i>RESET phrase</i>		
CURRENCY SIGN phrase		151
CURRENT-DATE special register	116, 183	
customization - see <i>Installation and Operation</i>		

D

DATA DIVISION - see *REPORT SECTION*

data-name	
of report entry	10, 15
of VARYING	226
referenced by SUM	192
database	
commands	322
in FUNCTION	276
input from	17, 72, 262
output to	282
DATE function	39, 116
DAY function	116
DAYSIN function	117
DBCS	53, 98, 151, 221 , 223
DE groups	7, 216
DECLARATIVES	149, 260
use of total fields	206
default	
for PAGE sub-clauses	84
for TYPE	214
DEFERRED option with STYLEs	186
DEPENDING ON phrase	
of OCCURS	26, 144, 146
of OCCURS with PRESENT WHEN	169
DEPTH - see <i>STEP</i>	
DETAIL - see <i>DE groups</i>	
diagnostics - see <i>messages</i>	
DISPLAY-1 - see <i>USAGE</i>	
DOS/VS COBOL – see <i>OS/VS COBOL</i>	
Double-Byte Character Set - see <i>DBCS</i>	
dummy COLUMN	166
dummy entries, with FUNCTION	114
dummy groups	140, 178
for subtotalling	207
DUPLICATED clause	56 , 271

E

elementary entries - see <i>COLUMN clause</i>	
error messages - see <i>messages</i>	
execution time - see <i>run time</i>	
exit routine - see <i>USE BEFORE REPORTING</i>	
expressions	12, 72, 78, 181, 191, 200
extensions, list of	315
EXTERNAL attribute, of report file	58
EXTERNAL report files	55
external SUM - see <i>non-REPORT SECTION SUM</i>	

F

FD entry	15, 50, 59
fiche	- see <i>microfiche</i>
File Control Area	284
file handler	- see <i>Independent Report File Handlers</i>
FILE SECTION	50
FILE STATUS clause	53
FILE-CONTROL paragraph	56
FINAL control	68, 71
<i>Fips</i> flagging option	320
FIRST DETAIL clause	8, 82, 83
FIRST PAGE NO ADVANCING clause of FD	58
FOOTING sub-clause	- see <i>LAST CF</i>
formats, summary of all	325
FROM phrase of VARYING	227
front sheet	- see <i>RH groups</i>
FUNCTION clause	39, 114
developing a Function	275
sample routine	278

G

GENERATE statement	16, 71, 240
effect on REPEATED	176
GLOBAL	
in USE BEFORE REPORTING	261, 263
report files	55 , 58
report groups	240
reports	19, 61 , 246, 257
Glossary	341
GROUP INDICATE	157
GROUP LIMIT clause	37, 122

H

HEADING sub-clause	81, 83, 130
hexadecimal literals	223
hierarchy of controls	20, 71
HOLD status	- see <i>SET statement</i>
horizontal repetition	- see <i>COLUMN, OCCURS</i>

I

IBM extensions, list	315
identifier form of CODE	66
in-line comments	321

Independent Report File Handlers	19, 40, 54, 244, 248, 281
how to write one	282
sample	291
use of CODE data	66
INITIATE statement	16, 246
with multiple reports	267
with total fields	199
insertion characters, in PICTURE	152
intermediate source	3

J

JCL - see Installation and Operation	
JUST option of PRESENT AFTER	158
JUSTIFIED clause	123

K

Kanji - see <i>DBCS</i>	
keyword tables	
report files and RD	47
report groups	89
verbs	238

L

label printing	176
laser printers	40, 187
LAST CF sub-clause	19, 84, 128
LAST CONTROL FOOTING - see <i>LAST CF</i> sub-clause	
LAST DETAIL sub-clause	19, 82
layout of page	83
layout of report	5
left-shift symbol "<"	153
level-numbers	10, 98
LINAGE (prohibited clause)	53
LINE clause	10, 125
LINE clause with OCCURS	26, 143
LINE LIMIT Clause	8, 76
identifier form	176
LINE-COUNTER	12, 127, 133 , 137, 244, 249
uses	133
LINKAGE areas for file handler	284
lower-case, use in REPORT SECTION	45

M

MDATE function	39, 117
MDAYS function	117
messages	349
microfiche	283
MODE clause of SELECT	19, 53, 55 , 65
MODL file handler	55, 281, 291
modular programs, using Report Writer	291
MONTH function	117
MOVE function	119
multiple multiple form of	
COLUMN	25, 108 , 184
CONTROL FOOTING	112, 162, 168, 197, 215
LINE	26, 131 , 184
SOURCE	25, 145, 174, 184
SUM	30, 192
VALUE	25, 131, 174, 224
MULTIPLE PAGE clause	39, 89, 127, 135
multiple reports	21, 41, 267
identical copies	271
multiple-choice entries	32, 172
MVS - see <i>Installation and Operation</i>	

N

negative values - see <i>SIGN</i>	
NEXT BODY GROUP - see <i>NEXT GROUP</i>	
NEXT DE OR CH GROUP - see <i>NEXT GROUP</i>	
NEXT GROUP clause	37, 133, 137
NEXT PAGE phrase	
of LINE	125, 135
of NEXT GROUP	37, 139
NO MULTIPLE PAGE clause	136
NO WRAP clause	234
non-hierarchical CONTROLS	73
non-REPORT SECTION SUM	192, 196
NONE option of multiple SOURCE	184
NOPF file handler	58, 281
NORMAL, with STYLE	186

O

OCCURS clause	26, 142 , 168, 184
with SUM	197
with VALUE	224
with VARYING	226

ON NEXT PAGE phrase - see <i>NEXT PAGE</i>	
OPEN statement	17, 246
optional entries - see <i>PRESENT</i>	
OS/390 - see <i>Installation and Operation</i>	
OS/VS COBOL	4
comparisons of formats:	
FILE SECTION and RD	47
PROCEDURE DIVISION	238
Report Groups	89
compatibility	4
<i>also at end of each section</i>	
migration from	299
OSVS precompiler option	63, 152, 206, 243
and COLUMN clause	107
and SOURCE SUM correlation	63
OTHER option of PRESENT WHEN	32, 172
OVERFLOW clause	19, 78

P

page advance processing - see <i>page-fit test</i>	
Page Buffer feature	41, 57, 248, 249
PAGE clause - see <i>PAGE LIMIT clause</i>	
PAGE FOOTING - see <i>PF groups</i>	
PAGE FOOTING totals	203
PAGE HEADING - see <i>PH groups</i>	
PAGE LIMIT clause	8, 81 , 248
PAGE option	
of CH group	38, 218
of PRESENT AFTER	158
page, regions of	83
PAGE-COUNTER	13, 149 , 244, 257
page-fit test	122, 127 , 138, 146, 168, 251
parameters to file handler	56
parameter to FUNCTION	114
PF groups	7, 17, 131, 140, 216
PH groups	7, 131, 140, 216
PICTURE clause	12, 36, 151 , 173, 179
with VALUE	223, 225
PLUS - see <i>relative forms</i>	
pre-break values of controls	20, 71
precision of totals	201, 206
precompiler	3
Preface	6
PRESENT AFTER clause	20, 156
PRESENT WHEN clause	14, 32, 162
effect on SUM	168, 199, 205

OTHER option	172
using total fields	205
with OCCURS	145
PRINT-SWITCH	243, 255
Printer Description File	187
PRNT file handler	54, 66, 248, 281
procedural references to Report Writer data-names	101
PROCEDURE DIVISION	16, 237
purpose of Report Writer	3

Q

qualification	
of CODE-VALUE	66
of control-id	68
of DETAIL group name	61
of LINE-COUNTER	134
of PAGE-COUNTER	149
of SET statement	249
of SOURCE operand	181
of total fields	192
of UPON operand	193
using control-id	72

R

railroad track, explanation	6
RANDOM PAGE feature	57, 249
RD entry	8, 19, 59
RDATE function	119
RECORD CONTAINS clause	54, 65
record description, after FD	54
RECORDING MODE	54, 57
with a file handler	58
reference modification	181
relative form of	
COLUMN	11, 105
LINE	11, 125
NEXT GROUP	138
relative subscripting	181
RELEASE - see SET	
REPEATED clause	28, 76, 175
repetition - see <i>OCCURS, REPEATED, multiple forms</i>	
REPLACE BY option of OVERFLOW	79
REPLACE statement	321
REPLACING option of COPY	321
REPORT as CONTROL operand	9, 68

Report Control Area	287
report files	47, 50
REPORT FOOTING - see <i>RF groups</i>	
Report Group Descriptions	61
report groups (see also <i>TYPE</i> clause)	4, 5
coding rules	98
definition	89
REPORT HEADING - see <i>RH groups</i>	
REPORT option of PRESENT AFTER	159
REPORT SECTION	8, 59
REPORT SECTION SUM	193
report-name - choice of	8
report-name – definition	53
REPORT-NUMBER	41, 271
reports - introduction to	4
REPORTS clause of FD	15, 19, 53
reserved words	323
RESET phrase of SUM	193, 205, 211
RF groups	7, 17, 140, 216
use for writing trailer	54
RH groups	7, 135, 140, 216
use for writing header	54
RIGHT option of COLUMN	11, 37
RMDATE function	119
rolling forward of totals	107, 112, 195, 199, 206, 241
ROUNDED phrase	12, 182
row totals	29, 198
run time messages	76, 79, 166, 175, 196, 206, 242, 252, 258
RYDATE function	119

S

SELECT...ASSIGN clause	15, 52
SET page buffer statements	41, 248
shorter forms of syntax	35
SIGN clause	179
size errors - see <i>OVERFLOW clause</i>	
snaking columns	252
snapshots of total field	203
sorting, via COBOL SORT	69
SOURCE clause	12, 181
as subject of SUM	191
OVERFLOW checks for expressions	79
referring to total field	203
SOURCE SUM correlation	63, 107, 196, 206, 207, 210, 241
SPECIAL-NAMES paragraph	65
spooling report data	19, 65, 271, 283

squeeze symbol "<"	36, 153
STATE function	120
STATEF function	120
STEP phrase	26, 143
STIME function	120
STYLE clause	186, 223
at FD level	56
at RD level	62
subheadings	158
subscripts - see <i>VARYING</i>	
subtotalling	72, 196, 203, 205, 206, 208 , 241
SUM	
as a term	31, 191
clause	15, 19, 25, 29, 191
from a different report	199
use of total fields	201
with PRESENT WHEN	168, 199, 205
SUM OVERFLOW clause	19, 78 , 196
sum-counter - see <i>total fields</i>	
summary reporting	6, 20, 178, 206, 207, 215, 240
SUPPRESS PRINTING statement	255
symbolic characters, as VALUE	223
Syntax summary	325

T

tables - see <i>OCCURS, SUM</i>	
TERMINATE statement	17, 257
with multiple reports	267
with total fields	206
TIME function	39, 120
TIME-OF-DAY special register	183
total fields	193
total fields, uses	201
totalling - see <i>SUM</i>	
totals only reports	207
truncation	
of expressions	79
of PICTURE	181
of line	167
of records	57
of totals	79, 193, 201
TYPE clause	7, 10, 21, 61, 214
TYPE clause of SELECT	55

U

UNLESS phrase of PRESENT	163
unprintable fields	107, 170, 187, 199, 202, 205
UPON phrase of INITIATE	53, 246
UPON phrase of SUM	193, 196, 209 , 210, 243
USAGE clause	221
USE BEFORE REPORTING directive	149, 206, 237, 240, 243, 255, 259
User-written extensions - see <i>Independent Report File Handlers, FUNCTION clause</i>	
USING phrase of MODE	56

V

VALUE clause	12, 223
as subject of SUM	193
variable number of repetitions	26
variable-length fields	36, 37, 103, 107, 110, 153
variable-length records	57
variable-position fields	77
VARYING clause	27, 226
verbs used in Report Writer	4, 237
vertical repetition - see <i>LINE, OCCURS</i>	
VM - see <i>Installation and Operation</i>	

W

WIDTH - see REPEATED clause, STEP	
wild cards, in COPY statement	321
WITH CODE - see CODE	
WITH PAGE BUFFER - see Page Buffer feature	
WITH RANDOM PAGE - see RANDOM PAGE feature	
WRAP clause	39, 77, 230
WRITE statements, explicit	54

Y

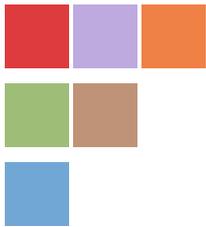
YDATE function	120
----------------	-----

Z

zero divide - see <i>OVERFLOW</i>	
ZIP function	121

etc

3800 - see laser printer	
< and > PICTURE symbols	36, 154



www.cobol-it.com

June, 2020

