COBOL REPORT WRITER PRECOMPILER

INSTALLATION AND OPERATION

for VSE

Program Number 5798-DYR

and Program Number 5798-DZX (Run Time Library only)

IBM Publication SC26-4864-00 with updates

On-Line version: cross references are yellow

Third Edition, May 2001

SPC Systems Ltd.
Wimbledon, London SW19 3PX
England.
Tel: (US) (206) 725-7431
Tel: (UK) +44-208-540-8409

www.spc-systems.com
info@spc-systems.com

Contents

Preface

This publication is intended for:

- technical planning and systems programming personnel engaged in the installation or customization of the COBOL Report Writer Precompiler,
- personnel who are writing JCL procedures to compile programs containing Report Writer, for use by application programmers,
- application programmers who need to compile COBOL programs containing Report Writer, or need additional information on the listings and other outputs produced by the precompiler.

This publication is designed to help you to:

- understand the basic functions and principles of operation of the COBOL Report Writer Precompiler, and its relationship to the *IBM VS COBOL II, IBM COBOL for VSE and DOS/VS compilers (Part 1), so that you will be able to make an appropriate choice of the options described in the remaining sections;
- plan for installing and customizing of the COBOL Report Writer Precompiler (Part 2);
- install and customize the COBOL Report Writer Pecompiler under VSE/ESA*, (Part 3) (referred to as ***VSE*** in the rest of this manual);
- precompile and compile a Report Writer program under VSE (Part 4);

You should already be familiar with VSE* Job Control Language. You will not require a detailed knowledge of either elementary COBOL or Report Writer to use this publication, but a knowledge of the requirements of the application programming functions at your installation is necessary in order to perform the customization tasks.

If your main concern is with the **language**, and how to code or understand a COBOL program incorporating Report Writer, you should consult the *Programmer's Manual*.

* *IBM*, *VSE* and VSE/ESA are trademarks of International Business Machines Corporation.

Related Publications

Precompiler

COBOL Report Writer Precompiler, Programmer's Manual, SC26-4301
(referred to henceforth as the *Programmer's Manual*)

VS COBOL II

VS COBOL II Application Programming: Language Reference, GC26-4695

VS COBOL II Installation and Customization for VSE, SC26-4696

VS COBOL II Application Programming Guide for VSE, SC26-4697

DOS/VS COBOL

IBM DOS *Full American Standard COBOL*, GC28-6394

IBM *VS COBOL for DOS/VSE*, GC26-3998

IBM DOS/VS COBOL Compiler and Library Programmer's Guide, SC28-6483

VSE

IBM Virtual Storage Extended, Advanced Functions, System Control Statements,
SC33-6198

1

Precompiler: General Information

This first part provides some basic information on the design objectives of the COBOL Report Writer Precompiler. It summarizes the COBOL language features and describes the basic principles of the precompiler, explaining its relationship to the COBOL compilers, and the inputs and outputs used in each step. By reading these sections, you will be better able to make the correct choice for the options that will be required when you install and customize this product.

1.1 Objectives

1.1.1 Purpose of COBOL Report Writer

COBOL Report Writer is a data-oriented addition to basic COBOL that greatly simplifies the production of all printed output. The language available through this Report Writer product contains the ANS-68 COBOL Report Writer supported by the DOS/VS COBOL compiler, together with the IBM, ANS-74 and ANS-85 extensions. The implementation covered by this publication also contains a large number of extensions that greatly expand the power and usability of the standard features.

The ANS-68 features cover, briefly, the following areas:

- Representation of the main components of the report in two-dimensional form in the DATA DIVISION by means of **LINE** and **NEXT GROUP** clauses (for vertical spacing) and **COLUMN** clause (for horizontal spacing),
- Automatic output of report lines to specified report file(s), controlled by **INITIATE**, **GENERATE**, and **TERMINATE** statements,
- Automatic storage of **SOURCE** fields in the report lines,
- Detection of the **page-full** condition and automatic generation of page headings and footings,
- Detection of **control breaks** and automatic generation of control headings and footings,
- Simple subtotalling, rolling forward and cross-footing of **totals**.

The extended Report Writer features cover the following areas:

- Rationalization of the syntax with more optional **abbreviations**,
- Automatic **repetition** vertically, horizontally, and in blocks,
- COBOL **conditions** in the REPORT SECTION to control the output of lines, or report items,
- **Subheadings** after page or control breaks,-
- Option to print **CONTROL HEADING groups at top of page**,
- Greatly extended functionality of the **SUM** feature,
- **Relative (floating) COLUMN** clause, plus **CENTER/RIGHT column** positioning,
- **Variable-length** fields (automatically trimmed),
- **Multiple COLUMN** and **LINE** clauses allowed in a single entry,
- **Arithmetic-expressions** allowed as SOURCE and SUM operands,
- Built-in and user-written **FUNCTION** facility,
- **Page Buffer** feature for generation of irregular page formats,
- **Multiple Report** facility,
- Direction of output through a built-in or user-written **file handler** to special devices or spooling software.

The ANS-85 features added in *Release 2* of the product were:

- **GLOBAL and EXTERNAL** report files,
- **GLOBAL reports**, and access to them from contained programs,
- Existing elements (e.g. SOURCE) extended to allow new **ANS-85** features.

The features added in *Release 3* of this product (the first for VSE) are:

- Use of compiler's **EXIT** feature,
- Generation of pure **SAA* COBOL** code,
- **No** dependence on **run time routines for DOS/VS COBOL** sources,
- In addition, an option to eliminate most dependence on run time routines for **new** programs by copying sources of COBOL run time routines as nested programs (**RTNEST** option),
- Many **new data clauses**: see *Programmer's Manual*,
- Option to skip the precompilation automatically when the source contains no Report Writer code (***CONTROL RW/NORW**),
- Option to show line numbers of intermediate source (**LGSEQ** option) for on-line debugging, plus other listing features,
- Automatic **skip-to-channel** feature,
- **DBCS** support,
- Amendments for the handling of listings from **Release 3.2** of VS COBOL II.

For additional information on the syntax and facilities provided within the language itself, you should refer to the *Programmer's Manual*.

1.1.2 Purpose of the Precompiler

This product gives you two different methods of processing a COBOL program containing Report Writer code. Both methods provide the same language features, because, from the top level, they use the same precompiler phases.

- Using the compiler's **EXIT** option.

  With this method, the precompiler runs under the control of the compiler. You use the IBM COBOL or VS COBOL II compiler as you would for a basic COBOL program, except that you include an **INEXIT(RW)** option. There is also a **PRTEXIT(RW)** option which modifies the compiler's listing by printing the original source code instead of the expanded code. Both can be permanently selected when you customize the compiler.

*SAA is a trademark of International Business Machines Corporation.

To specify them as parameters to the compiler, you code:

EXIT(INEXIT('parameters',RW),PRTEXIT(RW))

or, to use their abbreviated forms:

EX(INX('parameters',RW),PRTX(RW))

By this method, the compiler appears to handle Report Writer itself as a built-in part of COBOL.

If you need the compiler's **EXIT** option for another preprocessor, you can still use this method, because the precompiler has its own **EXIT** option, similar to the compiler's. This may also be permanently selected by customization (including any parameter strings). If you need the **EXIT** option for a third-party *librarian* product, you can use the **LIBEXIT** sub-parameter of either the precompiler's or the compiler's **EXIT** option for this purpose.

You will need a certain amount of extra virtual memory for the largest programs when you use the **EXIT** option, since the precompiler and its own data areas must be loaded in memory at the same time as the compiler's initial phase. However, the precompiler is deleted from memory during the principal compilation phases.

- Using the stand-alone precompiler.

  The alternative to using the **EXIT** option is to run the precompiler as a separate step. Here, the precompiler runs in "preprocessor mode". It scans the source program for any Report Writer elements, and converts them to basic COBOL, leaving the rest of the source program unchanged. The resultant *intermediate* source program is written to the SYSPCH. This may then be compiled normally as a second step.

  You must use the stand-alone precompiler if, for any reason, you need to access the intermediate source code.

1.1.3 Benefits

Whichever method you use, using a precompiler brings you these benefits:

- It enables the "higher-level" COBOL features to be enhanced without all the complications of installing a new compiler. (New releases of this product do not necessarily coincide with new releases of the compiler.)
- It makes it easier to provide good Programmer's documentation, because Report Writer is now far too rich to summarize in just one chapter of a COBOL language manual.
- It eases the debugging of Report Writer programs, because the generated COBOL code can be listed and looked at if required, or viewed via the on-line debugger.

1.2 Migration from DOS/VS COBOL to IBM COBOL

The precompiler enables you to use any current *IBM COBOL* to compile your source programs written for DOS/VS COBOL that incorporate Report Writer, **without needing to convert or re-write the Report Writer code**. The precompiler also enables you to continue to use Report Writer in **new** programs, with the additional benefit of a greatly enhanced set of features. All the ANS-85 features affecting Report Writer are supported. The additional ANS-85 Report Writer features are also supported, provided the **NOCMPR2** option is in effect.

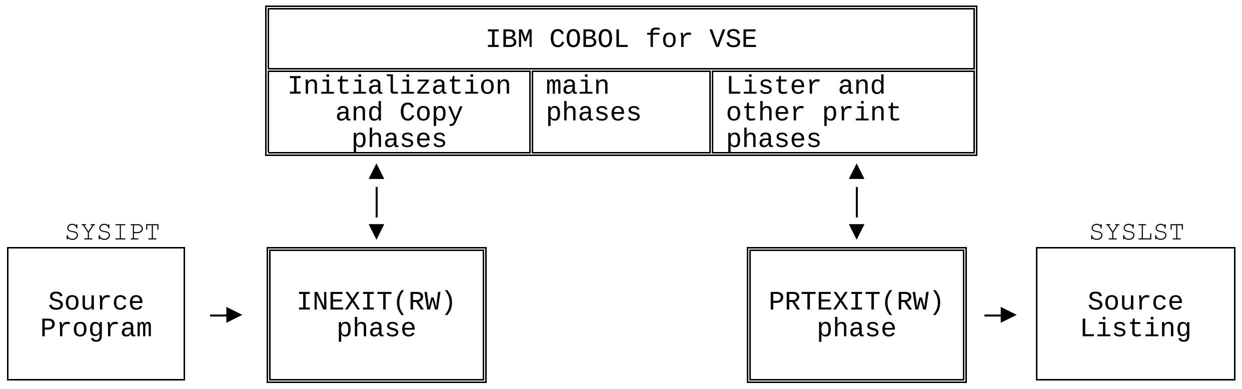
The precompiler processes only the Report Writer syntax in your program. Before attempting to precompile and compile it for the first time, you should first ensure that all the remaining (non-Report Writer) COBOL code in the program will be acceptable to the compiler.

Most DOS/VS COBOL Report Writer source programs are accepted completely unchanged by the precompiler. Where DOS/VS COBOL has allowed a "doubtful" or non-standard Report Writer construction, in the great majority of cases the precompiler issues a Warning message and still accepts the code. Generally speaking, the precompiler is stricter than the older compilers, so quite a number of Warning messages may be issued. Sometimes the message indicates a serious previously undetected flaw in the coding that must be attended to. Details of all these discrepancies and suggested means of avoiding them will be found in part 6 of the *Programmer's Manual*.

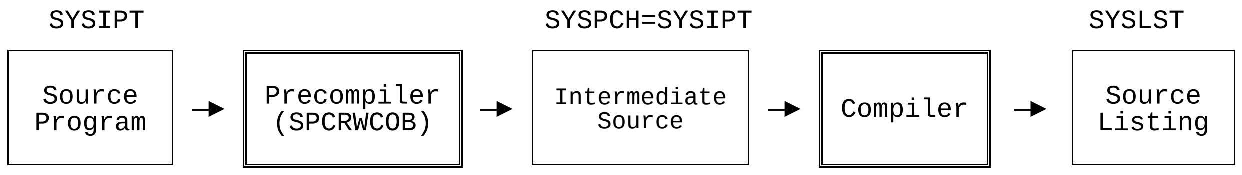
1.3 Precompiler System Overview

The diagrams on this page give you a pictorial view of how the precompiler operates.

Using INEXIT(RW),PRTEXIT(RW)



Using the Stand-alone Precompiler



1.4 Notes on Precompiler Operation

- Non-Report Writer Sources

  If the source program contains **no Report Writer** code, it is possible to **bypass the precompiler's conversion routines** by placing a ****CONTROL NORW** compiler-directing statement as the first or second line of the source (see 1.8.1 for details). This causes the precompiler to pass its input directly to its output and thus saves processing time, enabling you to use the **same JCL** for **all COBOL sources**. (Alternatively, you can specify that only those programs with a ****CONTROL RW** compiler-directing statement at the start of the source are to be precompiled.) If a non-Report Writer source escapes this filtering process and is unnecessarily precompiled, it emerges unchanged in the precompiler's output (apart from some comments inserted by he precompiler).

- Messages

  Embodied in the precompiler are a comprehensive range of error, warning and informational messages which are issued for every conceivable syntax error. An explanation of each precompiler message will be found in the *Programmer's Manual*.

  If the **FLAGSTD** option is specified, the precompiler will issue an informational FIPS-message against elements of the Report Writer code where appropriate.

- COPY books

  If your source program contains **COPY**, **BASIS** or **REPLACE** statements, the precompiler will expand them unless you specify the NOCOPY option. (Note that only the simple BASIS statement, without INSERT or DELETE is allowed.) Thus, your COPY books may contain Report Writer code.

- Virtual and disk memory

  The precompiler needs a minimum amount of virtual storage space for its own use, but also makes use of basic disk space to hold its tables and work areas. For this purpose it uses a data set **IJSYS11**. This may reside in VSAM-managed or basic disk space. The amount of GETVIS space used by the precompiler can be controlled using the **SIZE** option. This topic is covered fully in Part 2.

1.5 Purpose of PRTEXIT(RW)

If you use **PRTEXIT(RW)**, this routine is invoked by the compiler to print the whole of the source listing in a compact form that makes the source easy to understand and maintain. It embeds the **original** source program in the compiler listing and does not print the intermediate code (unless you specify **MGENER**). In addition, it copies the other parts of the listing (MAP, XREF etc.) and alters the line numbers so that they correspond to the original source. Any messages from the precompiler and the compiler are combined and printed as a single set. This is especially important because the precompiler relies on the compiler to report certain syntax errors in the REPORT SECTION. If the precompiler phase is successful, this does not guarantee that the original code is error free. For example, the validity of a data name coded in a *SOURCE* statement is not checked by the precompiler but by the compiler.

PRTEXIT(RW) conceals the distracting intermediate data definitions and procedural code. You do not need to "jump" from the original to the intermediate source listing to find an entry in the Cross Reference, Data Map, or Offset listing. The listing is presented in a manner close to that which you might have expected if there had not been a precompiler at all and the compiler had in fact handled the higher COBOL syntax itself. It is similar to the listing you are used to working with if you have used DOS/VS COBOL.

If **PRTEXIT(RW)** is not used, your listing is printed in two parts: the **original** source listing, printed by the precompiler, with any Report Writer error messages, and the **intermediate** source listing, printed by the compiler, together with any basic COBOL error messages and any additional compiler listing options. For technical reasons, the compiler options appear before any precompiler messages.

1.6 Options and Customization

A number of options are provided to control the precompilation and listing. Some are specific to the precompiler, while others, such as **ADV**, and **QUOTE/APOST** are shared by the precompiler and the compiler. If **INEXIT(RW)** is used, the precompiler will obtain the values of these shared options from the compiler and you do not need to specify them separately. All the precompiler options can be specified when the precompiler is customized. If the **INEXIT(RW)** method is **not** used, the shared options must also be specified in this way, because the precompiler then runs quite separately from the compiler.

The *Customization Routine* **INEXIT(RWCUS)** may be used to select or alter the default values of these options.

1.7 Run Time Library

A **run time library** is provided with the precompiler. However, Report Writer in principle do **not** depend on a run time system. These routines are needed only occasionally for the more advanced functions. A detailed description of this library will be found in 2.2.

1.8 Elements of Input Source

The purpose of this section is to describe how the precompiler handles some of the input source elements, apart from the Report Writer syntax itself, which is fully described in the *Programmer's Manual*.

1.8.1 Compiler-Directing Statements

The precompiler responds to certain compiler-directing statements. The following list shows the effects of each statement.

***CONTROL/*CBL**

***CONTROL (*CBL) SOURCE/NOSOURCE** are acted upon by the precompiler in producing its own listing. They are also passed to the compiler.

***CONTROL (*CBL) LIST/NOLIST** and **MAP/NOMAP** are **not** acted upon by the precompiler but are passed to the compiler. They will therefore be used by the compiler in suppressing parts of its own **LIST** and **MAP** listings, and this will be reflected in the final listing whether or not **PRTEXIT(RW)** is used.

CONTROL RW** and ***CONTROL NORW** are recognized **only** by the precompiler. (*CONTROL** may be written instead of ***CONTROL** so as not to cause a compiler error if you compile the source directly.) The directive must occupy the first or second line of the source. ***CONTROL RW** tells the precompiler to convert any Report Writer code in the source. ****CONTROL NORW** tells the precompiler that there is no Report Writer code in the program and that it may therefore **bypass** the precompiler and pass the **original** source directly to the compiler. If both forms of this statement are absent, the setting of the RW/NORW **option** (as customized or given explicitly in the PARM) is used.

BASIS, INSERT, DELETE

The precompiler recognizes the **BASIS** statement and will copy the source program specified. However, it does not recognize INSERT and DELETE statements.

CBL/PROCESS

The precompiler recognizes the **CBL** (or **PROCESS**) statement, processing any precompiler or *shared* options (see 2.3.3) and passing any compiler or *shared* options on to the compiler.

COPY

The precompiler processes this statement in all its forms, unless **NOCOPY** is in effect, when it is passed across unaltered. The **LANGLVL(1)** format of DOS/VS COBOL is not recognized. Copied text may itself contain COPY statements, up to six levels of nesting and **REPLACING** may be used at any level.

Lines containing the word **COPY**, up to the closing period, are passed to the compiler with an asterisk (*) in column 7. If COPY is not the first word in the line, the line is split into two lines to make it so.

In addition to the standard COPY statement, the precompiler allows the **wild card** combination **"??"** to represent "any non-null string", for example:

```
COPY...REPLACING ==0?? FILLER PIC ??.== BY == ==.
```

The wild cards may also appear in the replacing text, provided they are expected to be replaced in the same order, for example:

```
COPY...REPLACING ==WS-??-DATE PIC ??==
  BY  ==WS-??-DATE2 PICTURE ??==.
```

EJECT, SKIP1, SKIP2, SKIP3

These statements are passed to the compiler and therefore affect the printing of the source listing in the usual way. Blank lines and a slash ("/") in column 7 may be used for a similar purpose.

ENTER

This statement is passed unchanged to the compiler.

EXEC...END-EXEC

Any code between these two keywords is copied unchanged, thus allowing full use of DB2 and other COBOL extensions that use this format in their command language.

REPLACE

The precompiler processes this ANS-85 statement in all its forms, unless **NOCOPY** is in effect, in which case REPLACE statements are passed unchanged to the compiler. Processing of the source lines containing REPLACE is similar to that for **COPY** above, including its extensions. REPLACE **can** affect code brought in by a **COPY** (but not the COPY itself) and, if the COPY has a **REPLACING** phrase, the text is subject to change from both the REPLACING and the REPLACE (in that order).

SERVICE LABEL

This statement is not recognized and, like all Procedure Division items that are not specifically recognized by the precompiler, will be passed to the intermediate source unchanged.

SKIP1, SKIP2, SKIP3 - see *EJECT* above.

TITLE

This statement will be recognized and used by the precompiler in printing the page headings of the program source listings. It is also passed unchanged to the intermediate source.

USE

All USE statements are passed unchanged to the intermediate source, except for the **USE BEFORE REPORTING...** statements which are processed by the precompiler and deleted.

1.8.2 Sequence Numbers

Sequence numbers in columns 1 to 6 are passed unchanged by the precompiler to the compiler in any line that is altered or copied unchanged. The precompiler's own generated lines have blanks in these columns. The **NUMBER** option cannot be used with **PRTEXIT(RW)**.

1.8.3 Comment Lines

Comment lines, containing a **"/"** or **"*"** character in column 7, are ignored by the precompiler (apart from causing a page advance in the case of "/") and passed unchanged to the compiler.

Any character other than these and **"D"** and **space** is also assumed to be a **comment** and is passed across unchanged, thus allowing for other preprocessors that rely on a special character in column 7.

1.8.4 Debug Lines

The precompiler correctly processes debug lines (those with a **"D"** in column 7) in the following manner:

a. If **WITH DEBUGGING MODE** has been coded in the **SOURCE-COMPUTER** entry, debug lines are treated as normal lines, each **"D"** is removed, and the lines processed.

b. If WITH DEBUGGING MODE is **not** found in the SOURCE-COMPUTER entry, debug lines are treated as comment lines and are passed to the intermediate source unchanged, where it is the compiler's responsibility to ignore them.

1.8.5 Identification Columns

The contents of columns 73-80 are retained by the precompiler in any line that is altered or copied unchanged. In precompiler generated source lines these columns contain the characters:

" RWnnnn+" or **" RWnnnn="**

where **nnnn** is the version/release number.

1.8.6 Nested and Batched Programs

The input source may have *ANS-85* **contained programs** and *"batched"* programs (consisting of non-nested programs each terminated by an **END PROGRAM** header).

1.9 Intermediate Source

If you use **INEXIT(RW)**, there is no physical output from the precompiler and the only sight you may have of the intermediate (that is, converted) code is in the compiler listing (or the listing from **PRTEXIT(RW)** if you specify **MGENER**).

The intermediate source is produced by the stand-alone precompiler. It consists of the original source program, suitably modified and with additional generated COBOL code. Although the code is clear and modular in construction, it was designed primarily to be compiled efficiently, rather than to be inspected and understood. If you make any permanent alterations to the intermediate source, you and subsequent users will be unable to repeat the precompilation without losing the changes. Alterations to the intermediate source are therefore not recommended under any circumstances.

1.10 Source Listings

If **PRTEXIT(RW)** is **not** specified, you will obtain the following listings in tandem:

- Precompiler options in effect,
- Precompiler's listing of the **original** source,
- Compiler’s front sheet showing options in effect,
- Any precompiler messages,
- Compiler’s normal listing of the **intermediate** source, depending on the listing options requested.

If **PRTEXIT(RW)** is specified, you will obtain a **single** unified listing, with the following features:

- In the front sheet, the compiler's and the precompiler's options are shown side-by-side.
- The source listing (if **SOURCE** is in effect) shows the **original** source program. If the **MGENER** option is specified, precompiler generated source lines are shown merged into the original source.
- Messages from both the precompiler and the compiler are combined into a single set, and printed according to the **FLAG** option. If the second operand of **FLAG** is in effect, they are also embedded in the source listing. Some compiler messages that refer to unseen precompiler generated source lines are not embedded but appear only at the end. An **"*"** against a message at the end identifies messages that were embedded. The messages displayed may include any produced as a result of the **FLAGSTD**, **FLAGSAA**, or **FLAGMIG** options.
- The compiler's **embedded XREF** and **embedded MAP**, if specified, are placed correctly against the lines to which they refer.
- Sequence numbers of the corresponding intermediate source lines are printed over the original sequence numbers, if **LGSEQ** is in effect.
- Additional features of the compiler listing will, if specified, be modified by **PRTEXIT(RW)** as follows:

  VBREF causes the *Cross Reference of Verbs* to be printed, with line numbers changed to refer to the **original** listing.

  XREF causes the *Cross Reference* listing to be printed both embedded in the source and as a separate listing, with the line numbers changed to show the line numbers of the **original** source listing.

  MAP causes the *Data Division map* to be printed with its line numbers changed to show the line numbers of the **original** source listing.

OFFSET causes the *Procedure Division offset* summary to be printed together with Global Tables, Literal Pools, etc. Line numbers are changed to show the line numbers of the **original** source listing.

LIST causes the compiler's assembler-language listing to be printed with line numbers changed to show the line numbers of the **original** source listing.

The **summary and statistics** are also printed in suitably modified form.

The illusion that the compiler performed the Report Writer processing itself is occasionally broken by certain features which may nevertheless prove useful:

a. **PRTEXIT(RW)** does **not** suppress generated data names and procedure names and a number of names with the prefix **R- -** usually appear, many bearing the same sequence number that coincides with an RD entry or a report group. These items may be ignored if not relevant.

b. The **XREF** and **MAP** will not show the standard names for the Report Writer locations that are reserved words, namely **PAGE-COUNTER** and **LINE-COUNTER**. You must therefore look them up under their internal names, **R--rPCT** and **R--rLCT** (r = report number).

c. The **XREF** will **not** show DETAIL report group names used in a GENERATE, or report names used in an INITIATE, GENERATE or TERMINATE, or any of Report Writer's internal references (such as SUM...UPON or COUNT).

d. The **XREF** and **MAP** will **not** show the RD entry, and the FD entries for report files that use a file handler will have been re-located.

e. In the **VBREF**, **OFFSET**, and **LIST**, any INITIATE, GENERATE, or TERMINATE statements will not appear as such but as *PERFORM* statements. Report Writer **SET** and **SUPPRESS** statements will appear as MOVE statements. In addition, the precompiler-generated statements will have been taken into account in the VBREF. **LIST** always shows the whole of the Procedure Division, corresponding to the statements in the intermediate source.

f. If **RTNEST** is in effect, the locations belonging to the run time routines appear in any VBREF, XREF, MAP, OFFSET, and LIST.

g. If **TERM** is in effect, the line numbers displayed by the compiler as those of the intermediate, not the original, source.

1.11 Return Codes

The return code from the precompiler depends on the **highest severity level** of the precompiler messages using the same convention as the compiler. If **INEXIT(RW)** is used, the return code is the higher of the return codes from the precompiler and the compiler.

1.12 Debug

The **TEST** option can be used to enable on-line debugging in the usual way. Since the compiler sets up the debug information using the *intermediate* source as a reference, this will be the source shown on your terminal, whatever other options you specify. If the debugger reaches the point where an INITIATE, GENERATE, or TERMINATE statement had been coded, you will see a PERFORM statement. If you allow the debugger to execute the PERFORM you will step through the logic of the Report Writer statement. However, if you do not suspect any problems with the Report Writer logic, you can use break-points to proceed directly to other parts of the program, avoiding the Report Writer code. The line numbers of the intermediate source will not be the same as the original line numbers, but it is possible to perform most debugging operations by referring to *data-names* and *procedure-names* (which are not changed by the precompiler). If you need to use line numbers, you should either obtain a listing of the **intermediate** source, by specifying **NOPRTEXIT** or by using the stand-alone precompiler, or you should use the **LGSEQ** option which prints the compiler's line numbers against the original source.

1.13 Output from Report Writer Programs

1.13.1 Basic Printing

The normal output from a Report Writer file is a basic SAM file produced by a series of generated COBOL WRITE statements. The block size, logical record length and organization are therefore established from the **FD** clauses, supplemented by JCL. However, note that some features cause a *report file handler* to be used instead of generated WRITEs. These are listed under 2.1.1 together with some restrictions that result from using a COBOL file handler.

1.13.2 Special Printing

If the output device is not a regular printer, and needs special codes or control characters, or special software routines, output can be generated for it using a special *user-written file handler*. These are fully described in the *Programmer's Manual*. Even if the output is to a regular printer, a file handler may be used to achieve a particular technical objective, such as the use of printer channels (see **Appendix D**), output from a *modular system*, or output without page feeds (see **Appendix A** and *Programmer's Manual*).

1.13.3 Special Effects

The **STYLE** clause, by which special printer effects can be introduced, such as **UNDERLINE** and **HIGHLIGHT**, is described in the *Programmer's Manual*. The available printer TYPEs together with their available STYLEs are listed below in (See **Appendix E**).

2

Planning and Preparation for Installation

This part describes the minimum hardware and software requirements for the installation and use of the *COBOL Report Writer Precompiler*. It also describes the options available for customization and the planning you should perform before installing the product. If you intend to install the run time system only, you should read only section (see 2.2).

2.1 Requirements for Precompiler

2.1.1 Minimum Hardware and Software Requirements

This product is designed to run on an IBM System 3000-series, 9000-series, 2000-series, or any machine that supports VSE/ESA. Either *IBM COBOL for VSE*, or VS COBOL II, Release 3.0 or later, must be available.

2.1.2 Size and Memory Requirements

Apart from the control routine of the stand-alone precompiler, the precompiler runs entirely in GETVIS space below the 16-megabyte line. SIZE=SPCRWCOB should be coded in the JCL when the stand-alone precompiler is used. In calculating the size of the partition required, you need to add the following to the maximum memory required by the compiler:

Fixed Memory Requirements	Tally Column
The size of the largest phase, plus any dynamically loaded routines, held in memory at any one time is	**300K**
If PRTEXIT is specified, add for that routine:	**30K**
Since the precompiler is written in COBOL you must allow for the size of the general and VSE batch COBPACKs:	**......**
The precompiler also needs a minimum quota of GETVIS space for its own work areas:	**21K**

Variable Memory Requirements

The precompiler will also use GETVIS space above its minimum requirement (see above) when this is available. However, it can always run (albeit more slowly) with this minimum even for the largest sources by writing data out to disk (see IJSYS11 below). The amount of additional GETVIS it will take to avoid use of the disk is governed by the following considerations:

- The stand-alone precompiler will allocate itself GETVIS up to the size of the partition. This is usually much less than the GETVIS needed by the compiler.
- The **INEXIT(RW)** routine allocates a certain fraction (currently 3/8) of the available GETVIS, leaving the lion's share to the compiler. It frees most of this space before the compiler begins the compilation but it is difficult to predict how much GETVIS will be needed by the compiler.
- If a **SIZE** option is coded, **INEXIT(RW)** will ensure that the specified amount of space is available for the compiler. For example, if you code

SIZE(1000K)

you can be sure that at least 1000K will be available for the compiler's work space.

- If the compiler terminates with the message:

  '**IGYxx5062-U** There was insufficient storage for compiler processing. The "SIZE" compiler option value (and/or the PARTITION GETVIS size) should be increased.'

  you should try specifying a SIZE option as above, increasing the amount of GETVIS available to the compiler in stages (and reducing the amount allocated to the precompiler). If you increase the SIZE too far, you may obtain one of the messages:

  RW-PS01:MINIMUM GETVIS NOT AVAILABLE FOR PRECOMPILER

  RW-PM01:ERROR IN LOADING PHASE SPCHxxxx

  indicating that the remaining GETVIS is too little for the precompiler. Even if the precompiler runs, you may find that the precompilation is slow because of too much disk swapping. In these cases, your choice of action is:

  a. You can increase the partition size.

  b. You can try using the stand-alone precompiler. This will also enable you to determine the size of the inter-mediate source and check that the partition is large enough for the compiler to compile the source by itself. Note that there is no upper limit to the amount of GETVIS required by the compiler.

  c. If you are successful in using the stand-alone precompiler, it may be that your COBPACKs contain too many seldom-used routines and could be tuned.

  d. Placing the compiler or the COBPACKs, or both, in shared storage will of course reduce the amount of partition GETVIS required. You cannot place any part of the precompiler in shared storage.

- If you estimate the size of the partition needed for compilations on the basis of the size of your source programs, note that the Report Writer programs are effectively larger (by between 20% and 200%, average 60%) after being processed by the precompiler.

2.1.3 Data Set Requirements

The precompiler requires the following data sets:

- SYSIPT (input source)
- SYSLST (output listing)
- SYSPCH (stand-alone precompiler only)
- The working data set **IJSYS11**
- The source library, if COPY statements are present.

2.2 Requirements for Run Time Library

2.2.1 What Run Time Services are Required?

Run time routines are segments of code which are pre-written and brought in by a generated **CALL** rather than generated as in-line code. These routines are used only occasionally to perform certain more complex functions that cannot easily be generated as in-line code. If the option **NOXCAL** is specified, **no** run time routines will be invoked by an unchanged DOS/VS COBOL program. Certain additional run time routines (*file handlers* and *FUNCTION* routines) may be written by the user.

The names and functions of the run time routines will be found in **Appendix A**. **Appendix B** lists the language features or options that cause a run time routine to be used. Note that a few of the routines are written in **Assembler** rather than COBOL, so, if you intend to maintain a "COBOL-only" system, you may wish to avoid the few indicated language elements or options that cause them to be invoked.

You should also refer to **Appendix C** to understand the important topic of how **CONTROLS** are implemented.

2.2.2 How Run Time Routines are Incorporated

The way that **COBOL** run time routines are incorporated into your program depends on your use of the **RTNEST** option, as follows:

1. Using **RTNEST**

   The routines are placed in the program in **source** form as nested programs (an ANS-85 feature). Nested programs are incorporated by means of a **COBOL** COPY into the **outer** (or only) program of a nested structure. The **SUPPRESS** option of COPY is used so that they do not appear in the program listing. (Hence it is advisable to keep the listing of the entire COBOL library, which you normally receive if you compile them during installation.) Source modules are supplied in two libraries which are alike except that one uses **QUOTE** and the other **APOST**.

   Advantages of using **RTNEST** are:

   a. You do not need to remember to include parts of the run time library when you want to transfer the programs to a different computer system.

   b. You guarantee that the run time routines are compiled with the same compiler options (**RESIDENT/NORESIDENT, RENT/NORENT** etc.) as the program itself.

   c. You need not worry about the effect of the **DYNAM** option as nested programs are always called statically.

2. Using **NORTNEST**

   The routines are incorporated in **object** form. Since they are invoked by a generated COBOL **CALL**, there are two ways they can be brought in, depending on your choice of **DYNAM** or **NODYNAM** when you compile the application program. If **NODYNAM** is in effect, they will incorporated by the **link editor**. If **DYNAM** is in effect, they will be called **dynamically** at run time. In case **DYNAM** may be required, **PHASE** versions of each run time routine are provided in the run time library.

   Some routines are **always called dynamically**, because they are invoked via *CALL identifier*. They are as follows:

   a. All **file handlers**,

   b. The **Page Buffer** handlers **CXRPBFnn**, invoked whenever the **WITH PAGE BUFFER** clause is used,

   c. The **STYLE** handler **CXRSTYLE**, invoked whenever the **STYLE** clause is used.

   Because of these dynamic CALLs, it is necessary to specify the **RESIDENT** compiler option if any of these features are used. The supplied OBJECT versions of the COBOL run time routines were compiled with the **RESIDENT** option, so, if the **NORESIDENT** option is to used, the COBOL routines must be re-compiled at installation time.

   Advantages of **NORTNEST** are:

   a. You need not worry about the use of *Assembler* run time routines (listed in the previous section).

   b. Nested programs are **not** currently an **SAA** COBOL feature.

   c. They eliminate the overhead of repeatedly re-compiling the run time routines and reduce the size of the main object module.

   d. They can be shared (by use of the **RENT** option) between several run units.

   e. Only this method works with the **CMPR2** option, because **CMPR2** does not allow nested programs.

Routines written in *Assembler* are always incorporated in object form.

2.3 Preparing to Customize

2.3.1 Why Customize?

The precompiler is delivered with each of the available options set to its default value, indicated by underlined choices in 2.3.3 below. If you want to change any of these defaults, you must do the **Customization Program**. You can repeat it at any time if you decide to alter your installation defaults. You may wish to make modifications for any of the following reasons:

- You may need to ensure that stand-alone precompiler's defaults agree with those you established when you customized the compiler, for example in the use of **APOST** or **QUOTE**. This is not necessary if you use **INEXIT(RW)** as this obtains these defaults from the compiler.
- There may be a constant requirement among applications programmers for certain features such as **FMODE** or **PPSNS**. You may wish to pre-set the default values of these options so that the programmers are certain to use them as standard.
- There may be insufficient room in the JCL PARM string for certain common options.

If you will be using **INEXIT(RW)**, you need only worry about the options marked *precompiler only*, since the **compiler's** default values are used for all the options shared with the compiler (QUOTE, ADV, etc.).

2.3.2 How Options Control the Precompilation

The precompiler's options are described in the section that follows. Options may be specific to the precompiler or they may be common to the precompiler and the compiler, in which case they take exactly the same form as the standard compiler options.

The options **specific to the precompiler** are:

List A: COPY, CTRLEN, EXIT, FMODE, LGSEQ, MGENER, MONIT, OSVS, PPSNS, RTNEST, RW, XCAL

(note that a different EXIT parameter is also used by the compiler)

The options **shared by the precompiler and the compiler** are:

List B: ADV, CMPR2, DBCS, FLAG, FLAGSTD, LANGUAGE, LINECOUNT, QUOTE/APOST, SEQUENCE, SIZE, SOURCE, SPACE, TERM

There are **two** ways by which options may be specified:

1. By customizing them permanently.

   If you are using **INEXIT(RW)**, you need do this only for options which are specific to the precompiler *(List A)*. The shared compiler options *(List B)* will be obtained from the compiler whatever value you customize. If you will be using the stand-alone precompiler you should ensure that the settings of shared options *(List B)* agree with those you chose when you customized the compiler.

2. By coding them in the **PARM** in the JCL.

   If you are using **INEXIT(RW)**, options specific to the precompiler (List A) **must** be placed in the 'parameter string' of the INEXIT, for example:

   ```
   EXIT(INEXIT(' 'COPY,NOOSVS' ',RW))
   ```

   Note that the apostrophes have been doubled on the assumption that your entire PARM string is enclosed in apostrophes.

   The PRTEXIT does not take a parameter string. Even if the options apply chiefly to the listing, they should be coded with the INEXIT.

   If you are using the stand-alone precompiler, the options are placed in the PARM to phase SPCRWCOB.

Shared options *(List B)* are placed in the main PARM where they will be picked up both by the precompiler and (if you are using **INEXIT(RW)**) by the compiler, for example:

```
PARM='QUOTE,FLAG(I,W),EXIT(INEXIT(' 'OSVS' ',RW))'
```

Common options which have been customized for the compiler will also be picked up by the precompiler if you use **INEXIT(RW)**. Hence you can customize all the options you will need regularly and avoid exceeding the maximum size for the PARM.

The precompiler accepts the same **abbreviated** keywords as the compiler. Thus **F** may be coded for **FLAG**, **LC** for **LINECOUNT**, and so on. Keywords specific to the precompiler have no abbreviations.

2.3.3 Meanings of the Options

The following list explains each option. Alongside each keyword you will see the phase or phases it applies to (**precompiler only** or **shared**). The supplied default option value is **underlined** in each case. Note that the default setting of shared options is always the same as the default for the compiler. You will need to refer back to this section later when you read the sections describing customization (see Part 3) and operation (see Part 4). In those parts you will be shown exactly how and where to code the parameters.

```
┌─ ADV ──────────────────────┐
│ ►►─────┬─ADV──┬─►◄         │
│        └NOADV─┘            │
└────────────────────────────┘
```

shared

ADV instructs the precompiler and compiler to reserve an extra byte at the start of each report file record for the carriage control character.

NOADV states that the first byte of each report file record, as defined in the program, is set aside by the Programmer for this purpose.

APOST – see QUOTE

```
┌─ CMPR2 ────────────────────┐
│ ►►─────┬─CMPR2───┬─►◄      │
│        └NOCMPR2──┘         │
└────────────────────────────┘
```

shared

The option **CMPR2** modifies the code generated by the precompiler so that it conforms with the requirements of the compiler CMPR2 option.

```
┌─ COPY ─────────────────────┐
│ ►►─────┬─COPY───┬─►◄       │
│        └NOCOPY──┘          │
└────────────────────────────┘
```

precompiler only

COPY instructs the precompiler itself to process any **COPY** and **REPLACE** statements in the source program. If it is specified, the **COPY** statements and their expansions (unless **SUPPRESS** is specified in the COPY statement) are then printed in the source listing and the compiler's source input will contain no COPY statements. Similarly, **REPLACE** statements are processed and the results **after** replacement are printed in the listing. This option is required if any of your COPY books contain any Report Writer statements, or if a REPLACING statement affects any Report Writer statements.

NOCOPY prevents the processing of COPY and REPLACE statements, leaving this task to the compiler, if necessary. NOCOPY differs from **NOLIB** in that it affects the precompiler only. If NOCOPY is specified, **PRTEXIT(RW)** should not be used.

```
┌─ CTRLEN ─────────────────────────┐
│ ►►─CTRLEN (─┬integer──┬─►◄       │
│             └───80────┘          │
└──────────────────────────────────┘
```

precompiler only

CTRLEN gives the size in bytes of the largest Data Division item that will be used in the **CONTROL(S)** clause of a Report Description. This option is not relevant unless **NOXCAL** is also specified (see below). If in doubt, you may give a value of up to 256 for this option.

The precompiler allocates for each CONTROL item (other than FINAL) a **saved control location** which holds the previous contents of the control. This is used by Report Writer both to check for control breaks and also to restore controls to their previous values during the processing of CONTROL FOOTING report groups.

```
┌─ DBCS ────────────────────────────┐
│ ►►──────┬─DBCS───┬──►◄            │           shared
│         └─NODBCS─┘                │
└───────────────────────────────────┘
```

DBCS tells the precompiler and compiler that there may be Double Byte Character Set literals in the source, so that the **Shift Out** and **Shift In** characters will be recognized as such.

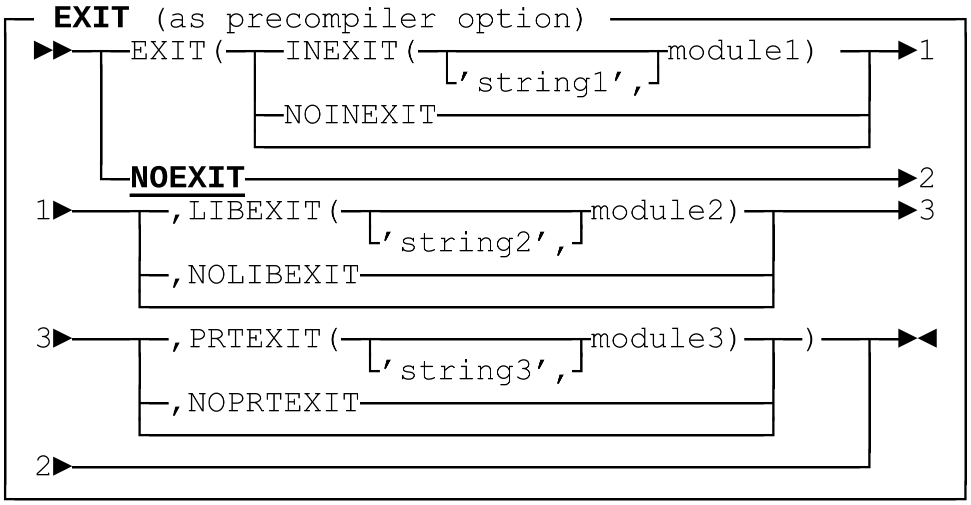
```
┌─ EXIT (as compiler option) ───────────────────────┐
│ ►►──┬─EXIT(─┬─INEXIT(─┬─────────────┬RW) ───┬──►1 │      compiler only
│     │       │         └'options',───┘       │     │
│     │       ├─NOINEXIT──────────────────────┤     │
│     │       └───────────────────────────────┘     │
│     │                                             │
│     └─NOEXIT──────────────────────────────────►2  │
│ 1►─────────┬─,LIBEXIT(─┬───────────┬module)─┬──►3 │
│            │           └'string',──┘        │     │
│            ├─,NOLIBEXIT─────────────────────┤     │
│            └────────────────────────────────┘     │
│ 3►─────────┬─,PRTEXIT(RW)┬──────────────)───┬──►◄ │
│            ├─,NOPRTEXIT──┤                  │     │
│            └─────────────┘                  │     │
│ 2►──────────────────────────────────────────┘     │
└───────────────────────────────────────────────────┘
```

This is the option that enables the precompiler to run under the control of the compiler. Abbreviations are: **EX**, **INX**, **LIBX**, **PRTX**.

The 'options' represent any string of precompiler-specific options, as described here. Apostrophes must be doubled if the main PARM is enclosed in apostrophes. Commas in the above syntax are optional.

LIBEXIT cannot be used to invoke the precompiler, and it is included here for completeness. If you are using a third-party librarian product that uses a LIBEXIT, you can put its name here and specify **NOCOPY**, in which case the compiler will be given the library expansions, or you can put it in the precompiler's own LIBEXIT slot (see next).

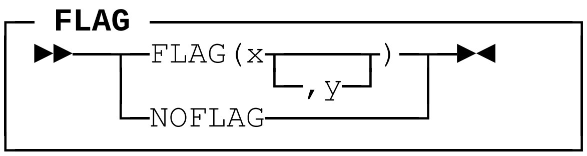
PRTEXIT(RW) is necessary to obtain a single compact source listing. It is required if you specify the options **LGSEQ** or **MGENER**, or if you specify **FLAG** with a second operand, or **SEQUENCE**, and you want the precompiler listing to show the effect of these. (See 1.5 for an explanation of the benefits of **PRTEXIT(RW)**.)



precompiler only

This option is similar in format to the compiler's EXIT option, except that quotes may be used instead of apostrophes. It is provided for those who already need the EXIT option for some other purpose (another preprocessor or a librarian utility) other than for Report Writer. It can be used both by **INEXIT(RW)** and by the stand-alone precompiler. The effects of INEXIT, LIBEXIT, and PRTEXIT are exactly as described in the *Application Programming Guide for VSE*. The LIBEXIT is not used if NOCOPY is in effect. If you want LIBEXIT to take effect with NOCOPY, you should place it within the compiler's EXIT option.

This entire EXIT option can in turn be coded as one of the options'of the *'string'* passed to the precompiler via the compiler's *INEXIT('string',RW)* option, thus *nesting* the EXIT options. To avoid a long and complex PARM 'string' in your JCL, this, like all the precompiler options, may be *customized*. Unlike the compiler's EXIT option, the precompiler also allows the three parameter 'strings' to specified on customization, with a maximum of 64 characters each.



shared

FLAG controls the printing of precompiler and compiler messages. See your *Application Programming Guide for VSE* for full details. The default value is **FLAG(I)**.

If the second operand is present, **PRTEXIT(RW)** is required if precompiler messages are also to be embedded. **PRTEXIT(RW)** collects together the messages from both the precompiler and compiler, merges them in sequence and displays them embedded in the source, according to the second operand, and at the end of the listing, according to the first operand. The second severity level must be higher than or equal to the first severity level. For example, **FLAG(I,E)** will print **all** messages at the end and only level **E**, **S**, or **U** messages embedded.

```
┌─ FLAGSTD ─────────────────────────────────────┐
│ ►►──┬─FLAG(x─┬────┬─┬────┬─)─┬─►◄             │       shared
│     │        └─yy─┘ └─,o─┘   │                │
│     └─NOFLAGSTD──────────────┘                │
└───────────────────────────────────────────────┘
```

FLAGSTD provides informational messages about the conformance of your program to the Standard. See your *Application Programming Guide* for full details. The precompiler provides additional FIPS messages- relating to the Report Writer syntax. All IBM and SPC extensions to the ANS-85 syntax are flagged as "*NONCONFORMING NONSTANDARD*" and whatever level is implied by the first character, all clauses and statements specific to Report Writer are flagged as "*NONCONFORMING STANDARD*". If **O** is specified, all obsolete language features held over from either the ANS-68 or ANS-74 standard are flagged as "*OBSOLETE*". The FLAGSTD option also has its standard effect on the compiler.

Note: If you require Report Writer FIPS messages but you do **not** want the compiler messages, you may place the FLAGSTD option in the 'parameter string' of the INEXIT instead of in the main PARM. For example, under VSE/ESA, to obtain just details of extensions to Report Writer, code:

PARM='EXIT(INEXIT(' 'FLAGSTD(H)' ',RW))'

```
┌─ FMODE ────────────────────────┐
│ ►►─FMODE(mode)────────►◄       │          precompiler only
└────────────────────────────────┘
```

This option can be used to **redirect** all the output from the reports in a program through a specified *file handler*, in order to give it special treatment or to direct it to a special output medium. If **FMODE** is coded, every report file in the program that does not already have a **MODE** clause in its **SELECT** statement is treated as though **MODE IS mode** had been coded. **Mode** may be any alphanumeric string of up to four characters.

```
┌─ LANGUAGE ─────────────────────────┐
│ ►►─LANGUAGE(─┬─UE─┬─)─►◄           │          shared
│              └─EN─┘                │
└────────────────────────────────────┘
```

This option chooses whether the English text printed by the precompiler should be only in upper case (**EN**) or whether lower-case is acceptable (**UE**). If any other value is used by the compiler, the precompiler assumes **EN**.

```
┌─ LGSEQ ─────────────────────────────┐
│ ►►──────┬─LGSEQ───┬─►◄              │
│         └NOLGSEQ──┘                 │
└─────────────────────────────────────┘
```

precompiler only

LGSEQ causes **PRTEXIT(RW)** to place sequence number of the corresponding intermediate source line in **columns 1 through 6** of each original source line in the source listing. (The actual contents of columns 1 through 6 in the input source are not shown but are of course left unchanged in the input source.) Thus you obtain all of the **compiler's source line numbers** (except for precompiler-generated lines) and this option saves you the need to print and save the intermediate listing in the cases such as the following:

a. If a *compiler-generated run time message* is issued from your program, you can locate the source line that caused it.

b. If you wish to refer to line numbers during **on-line debug**, you can find them in the original source listing.

```
┌─ LINECOUNT ───────────────────────────────┐
│ ►►─LINECOUNT(┬integer┬─►◄                 │
│              └──60───┘                    │
└───────────────────────────────────────────┘
```

shared

LINECOUNT gives the maximum number of lines per page for all the precompiler's and compiler's listings. *Integer* must be at least **4**.

```
┌─ MGENER ──────────────────────────────┐
│ ►►──────┬─MGENER──┬─►◄                │
│         └NOMGENER─┘                   │
└───────────────────────────────────────┘
```

precompiler only

MGENER causes generated source lines to be printed embedded in the original source. Source lines that have not changed are listed once only. Additional source lines generated by the precompiler are indicated by the following characters in the identification columns:

"RWnnnn+" (nnnn = version/release number)

Each altered line is followed by the new line containing:

"RWnnnn="

in these columns. **NOMGENER** suppresses this function.

```
 MONIT
►►─┬MONIT(integer┬─►◄                    precompiler only
   └NOMONIT──────┘
```

The **MONIT** option is reserved for program maintenance by, or under the direction, of your support center.

```
 OSVS
►►──────┬─OSVS──┬─►◄                     precompiler only
        └NOOSVS─┘
```

OSVS specifies that the DOS/VS COBOL variants of the Report Writer semantics are to be used wherever they differ from the standard used in the precompiler. It should be specified for the migration of DOS/VS Report Writer programs. The effects of specifying OSVS are:

a. *SOURCE SUM correlation* will assumed to be in effect for any SUM clauses, causing them to be interpreted according to the ANS-68 rules, rather than ANS-85. It may be overridden in a Report Description entry in the source program by means of the **ALLOW NO SOURCE SUM CORR** clause. Further information will be found in the *Programmer's Manual* under *ALLOW clause*.

b. Any subtotalling is performed **after** the printing of any PAGE FOOTING and/or PAGE HEADING groups and **after** the execution of any associated USE BEFORE REPORTING Declarative section.

c. An entry with a **PICTURE** clause but **no data-name or COLUMN** clause is **not** printed.

d. The positioning of **relative** REPORT HEADING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING groups is one line **lower** than when the option is not specified.

NOOSVS produces the opposite result for each of these items. In particular, the ANS-85 rules for the **SUM** clause will apply, with no checking for correlation of SOURCE and SUM entries unless **ALLOW SOURCE SUM CORR** is coded in the source program.

For further details, refer to the *Programmer's Manual*.

```
 PPSNS
►►─PPSNS(┬integer┬┬────┬)──►◄            precompiler only
         └──132──┘└─/F─┘
```

The **PPSNS** option gives the default value of *LINE LIMIT* to be assumed for any RD entry that has no *LINE LIMIT* clause. In other words, it specifies the highest value a report COLUMN will be allowed to attain. For details of the LINE LIMIT clause, refer to the *Programmer's Manual*.

If the optional **/F** is coded, the value is taken as the default value of the **RECORD CONTAINS** (**plus 1** if **NOADV** is in effect). This has the additional effect of **forcing** the logical record length of the given value (as opposed to setting a **maximum** value). Use this option if it is essential for the print files to have a logical record length of a certain value even when some reports never use that many columns' width. For example, to force a record length of 133, code **PPSNS(132/F)**.

```
┌─ QUOTE/APOST ─────────────────┐
│►►─────┬─QUOTE─┬──►◄           │
│       └─APOST─┘               │
└───────────────────────────────┘
```

shared

This option tells the precompiler which delimiter it should use for **generated** non-numeric literals and has its usual effect on the compiler. The precompiler will accept either delimiter in the **input** source, regardless of this option.

```
┌─ RTNEST ──────────────────────┐
│►►─────┬─RTNEST───┬──►◄        │
│       └─NORTNEST─┘            │
└───────────────────────────────┘
```

precompiler only

This option causes the precompiler to **copy any COBOL run time routines into the intermediate source as nested programs**, rather than requiring them to be linked in or loaded at run time. This option enables you to generate self-contained "pure" sources that do not have "hidden calls" to run time routines that might be overlooked, in the occasional instances when a run time routine is needed. It also has the advantage that calls to the precompiler's run time routines are not affected by your choice of **DYNAM/NODYNAM**. If **RTNEST** is present, **LIB** and **NOCMPR2** must be in effect.

Of course, *Assembler* routines cannot be included as nested programs and are unaffected by this option. However, as indicated earlier (see 2.2.1), use of these can be avoided in all but some exceptional cases.

```
┌─ RW ──────────────────────────┐
│►►─────┬─RW───┬──►◄            │
│       └─NORW─┘                │
└───────────────────────────────┘
```

precompiler only

This option gives the default action if the source contains no ****CONTROL RW** or **NORW** precompiler-directing statement (see 1.8.1). These options enable you to use the **same** JCL or command for **all compilations**.

If **RW** is specified, every source program will be assumed to contain Report Writer code **unless** an **CONTROL NORW statement is present at the start of the program. This statement thus saves the small unnecessary overhead of running the precompiler in such cases.

If **NORW** is specified, every source program will be assumed to contain **no** Report Writer code **unless** an *CONTROL RW statement is present at the start of the program.

```
┌─ SEQUENCE ──────────────────────────┐
│ ►►─────┬─SEQUENCE───┬──►◄            │
│        └─NOSEQUENCE─┘                │
└──────────────────────────────────────┘
```

shared

SEQUENCE indicates that the *sequence columns* (1 through 6) in the original source should be checked for correct ascending sequencing. If sequence errors are found, two asterisks (******) are printed against the offending line and a single warning message (RW-882) is printed at the end. The compiler's messages resulting from this option are ignored by **PRTEXIT(RW)** because the sequence numbers in the intermediate source are never in strict ascending sequence. This option should therefore not be used without **PRTEXIT(RW)**.

NOSEQUENCE indicates that sequence checking is to be omitted.

```
┌─ SIZE ──────────────────────────────────────┐
│ ►►──SIZE(──┬─integer──┬────┬──┬─)──►◄       │
│            │          └─K──┘  │             │
│            └─MAX──────────────┘             │
└─────────────────────────────────────────────┘
```

shared

The SIZE clause has its usual effect on the compiler. The precompiler uses this clause to ensure that the compiler receives the intended amount of GETVIS space. (See 2.1.2.)

```
┌─ SOURCE ─────────────────────────┐
│ ►►─────┬─SOURCE───┬──►◄           │
│        └─NOSOURCE─┘               │
└───────────────────────────────────┘
```

shared

SOURCE causes the original source program to be listed.

NOSOURCE suppresses the listing of the source. NOSOURCE is invalid if you specify LGSEQ, MGENER, SEQUENCE or FLAG with two operands.

```
┌─ SPACE ──────────────────────────┐
│ ►►──SPACE(──┬─1─┬──►◄             │
│             ├─2─┤                 │
│             └─3─┘                 │
└───────────────────────────────────┘
```

shared

This option specifies the spacing between successive lines of the SOURCE listings.

```
┌─ TERM ──────────────────────────┐
│ ►►─────┬─TERM───┬──►◄            │
│        └─NOTERM─┘                │
└──────────────────────────────────┘
```

shared

TERM causes the precompiler to display all messages on the System Log, as well as having the usual similar effect on the compiler.

```
┌─ XCAL ─────────────────────────┐
│ ►►────┬─XCAL───┬──►◄            │
│       └─NOXCAL─┘                │
└─────────────────────────────────┘
```

precompiler only

XCAL enables the precompiler to invoke external run time routines in certain cases (see 2.2.1) where doing so will produce a more efficient, or completely compatible, or more satisfactory program. If XCAL is in effect, there are no restrictions on the size and format of controls and any run time error messages are displayed in full.

NOXCAL instructs the precompiler to avoid, if possible, the generation of CALLs to certain run time routines. The effects of specifying NOXCAL are as follows:

- If a *CONTROL* clause is present in an RD (other than *REPORT* or *FINAL*), control breaks will be processed through generated in-line code, rather than by using the LENGTH register and subroutines CXRCTCP, CXRCTUS, and CXRCTRS. If this option is used, CONTROL operands must be, implicitly or explicitly, **USAGE DISPLAY** and there is an upper limit of 256 to their size (see CTRLEN option above). See the *Programmer's Manual* for details.
- If an error occurs during the running of the program, a short message is printed, showing only the identity of the message but no explanatory text, page, or line number information.

Use of **NOXCAL** prevents the generation of calls to all external subroutines from an unchanged DOS/VS COBOL program.

2.3.4 Restrictions to Other Compiler Options

The **NOCOMPILE(x)** option is unaffected by errors found by the precompiler.

The **DYNAM/NODYNAM** option has an important indirect effect on the precompiler since calls to the run time routines are generated as a CALL "literal". See 2.2.2 for more information.

The **NUMBER** option is not allowed with PRTEXIT(RW).

The **WORD** option could adversely affect the precompiler's code generation if you use it to restrict the use of certain reserved words. The precompiler does not generate any reserved word (such as ALTER) that you would be **likely** to restrict. A list of the reserved words that may be used by the precompiler in its COBOL code generation are listed in **Appendix F**.

3

Installation and Customization for VSE

This part contains information to enable you to install the Report Writer software under VSE/ESA from the supplied distribution tape, customize the precompiler, prepare the run time library, and verify that the installation and customization steps have been successful.

If you are installing the Run Time Library only (5798-DZX), proceed at once to stage 3.6.

Before installing, you should check with the Information Network, or the supplier (see inside front cover), to ensure that you have any amendments due for this release.

3.1 Allocating the Report Writer Library

To install COBOL Report Writer, you need to do just **two** things: **allocate** enough disk space for the library, and **copy** it using LIBR. It is then ready for immediate use. Optional additional steps can be done to customize the system.

Before beginning the copy, you should first allocate disk space for the Report Writer library **RW**. It requires **2700** 1024-byte blocks (although you can reduce this space requirement after installation - see next section). When you have allocated either VSAM-controlled or basic disk space for this library, you should code the appropriate **DLBL** for **RW**, plus appropriate **EXTENT** and **ASSGN** statements in the case of non-VSAM space, in the JCL of the next section. These definitions can then be added to your *standard labels*, if you wish, to make **RW** available for all COBOL jobs.

3.2 Copying the Precompiler (5798-DYR)

First **MOUNT** your installation tape *read-only* on a tape unit of your choice. It was created using LIBR **BACKUP**, so you simply use LIBR **RESTORE** to recreate it. The following JCL may be used, tailored as required by your installation standards:

```
// JOB RWINSTL INSTALL COBOL REPORT WRITER PRECOMPILER
*  definition of library
if VSAM...
// DLBL RW,'REPORT WRITER LIBRARY',,VSAM,CAT=catalog
if basic disk space...
// DLBL RW,'REPORT WRITER LIBRARY',9999,SD
// EXTENT  SYS001,volser,...
// ASSGN   SYS001,DISK,VOL=volser,SHR
followed by...
// MTC  REW,cuu
// EXEC LIBR,SIZE=256K
DEFINE LIB=RW
RESTORE *    -
  TAPE=cuu   -
  LIST=YES   -
  REPLACE=YES
```

If you wish to preview the contents of the tape before actual copying, you could add the option **SCAN=YES -** in the above JCL.

Your library now contains phase, object, and source sub-libraries as follows:

RW.JCL	JCL procedures
RW.PREC	precompiler phases
RW.RUN	run time object and phase library
RW.COB	run time COBOL sources (APOST format)
RW.COBQ	run time COBOL sources (QUOTE format)
RW.ASM	run time Assembler sources

If you use the supplied defaults (in particular **NORTNEST**), then the only sub-libraries constantly required are **RW.PREC** and **RW.RUN**. The run time sources are provided to enable you to re-compile the run time library, if the need arises, and to use the **RTNEST** option (which copies run time routines in source form). If you have standardized on **QUOTE** or **APOST** throughout your facility, you will need only **RW.COBQ** or **RW.COB** respectively. Hence you can reduce the size of your library if disk space is short (or select only the sub-libraries you need by coding **SUBLIB=...** instead of the asterisk in the **LIBR RESTORE** operation above).

Both COBOL source libraries contain sample COBOL file handlers, **CRFHNOPF**, **CRFHCHAN**, and **CRFHMODL** and **COPY** books for their standard linkage areas (**RWFCACOM**, **RWRCACOM**, and **RWPLNCOM**). You may use one of these as a basis for producing your own Report Writer file handlers for directing Report Writer output to a non-standard medium in a manner that you decide. For further information, consult the *Programmer's Manual*.

3.3 Customizing the Precompiler

Perform this step only when you want to change the supplied precompiler default settings. These settings are all indicated by underlined choices in 2.3.3 above. You can do the following step now or at any time in the future if you decide to alter your installation defaults. If you will be using **INEXIT(RW)**, you need only worry about the options marked *precompiler only*, since the **compiler's** default values are used for all the options shared with the compiler (QUOTE, ADV, etc.).

This step runs the COBOL compiler and link editor to create a new copy of the customized-options phase **SPCHOPTS**. JCL to perform this task is printed below. Insert your options before the first /* line. The definitions for the work files (**IJSYS01** through **IJSYS07**) have been omitted. If you do not have these defined as standard labels, you must insert them into the JCL also. The LIBDEF statement must include both the compiler and the COBOL run time system. You will need to modify this to reflect the COBOL compiler you are currently using.

The option **EX(INX(RWCUS))** invokes the **INEXIT** routine **RWCUS**, which generates all the COBOL source code automatically. Where the compiler would normally expect a COBOL source program, you code a list of options instead. The rules for coding options is given in the next paragraph.

```
// JOB RWCUSTM  CUSTOMIZE COBOL REPORT WRITER PRECOMPILER
// usual COBOL workfiles, if not globally defined
// LIBDEF PHASE,SEARCH=(lib.COMPnnn,RW.PREC,lib.PRODnnn,lib.SCEEBASE)
// OPTION CATAL,NODECK
// EXEC
IGYCRCTL,SIZE=IGYCRCTL,PARM='QUOTE,RES,NAME,EX(INX(RWCUS),NOPRTX)'
default options (see next paragraph)
/*
// LIBDEF OBJ,SEARCH=(lib.PRODnnn,lib.SCEEBASE)
// LIBDEF PHASE,CATALOG=RW.PREC
// EXEC LNKEDIT
/*
```

You may wish to alter your default options by repeating this step at any time, so it is advisable to keep a copy of the JCL with the options that you last used.

3.3.1 Options

Code options beginning anywhere on the line. You may combine several on a line separated by a comma. The precompiler's **EXIT** parameter can be split across several lines at the commas. (You cannot code INEXIT, LIBEXIT, or PRTEXIT as though they were independent options.) Comment lines may be coded by placing an asterisk (*****) in the first column. Here is a random sample showing how options may be coded:

```
*Report Writer Precompiler Options
NOOSVS
FMODE(PRNT),NOMGENER,NORW
EXIT(INEXIT('string1',module1),
    (LIBEXIT('string2',module2),
    (PRTEXIT('string3',module3))
```

Options that you do **not** specify are set to their **supplied default** values (<u>underlined</u> in 2.3.3, **not** the values you last set them to. If you wish to reset **all** options to their supplied defaults, follow the EXEC statement directly with a **/*** line.

3.4 Compiling the COBOL Run Time Routines

You need to perform this step if your COBOL compiler is not COBOL/VSE and also if you want to change the COBOL options originally used in generating your run time library. Your run time library **RW.RUN** was generated using IBM COBOL for VSE Release 1.1.0 and Assembler, as appropriate. Since every run time routine is provided in source form, you can re-compile or re-assemble all or any part whenever necessary. The only forseeable reason for doing this is to change the compiler options. The only options originally specified that can affect operation are:

RENT,NOOPTIMIZE,OUTDD(SYSOUT),NOSSRANGE,NOTEST

In particular, if you want the NORENT option, because you do not require *above-the-line* execution, then you must re-compile all the COBOL run time routines.

If you have **POWER**, you can make use of the procedure **COMPRWRT** in **RW.JCL**. This in turn uses the procedure **COMPRWR** to compile each routine. You should first use LIBR or an editor to fetch **COMPRWRT PROC** and **COMPRWR PROC** from sub-library **RW.JCL**. Change the **PARM** option settings in **COMPRWR PROC** according to your requirements. It contains no COBOL work file definitions, so these must also be inserted if they are not already defined in your standard labels. You may also need at change or insert **POWER** commands, and **CP** control commands if you are executing your jobs via VM. The positions are indicated by comments in the JCL of these procedures. When you re-catalog them in the library, be sure to specify **DATA=YES**.

These compilation procedures use an input-to-punch program to "punch out" a job that uses LIBR to catalog the object decks. Therefore your job should contain no other job steps other than **EXEC PROC=COMPRWRT** and the POWER **$$ PUN** line should specify **DISP=I,CLASS=0**. Here is an outline of the JCL required:

```
* $$ PUN PUN=SYSPCH,DISP=I,CLASS=0
// JOB RWCOMP  COMPILE COBOL REPORT WRITER RUN TIME LIBRARY
// LIBDEF PROC,SEARCH=RW.JCL
// EXEC PROC=COMPRWRT
```

If you are **not** using **POWER**, or if you prefer to use your own JCL for compilations, you should examine **COMPRWRT PROC** for the names of the routines to be compiled and compile each of them into **OBJ** form in **RW.RUN**. (They are also listed in **Appendix A**.)

Always follow this step with a separate job to link edit the routines into **PHASE** form, as explained in the next paragraph.

3.4.1 Link Editing the COBOL Run Time Routines

This step **must** be done whenever you re-compile or re-assemble any of the run time routines. The run time routines must be present in **PHASE** versions, so that programmers can use the **DYNAM** option, and also because certain of the routines are always called dynamically. The JCL to perform this task will be found in the procedure **LINKRWRT** in sub-library **RW.JCL**. Suitable JCL is as follows:

```
// JOB RWLINK  LINK EDIT COBOL REPORT WRITER PRECOMPILER
// LIBDEF PROC,SEARCH=RW.JCL
// EXEC PROC=LINKRWRT
```

3.5 Installation Verification

RW.COB and **RW.COBQ** contain a sample Report Writer source program, **RWTEST01**, which should be precompiled, compiled, link edited, and run to check that the installation has been successful. The JCL to do this will be found in the file **TESTRW01** in **RW.JCL**. (This also contains no COBOL work file descriptions. The program is too small to need the precompiler work file **IJSYS11**.) Suitable JCL is as follows:

```
// JOB TESTRW RUN COBOL REPORT WRITER SAMPLE PROGRAM
// usual COBOL workfiles, if not globally defined
// LIBDEF PROC,SEARCH=RW.JCL
// EXEC PROC=TESTRW01
```

Alternatively, you may use your own standard JCL for a COBOL compilation with the additional option **EXIT(INEXIT(RW),PRTX(RW))**, followed by a link edit and run.

You should obtain a calendar for the current year.

Try out any **PARM** options that are likely to be used by programmers. This may be done by overriding the JCL line containing the **PARM** or by extracting the JCL and embedding it directly in your job.

3.6 Installing the Library Only (5798-DZX)

Your supply tape contains only the run time libraries (that is, all but **RW.PREC**). Allocate the library RW and copy the tape as described in and above. A total of 2000 1024-byte blocks are required for the library.

4

Using the Precompiler on VSE

This part describes in detail how to load and run the precompiler on **VSE** after it has been installed from the supply tape and customized. It also describes which data sets are required and explains how options may be selected. Finally, it describes the procedures necessary to link-edit and run your resultant program.

4.1 Using INEXIT(RW),PRTEXIT(RW)

INEXIT(RW) is the normal way to precompile and compile any source program. You use your existing JCL for a COBOL compilation, adding an additional **EXIT** option to the compiler's PARM string as follows:

PARM='...EXIT(INEXIT('' 'parameters' '',RW),PRTEXIT(RW))'

which may be abbreviated to

PARM='...EX(INX('' 'parameters' '',RW),PRTX(RW))'

The doubled apostrophes are required by the rules of JCL syntax. In the 'parameters' you code any options that apply to the **precompiler only**. The compiler does not recognize options placed inside the EXIT construction, so any options which are also used by the compiler must be coded **outside** the EXIT in the compiler's main PARM string. For example, if you want the options QUOTE and NOOSVS, you must code the following:

PARM='QUOTE,EX(INX('' 'NOOSVS' '',RW),PRTX(RW))'

If there are no precompiler options, because you are happy with the supplied or customized defaults, you code simply:

PARM='...EX(INX(RW),PRTX(RW))'

If you do not require the PRTEXIT, code the following:

PARM='...EX(INX(RW))'

If the VSE PARM string becomes too long, you can define the EXIT sub-options in the COBOL compiler's customization macro. (You can create a second copy of IGYCDOPT with these EXIT sub-options set, placing it in a different library, and select it by placing this library at the front in your **LIBDEF PHASE** line.)

The COBOL compiler does not allow the *parameters* to the INEXIT to be customized, but you can specify any of the precompiler options to the precompiler customization procedure (see 3.3).

4.1.1 The Work File IJSYS11

The additional data set **IJSYS11** must be assigned in your compilation JCL as working space for the precompiler in case GETVIS space is insufficient. It may be assigned in a similar way to the compiler work files **IJSYS01** to **IJSYS07**. It may be assigned to any basic (maximum 16 extents) or VSAM-controlled disk area, on any CKD or FBA device.

4.2 Using the Stand-alone Precompiler

The stand-alone precompiler **SPCRWCOB** reads the source from **SYSIPT** and writes it to **SYSPCH**. The output may be retained or fed direct to the compiler. Sample JCL to do a separate precompilation followed by a compilation is given below.

```
// JOB      RWPREC COBOL REPORT WRITER COMPILATION IN TWO STEPS
*  Precompilation step:
// DLBL     IJSYS11,'COBOL.WORKFILE.IJSYS01',0,SD
// EXTENT   SYS001,volser,...
// ASSGN    SYS001,DISK,VOL=volser,SHR
// DLBL     IJSYSPH,'intermediate-filename',0,SD
// EXTENT   SYSPCH,volser,...
ASSGN    SYSPCH,DISK,VOL=volser,SHR
// EXEC     SPCRWCOB,SIZE=SPCRWCOB,PARM='options'
CLOSE    SYSPCH,00D
*  Compilation step:
// DLBL     IJSYSIP,'intermediate-filename',0,SD
// EXTENT   SYSIPT,volser,...
ASSGN    SYSIPT,DISK,VOL=volser,SHR
   ... rest of JCL as for usual COBOL compile ...
CLOSE    SYSIPT,SYSRDR
```

As the first file identifier suggests, IJSYS11 may re-use some of the compiler work space.

If **COPY** statements may be present in the source (**LIB** option set) and the precompiler option **COPY** is in effect, add the following before the EXEC statement:

```
// LIBDEF SOURCE,SEARCH=(your copy libraries)
```

If **NOCOPY** is in effect, the LIBDEF may be placed in the compilation step.

If **RTNEST** is in effect, add before the EXEC statement:

```
// LIBDEF SOURCE,SEARCH=(RW.COB,your copy libraries)
```

or if the option QUOTE is in effect:

```
// LIBDEF SOURCE,SEARCH=(RW.COBQ,your copy libraries)
```

4.3 Linking and Running the Compiled Program

4.3.1 Run Time Library

Follow your compilation step with a **LNKEDT** step and the program is ready to use. In general, the precompiler will have generated some external references to the Report Writer run time library. An explanation of the functions of all the subroutines in this library are given in **Appendix A**. If your program requires run time routines (and they are not all included in source form using **RTNEST**) and the compiler option **NODYNAM** was in effect, you should include the Report Writer run time library in your JCL for the Linkage-Editor step:

```
// LIBDEF OBJ,SEARCH=(...,RW.RUN,...)
```

If the compiler option **DYNAM** was in effect, the Report Writer run time library is needed instead at program run time:

```
// LIBDEF PHASE,SEARCH=(...,RW.RUN,...)
```

4.3.2 User-Developed Report Writer Routines

As well as the supplied routines, the run time library may contain routines written by you or other local personnel. These are fully described in the *Programmer's Manual* and consist of:

- Report Writer FUNCTION routines
- Independent Report File Handlers

If **NORTNEST** and **NODYNAM** are in effect, and user-written FUNCTION routines are required, the library containing them should also be included in the link edit step. User-written file handlers are always called dynamically.

Appendices

Appendix A

List and Description of Programs and Library Routines

The following is a list of the compile-time and run time programs, together with their sources that are copied from your distribution tape on installing.

(i) Precompiler

Phase Name	Purpose
RW	**INEXIT/PRTEXIT routine** This phase is executed by the compiler when you code **EXIT(INEXIT(RW))** in its PARM string. If you include **PRTEXIT(RW)**, the same phase is used for the PRTEXIT.
SPCRWCOB	**Stand-alone Precompiler** This program may be used as instead of the INEXIT to precompile a Report Writer source separately.
SPCHCXIN **SPCHCXTB** **SPCHCXDP** **SPCHCXGE**	**Precompiler Phases** These run in succession to precompile a single source. For a nested or batched program source, they run again for each new PROGRAM-ID.
SPCHNDLR	**Common Routines** This phase contains the common routines used by the above precompiler phases.
SPCHCXOT	**Output Phase** This phase runs at the end of the precompilation, either to write out the intermediate source or, if **INEXIT(RW)** is in use, to feed that to the COBOL compiler.
SPCHPRTX	**PRTEXIT Phase** This module is called by phase RW if you include PRTEXIT(RW) in the EXIT option.
SPCHMESG	**Messages Phase** This module holds the text of the precompiler's standard messages.
RWCUS	**Customizing Program** This module is run as an INEXIT routine when you customize the precompiler.
SPCHOPTS	**Customized Options** This module contains default values for all the precompiler options. It can be re-generated using the customizing program.

SETCHAN	**Set up Printer Channels** This phase is used optionally to set the positions of printer channels before running a program that uses the **CHAN** file handler. Strictly speaking, this phase is part of the run time system, but it is stored in the precompiler library.
COBINPUT	**Compilation INEXIT Routine** This phase is used by the compilation procedure **COMPRWR** to enable the program name to be passed as a parameter.
SPCIPTPH	**Input-to-Punch Routine** This phase is used by the procedure **COMPRWR** to generate a LIBR job to catalog the object decks.

(ii) <u>Run Time Library</u>

Run time routines are invoked **not** as a constant overhead, but only occasionally when required by the application. The circumstances of the use of each is given against each entry following.

<u>Module Name</u>	<u>Purpose</u>
(a) <u>COBOL Routines and Areas</u>	
CXRFHCON	**File Handler Steering Routine** This routine is invoked if the Report Writer program uses an *Independent Report File Handler*. It directs control to and from the file handler on each call, performing housekeeping and checking functions. It also handles the *PAGE BUFFER* and the *DUPLICATED* features, if called for.
CXRSTYLE	**STYLE processing** This routine is used whenever a STYLE clause is found in a Report Writer entry. It looks for the escape sequences inserted into the print data by the Report Writer code to implement <u>underline</u>, **boldface** and any other special effects required.
CXRCHMV **CXRCHNF**	**Variable-position field processing** These routines handle the positioning of fields in the report line whose starting position depends on the value of *COLUMN-COUNTER*.
CXRERNF	**Log Run time Error Condition** This routine displays an error number and message to indicate that a standard error condition has arisen during the execution of the Report Writer program. It is not used if **NOXCAL** is specified, in which case the program *DISPLAYs* a shorter message directly. Standard run time errors are listed below in **Appendix G**.

CXRPBF01-nn	**Page Buffer Handling** These routines handle the buffering of up to a whole page of data when the clause **WITH PAGE BUFFER** is used. Any number of these may be present, depending on how many report files **open simultaneously** require a page buffer. Six are normally provided (01-06), but additional routines can be produced simply by increasing the value of **nn** in the *program-id* and re-compiling.
CXRRELA	**REPEATED processing** This routine handles the buffering and side-by-side alignment of groups defined with the **REPEATED** clause.
CXRVARF	**Variable-length field processing** This routine is used to process fields defined with "<" (variable-length) PICTURE symbols.
CXRLWRP	**WRAP processing** This routine handles the **WRAP** clause which is used to produce "wrap-around".
CXRCTMV	**Copy CONTROL item** This routine is used to copy the contents of a *control data item* to and from its saved control area. It is used **only as a nested program**. If **NORTNEST** is specified, in-line code is substituted.

<u>FUNCTION Routines</u>

These are invoked when the corresponding FUNCTION is used. The second character of the alias name represents the number of parameters given in the **FUNCTION** clause. For instance, **FUNCTION DATE** (i.e. print today's date) calls **R0DATE**, whereas **FUNCTION DATE (WS-IP-DATE)** (i.e. print the given date) calls **R1DATE**.

Module Name	Mnemonic	Purpose
R0CTIME	**CTIME**	Print 12-hour clock time
R1DATE,R0DATE	**DATE**	Print current or specific date (day-month-year)
R1DAY,R0DAY	**DAY**	Print current or supplied day-of-week
R1DAYSIN	**DAYSIN**	Print date, converting from days elapsed (day-month-year)
R1MDATE,R0MDATE	**MDATE**	Print current or specific date (month-day-year)
R1MDAYS	**MDAYS**	Print date, converting from days elapsed (month-day-year)
R1MONTH,R0MONTH	**MONTH**	Print month name
R2MOVE	**MOVE**	Capture contents of internal register or counter

R0RDATE	**RDATE**	Real date, updated at midnight
R0RMDATE	**RMDATE**	Real date (month-day-year)
R0RYDATE	**RYDATE**	Real date (year-month-day)
R1STATE	**STATE**	Print US State name
R1STATEF	**STATEF**	Print US State name, including overseas territories
R0STIME	**STIME**	Print time, fixed at start
R0TIME	**TIME**	Print actual time
R1YDATE,R0YDATE	**YDATE**	Print current or specific date (year-month-day)
R1YRDAY,R0YRDAY	**YRDAY**	Print current date (YYDDD)
R1ZIP	**ZIP**	Print US ZIP code

<u>File Handlers and Dependent Routines</u>

As well as a copy of the source of each of the routines above, the supplied source library contains the following file handler source items.

<u>Handler Name</u>	<u>Function</u>
CRFHMODL	**MODL** File Handler. This file handler enables several independently compiled modules to write to the same report file. It may be used as a basis for other user-written file handlers. Full details are given in the source. It is also described in Part 5 of the *Programmer's Manual*.
CRFHNOPF	**NOPF** File Handler. This file handler writes without *page feeds*. Whenever it needs to advance a page, it writes blank lines down to the bottom of the page. Full details are given in the *Programmer's Manual*.
CRFHCHAN	**CHAN** File Handler. This file handler writes using *printer channels* wherever possible. Details are given in Appendix and in the *Programmer's Manual*.
CRFHDUPL	**DUPL** File Handler. This file handler emulates the ability of DOS/VS COBOL to write simultaneously to two report files.
RWFCACOM	COPY library source of *File Control Area*.
RWRCACOM	COPY library source of *Report Control Area*.
RWPLNCOM	COPY library source of Report Writer *Print Line*.

(b) <u>Assembler Routines</u>

CXRCTCP **CXRCTUS** **CXRCTRS**	**Handle controls for VS COBOL II** These routines save, compare, and restore Control values when **XCAL** and **NORTNEST** are specified. If NOXCAL is specified, in-line code is generated instead.

CXRGBLS	**Process GLOBAL requests** This routine executes the inter-program linkage for GLOBAL items. It is invoked when a program issues an INITIATE, GENERATE, or TERMINATE statement for a **GLOBAL** report, or when a **USE GLOBAL AFTER REPORTING** section is implicitly invoked, in a different containing program.
CRFHPRNT	**Basic Print File Handler (MODE PRNT)** This file handler is used if a report requires the STYLE, DUPLICATED, or PAGE BUFFER facility, and is not already using a file handler. CRFHPRNT writes direct to **SYSLST**.
CXRFHUSG	**Assist Routine #1 for USING Phrase** This is called as a "front end" to a COBOL file handler control routine **CXRFHCON** with additional parameters defined by a **USING** phrase.
CRFHXXXP	**Assist Routine #2 for USING Phrase** This is called as a "front end" to a COBOL file handler with additional parameters defined by a **USING** phrase.
CXPHJCOM **CXPHPRNT**	**Subroutines of CRFHCHAN**

Appendix B

Clauses that Require Run Time Routines

1. Routines Written in Assembler

a. If the option **XCAL** is in effect, and a program contains a **CONTROL** clause (a commonplace feature at all levels) the *Assembler* routines **CXRCTCP, CXRCTUS, CXRCTRS** will be used for the testing of control breaks and copying of your CONTROL *identifiers*. See page 33 for full details.

b. If any report is defined as **GLOBAL** (so that it can be referred to from a different program in a nested structure), the *Assembler* routine **CXRGBLS** is invoked.

c. If there is a **MODE** clause with a **USING** phrase, the *Assembler* routines **CXRFHUSG** and **CRFHXXXP** are used.

d. If the program uses one of the following features and there is no **MODE** clause in the **SELECT** statement for the corresponding report file, the *Assembler* report file handler **CRFHPRNT** will be used. The features in question are:

- The **DUPLICATED** clause,
- The **WITH PAGE BUFFER** clause,
- The **STYLE** clause,
- The **UPON** option of the INITIATE statement,
- The use of **CODE** in more than one **RD** for the same file, where not all CODE operands have the same length,
- The (erroneous) omission of the **FD** entry,

If none of these features is present, the program will use direct WRITEs at run time (unless a **MODE** clause is coded for the file). For full details of file handlers, see the *Programmer's Manual*.

Even if these features are present, use of the *Assembler* file handler can be avoided by using a COBOL file handler: either one of those which are supplied with the precompiler, such as **CRFHMODL**, or a user-written file handler. This may be done in one of two ways:

i In the program source, add the clause **MODE IS mode** to the appropriate **SELECT** statement(s), **or**

ii Assuming that there is no **MODE** clause already for the report file(s) in question, use the option **FMODE(mode)**.

Either of these ways forces use of the file handler **CRFHmode**.

A **COBOL** *file handler* may however have the following disadvantages over the *Assembler* file handler:

i The supplied COBOL file handlers cannot handle **more than one** file if they will be **open simultaneously**, since they contain only a single FD (File Description) entry. (However, a *user-written* file handler may of course have any number of FD entries and could, for example, allocate multiple files to different FDs in order of OPENing.)

ii **COBOL** file handlers use a pre-defined *logical record length* and *record format* by virtue of the record description and/or FD clauses and hence ignores the **RECORD CONTAINS** and **RECORDING MODE** of the original report file. (The supplied COBOL file handlers assume fixed-length records of 133 bytes.)

iii The supplied **COBOL** file handlers place the value of any **CODE after** instead of **before** the carriage control character of each record. A *user-written* file handler can of course place the CODE wherever desired. (Note that CODE is now very little used for its original purpose, namely to **spool** several print files to the same "tape".)

2. Routines Written in COBOL

COBOL run time routines are used to implement certain special functions, chiefly of the more "advanced" kind. The Report Writer features that cause them to be introduced are as follows:

a. If **XCAL** is in effect, and no file handler is in use, a CALL to **CXRERNF** is always generated at the TERMINATE statement. This routine handles run time errors by printing a full explanatory message. If **NOXCAL** is in effect an in-line **DISPLAY** is used instead, but this gives only the reference number of the error.

b. If any **MODE** clause (other than **PRNT**) or any **FUNCTION** clause is specified, the corresponding run time routine will be invoked. Any of these routines may be *user-written*.

c. If **RTNEST** and **NOXCAL** are specified, and the program contains a CONTROL(S) clause, the routine **CXRCTMV** is invoked to copy controls to and from the "saved controls" area (see **Appendix C** - *How CONTROLS are Implemented*).

d. The following "more advanced" features cause additional CALLs to be generated: **REPEATED clause, PICTURE symbol <, WRAP clause**, any field that has a **variable horizontal position**.

Appendix C

How CONTROLS are Implemented

It is important to understand the way Report Writer's **CONTROL(S)** clause is handled because this clause frequently appears in both old and new programs.

If the **XCAL** option is in effect, the precompiler uses Assembler library routines to test for control breaks and save the controls.

The advantage of these routines are:

i They run rather more efficiently than the alternative methods described under **NOXCAL** below.

ii They allocate space for the *"saved controls"* dynamically and fully automatically, so there is no need to worry about the "maximum control size" option (**CTRLEN**), and no unexpected run time errors will arise because the saved controls are shorter than the actual controls.

iii As with the **RTNEST** option (see below), they allow **CONTROL** *identifiers* to have **any** COBOL PICTURE, so no unexpected compilation errors will arise from the CONTROL(S) clause of a correct DOS/VS COBOL program.

These routines assume a native collating sequence. Hence, if an **ALPHABET** clause is specified in the program, **NOXCAL** may be necessary to ensure that the specified collating sequence is used.

If **NOXCAL** is in effect, the following different implementation is used:

The precompiler allocates a "saved control" area for each control level (other than REPORT/FINAL). The size of each saved control location is taken from the value established in the **CTRLEN** option (see 2.3.3). This would ideally be exactly the same as the length of the corresponding CONTROL data item. However, the precompiler **does not scan the whole DATA DIVISION** for the **PICTURE** of the control item and must therefore assume a reasonable maximum size. The default as supplied is **80** (one screen's width) but up to 256 may be specified.

If **RTNEST** is also in effect, the precompiler will generate a CALL to the nested program **CXRCTMV** to move the CONTROL data items to and from the saved control areas.

If instead **NORTNEST** is also in effect (preventing the inclusion of run time routines in source form), the code to move the CONTROL data items to and from the saved control areas is generated **in line**. This has the drawback that only data items with an implicit or explicit **USAGE** of **DISPLAY** can be used as *CONTROL identifiers* (not **COMPUTATIONAL** or **COMPUTATIONAL-n**). So it is possible for a valid DOS/VS COBOL program to result in a compilation error. This situation is easy for the programmer to rectify (see *Programmer's Manual: CONTROL Clause*). This combination (**NOXCAL,NORTNEST**) is necessary to ensure SAA compatibility of the intermediate source.

If any program uses CONTROL items that have a length of more than the value of **CTRLEN**, this error will be detected (by the subroutine CXRCTMV) if **RTNEST** is in effect but will **not** be detected if **NORTNEST** is in effect. In either case, the contents of CONTROL fields when printed in a CONTROL FOOTING group may then appear truncated.

Appendix D

Using the CHAN File Handler

This file handler makes best use of any available printer channels by calculating how to get to each next line position in the fewest number of transfers. It is invoked by coding MODE IS CHAN in the program or by using the **FMODE(CHAN)** option. The positions of the channels are defined by executing the program **SETCHAN** before running the Report Writer program. The parameters follow in the JCL. The format of these parameters is:

c(p q...)

where *c* is an integer from 1 to 12 and *p, q,...* are integers from 1 to 255. A new line may begin anywhere. A comma may be used to separate the p, q... terms if desired and spaces may appear anywhere between terms and separators and at the start of the line, if desired. Comment lines may be written by placing an asterisk (*****) in column 1.

The term *c* represents the channel number and *p, q,...* are the line numbers that can be reached by skipping to that channel. If several channels are defined, these parameters follow one another in any order. If channel 1 is defined, the only line number defined for it must be 1; if channel 1 is not defined, this is assumed.

For example, to set up channel 1 and channel 2 with positions 20 and 40, the following JCL is used:

```
// LIBDEF PHASE,SEARCH=(RW.PREC,libr.SCEEBASE)
// EXEC SETCHAN,SIZE=SETCHAN
1(1)
2(20,40)
/*
```

The Report Writer program is then executed directly afterwards in the same job.

To **reset** all channels, an empty SYSCHANS data set can be set up.

If a 3800-type printer is in use, the Forms Control Block (FCB) still needs to be configured. This file handler does not do that. Moreover, the FCB channel settings must agree with those given to SETCHAN.

Appendix E

Printer Styles

Printer styles are defined by means of the **STYLE** clause (see *Programmer's Manual*). They enable special effects to be made use of, depending on the type of printer in use. The following printer **TYPE**s are available:

IBM-3800 or 3800 (the IBM laser printer)
IBM-3211 or 3211 (line printer)
IBM-1403 or 1403 (line printer)

If no **TYPE** is given, **TYPE 3800** is assumed in default.

On 3800, all the styles apart from **UNDERLINE** are implemented using a *Table Reference Character* (TRC). This is an additional character that appears immediately after the carriage control character. This option is set by the file handler **PRNT**. TRC 0 indicates **NORMAL** printing. TRCs 1 through 3 are arbitrarily assigned the following names:

1 - **HIGHLIGHT**
2 - **ALT-FONT**
3 - **GRAPHIC**

Thus TRC 1 is normally assigned to a HIGHLIGHT (i.e. BOLDFACE effect). It is up to the user to attach appropriate meanings to ALT-FONT and GRAPHIC. UNDERLINE and HIGHLIGHT may also be achieved on an older impact printer.

The following STYLEs are therefore available:

NORMAL	
UNDERLINE	This causes under-strike characters ("_") to be written in the same positions as the characters to be underlined.
HIGHLIGHT	This is implemented on **3800** as **Table Reference Character 1**. It is implemented on impact printers by printing the same line twice ("double hammering").
ALT-FONT	This is implemented on **3800** as **Table Reference Character 2**. It is not available on impact printers.
GRAPHIC	This is implemented on **3800** as **Table Reference Character 3**. It is not available on impact printers.

Internal formats

These styles are encoded by the precompiler as Escape sequences with the following composition (Esc - Escape character):

start-UNDERLINE:	*Esc*:UN>
start-HIGHLIGHT:	*Esc*:HI>
start-ALT-FONT:	*Esc*:AL>
start-GRAPHIC:	*Esc*:GC>
end-style:	*Esc*:<

Appendix F

COBOL Reserved Words Generated by the Precompiler

The following COBOL reserved words may be used by the precompiler in its COBOL code generation.

ADD	LOW-VALUES	SIZE
ALL	MOVE	SOURCE-COMPUTER
AND	MULTIPLY	SPACES
CALL	NOT	SUBTRACT
COMP	OBJECT-COMPUTER	SUPPRESS
COMPUTE	OCCURS	SYNC
COPY *	OF	THRU
DEPENDING	ON	TIMES
END-PROGRAM *	OR	TO
ERROR	PERFORM	UNSTRING
EXIT	PICTURE	UNTIL
FROM	RIGHT	USING
GIVING	REDEFINES	VALUE
GO	REPLACING	VARYING
IF	ROUNDED	WORKING-STORAGE
IN	SECTION	ZERO

In addition, the precompiler will reproduce any COBOL condition used in a PRESENT/ABSENT WHEN clause, thereby reproducing any reserved words used within it.

Words marked * are generated only if the **RTNEST** option is used. If RTNEST is used, COBOL code from any of the run-time routines is incorporated directly into the program, and *these may contain keywords other than those listed above*.

Appendix G

Run Time Messages

These conditions occur only at run time. Unless an *Independent Report File Handler* is in use that directs them elsewhere, all messages are displayed on **SYSOUT**. The line and page number are also displayed and, if a file handler is in use, the name of the report.

Internal Report Writer Errors

These appear as the result of an error during the formation of a report line or page, and are generated by the Report Writer code itself, rather than by a run time routine.

REPORT WRITER ERROR n

is always printed, and in addition if the **XCAL** option is in effect (see page 33), one of the following explanatory messages will appear:

Value of n	Message and Explanation
1	**COLUMN OVERLAP WITHIN LINE: PREVIOUS CHARACTER(S) OVERWRITTEN** This happens when two or more absolute elementary overlapping fields, or groups of columns, with PRESENT WHEN clauses (or the equivalent) were both present at the same time. The second field will overwrite all or part of the first. The precompiler will have given an informational message RW-251-I at precompilation time and will have assumed them to be mutually exclusive This error usually has no serious effect on execution.
2	**LINE EXTENDED BEYOND LIMIT: TRUNCATED** This condition will occur when several conditional relative COLUMN entries (COLUMN + n PRESENT WHEN ...) happen to be all present and their total size exceeds the LINE LIMIT. The precompiler would have assumed that at least some were mutually exclusive.
3	**LAST DETAIL IDENTIFIER OUT OF RANGE: USING PAGE LIMIT** This implies that the program contains the identifier form of the LAST DETAIL clause but, when this was evaluated, the contents were found to be higher than the LAST CONTROL FOOTING value.
5	**LINE OVERLAP: UNSCHEDULED PAGE ADVANCE MAY OCCUR** This happens when two or more absolute lines, or groups of lines, with PRESENT WHEN clauses (or the equivalent) were both present. The precompiler would have assumed that at least some where mutually exclusive. This will cause an unscheduled page advance without the usual production of PAGE FOOTING and PAGE HEADING groups, with lines that have the same LINE number appearing on successive pages.

6 **PAGE OVERFLOW: PAGE WILL EXCEED LIMIT**

This condition will occur when several conditional relative LINE entries (LINE + n PRESENT WHEN ...) all happen to be present and their total vertical size exceeds the maximum size normally allowed for the group. The precompiler would have assumed that at least some were mutually exclusive.

7 **LINE LIMIT IDENTIFIER OUT OF RANGE: USING DEFAULT**

This message implies that the identifier form of the LINE LIMIT clause has been used and that its value was found to be higher than the maximum record length of the report file.

8 **REPORT-NUMBER OUT OF RANGE: CHANGED TO 1**

This means that the field REPORT-NUMBER was not in the range 1 to the DUPLICATED value. Its value is changed to 1.

10 **SIZE ERROR ON STORING EXPRESSION**

This message will appear when a SOURCE clause contains an expression that causes a zero-divide error or an overflow when its value is computed before storing in the report line. Report Writer will take the error action specified by the OVERFLOW clause.

11 **SIZE ERROR ON SUMMING**

This message will appear when a SUM clause or term was coded and an overflow condition occurred on adding into the total field. Report Writer will take the error action specified by the SUM OVERFLOW clause.

14 **REPORT WRITER HAS INITIATED REPORT BY DEFAULT**

This message implies that the INITIATE statement has not been executed when a GENERATE for the same report was executed.

15 **AT LEAST ONE TOTAL FIELD HAD NOT BEEN PRINTED ON TERMINATE**

This message is issued when the total fields (other than those with RESET ON FINAL) are checked on TERMINATE to ensure that their values are all zero, indicating that they have all been "printed" in the report. This message will appear in the following circumstances:

(a) When a SUM or COUNT clause or term was coded in a DETAIL group that was not generated at the end when non-zero values had been accumulated.

(b) When a SUM or COUNT clause is subject to a PRESENT WHEN clause, or the equivalent, and the condition prevented the last total from being displayed. This fault may occur innocently when a SUM or COUNT is used for some purpose other than to be "printed".

File Handler Errors

These messages may be issued by the File Handler Control routine:

REPORT WRITER ERROR n IN FILE HANDLER xxxx

or

REPORT WRITER PAGE BUFFER ERROR n

always appears, and

IN REPORT rrrrrr ON PAGE ppp LINE lll

appears if the report has been initiated.

The value of **n** may be any of the following:

Value of n	Message and Explanation
8	**REPORT-NUMBER OUT OF RANGE: NO DUPLICATION** The value of REPORT-NUMBER was found to be less than 1 or greater than the DUPLICATED integer. This indicates a corruption, since REPORT-NUMBER is checked independently by the Report Writer code.
11	**FILE ALREADY OPEN** An OPEN is being performed but the state of the current file is already "open". The OPEN is ignored.
33	**FILE NOT OPEN: OPEN OUTPUT EXECUTED IN DEFAULT** The report file was not in "open" mode for an operation other than OPEN, the file was opened as for OUTPUT.
34	**REPORT NOT INITIATED: INITIATED BY DEFAULT** The report was not in an "initiated" state when a GENERATE was executed. The file handler performs the INITIATE action by default. However, not all the actions, such as the clearing of total fields, will have been performed and the results are therefore unreliable.
35	**CHARACTERS IN PAGE BUFFER OVERWRITTEN BY DIFFERENT CHARACTERS BEFORE OUTPUT** Two different entries placed different non-space characters in the same position in the Page Buffer. The second entry will overwrite the first. (Space characters do not rub out a previous character. Identical characters are allowed to coincide without provoking this message.)
36	**COLUMN SET > LINE LIMIT: CHANGED TO 1** A SET COLUMN statement has set the margin beyond the LINE LIMIT.
37	**COLUMN SET NEGATIVE OR ZERO: CHANGED TO 1** A SET COLUMN statement has attempted to set the value of the margin to less than 1. The SET is ignored.

38 **PAGE BUFFER WIDTH EXCEEDED DUE TO SET COLUMN OR TOO MANY STYLES**

The left-hand margin (resulting from a possible SET COLUMN) and the size of the data line taken together exceed LINE LIMIT. The line is truncated at the limit.

39 **LINE SIZE EXCEEDS LIMIT: TRUNCATED**

The width of the line data, without taking account of any margin, exceeds the LINE LIMIT. Either the byte count of the data line or the LINE LIMIT held in the report control area has been corrupted.

40 **DATA LENGTH OVERRIDE EXCEEDS LINE SIZE: IGNORED**

The value stored in the field L-RCA-LINE-SIZE (the line size override) is greater than the size of the data line itself. This indicates either a corruption to the report control area or a fault in the setting of the line size and may be the result of incompatibilities between the precompiler and the run time software.

41 **LINE LIMIT TOO LARGE: CHANGED TO MAXIMUM (m)**

The LINE LIMIT should not exceed the absolute upper limit of 256.

44 **INTERNAL FILE HANDLER ERROR**

The file is being OPENed other than OUTPUT or EXTEND. Some file handlers may give this a special interpretation. Others will issue this message and assume OUTPUT.

49 **INTERNAL FILE HANDLER ERROR**

The file handler has detected an improbable line advance, indicating corruption of LINE-COUNTER (L-RCA-LINE-CNTR). The file handler does a "PLUS 1" advance.

50 **DUPLICATED NUMBER > m**

The integer of the DUPLICATED clause exceeds the maximum permitted (m). This messages indicates a corruption, since the maximum is an arbitrary high value.

55 **ATTEMPT TO SET LINE OUT OF RANGE OR BEFORE POSITION ALREADY WRITTEN**

Either: a SET LINE clause has either set the LINE-COUNTER to a value outside the range 1 to PAGE LIMIT. Or it has set it to a position above a line that has already been written in RELEASE mode; the program should generate the upper lines with the page SET to HOLD.

56 **REPORT'S MAXIMUM LINE BYTE WIDTH IS TOO HIGH**

57 **REPORT'S MAXIMUM PAGE SIZE IS TOO HIGH**

These messages are displayed by the PAGE BUFFER handler if the byte length of a print line, or the number of lines per page, respectively, exceeds the dimensions of its own internal storage table. These are set to generous limits as supplied (see the source of CXRPBF01). To change these limits, re-compile the PAGE BUFFER handler(s) changing the limit both in the OCCURS and clause and in the location used in the test, and inform your supplier, so that the limits can be increased in any future release.

58 **LINE-COUNTER < 1 OR > PAGE LIMIT**

A check on the feasibility of the value of LINE-COUNTER has failed. The line will appear in an unscheduled position on the page.

61 **REPORT ALREADY INITIATED: INITIATE IGNORED**

An INITIATE was executed when the report was already "initiated".

63 **INTERNAL FILE HANDLER ERROR (not COBOL)**

The DDname for the main report file is not declared. The **OPEN** cannot take place.

64 **INTERNAL FILE HANDLER ERROR (not COBOL)**

For a multiple file (**DUPLICATED** clause), a series of DDnames are required of the form **dddddd01** to **ddddddnn** where **dddddd** is the root name and nn is the maximum number given in the **DUPLICATED** phrase. One of these DDnames had not been declared.

67 **NO FREE PAGE BUFFER AVAILABLE**

Too many files are open simultaneously and requiring a PAGE BUFFER routine. These are called **CXRPBFnn** (nn = 01,02,...) and are allocated in sequence. New PAGE BUFFER routines may be generated by "cloning" and re-compiling module **CXRPBF01**, changing its last two digits to new successive values.

69 **NO PAGE BUFFER FOR LINE IN 'HOLD' STATUS**

The report is in **HOLD** status but no PAGE BUFFER has been allocated to it. This would indicate another serious error condition earlier than this point.

81 **REPORT NOT INITIATED ON TERMINATE**

A TERMINATE was executed when the report was not in "INITIATEd" state. The statement is ignored.

91 **NOT ALL REPORTS FOR FILE WERE TERMINATED ON CLOSE**

An attempt is being made to close a file for which one or more associated reports are still in an "initiated" state. The CLOSE is actioned but an error will occur if any of those reports is subsequently TERMINATEd.

92 **FILE ALREADY CLOSED**

A CLOSE has been actioned when the file was not in the "OPENed" state. The CLOSE is ignored.

In addition to the above, individual file handlers may display values and messages of their own, in particular:

CRFHmode ERROR: LINE TOO LONG - TRUNCATED

which indicates a corruption to the print line's two-byte header.

Report Writer FUNCTION Errors

These errors are issued by the run time component of a FUNCTION. They always begin with:

function-routine-name: ERROR

followed by the text of the message:

REPORT FIELD OF WRONG LENGTH: n

means that the size of the report field is outside the permitted limits, such as when a printed DATE has less than six characters.

GIVEN DATE HAS INCORRECT PACKED FORMAT

means that the date parameter to the function, which should have the *COMP-3* format **YYDDDs**, is not in this packed form.

STYLE Errors

These are issued by the STYLE handler **CXRSTYLE**. They all indicate errors in the implementation of the STYLE clause at run time. They always begin with:

CXRSTYLE:ERROR n

followed by the text, depending on the value of n:

1 **ONLY UNDERLINE AND HIGHLIGHT POSSIBLE ON IMPACT PRINTER**

A STYLE other than **UNDERLINE** and **HIGHLIGHT** has been defined but the TYPE of printer is not an IBM 3800 Laser Printer or compatible. Only these two STYLEs can be implemented on an "impact" printer.

2 **STYLES NESTED OTHER THAN WHEN JUST ONE IS UNDERLINE**

Nesting of STYLEs, though syntactically permitted, can only work on a mainframe printer when UNDERLINE is nested with just one of the others (HIGHLIGHT, ALT-FONT, GRAPHIC).

3 **UNNECESSARY CALL TO STYLE ROUTINE**

This warning message is issued when a print line passed to the STYLE handler is found not to contain any STYLE *escape sequences* at all.

4 **UNRECOGNIZED STYLE**

One of the STYLEs in the print line is not one of NORMAL, UNDERLINE, HIGHLIGHT, ALT-FONT, or GRAPHIC and so cannot be processed.

5 **INCOMPLETE STYLE SEQUENCE**

The input record was exhausted before the end of the *escape* sequence.

6 **MATCHING END-STYLE NOT FOUND**

Every *escape* sequence that begins a STYLE must pair with an escape sequence to **end** it. When the file was closed, at least one of the former was still unpaired.

7 **END-STYLE FOUND WITHOUT PREVIOUS START-STYLE**

An **ending** escape sequence was encountered without having first had the **starting** sequence.

Other Run Time Errors

Several other messages can be issued by run time routines. These normally signal only very rare conditions caused by corruption of the program. The message always begins with the name of the routine and can therefore be found and understood in the source of the routine in question.

Index

Click on page numbers