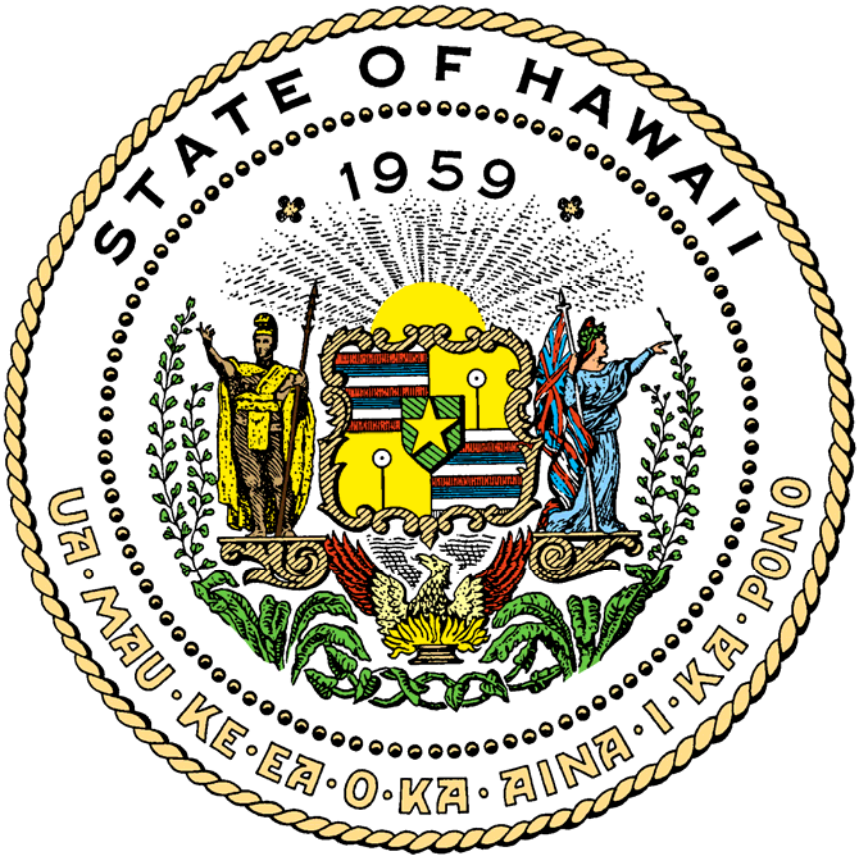

COBOL VS STANDARDS AND CONVENTIONS

July 2005

TABLE OF CONTENTS

1 INTRODUCTION

COBOL FOR MVS is the official application system language supported at the State of Hawaii Executive Branch's central computer site. Since 2001, COBOL FOR OS/390, COBOL FOR MVS AND VM became IBM'S current COBOL language products, and they replace both COBOL/370 and VS COBOL II which will not be supported.

This document assumes that the reader is at least familiar with VS COBOL II and would like to learn and understand the COBOL FOR MVS language standards, conventions, procedures, and/or guidelines that must be understood and observed when a COBOL application program is intended to run on the State of Hawaii mainframe computer.

This document is intended to be used by application Data Processing Systems Analysts (DPSA) and Computer Programmers (CP) during both the development, design, and construction of new applications; and for application program maintenance, enhancement, or update of existing application programs. *Permissive-type* words such as "should", "avoid", "minimize", "try", or "encouraged" are included in this document.

NOTE: "Permissive-type" words are intended to provide guidance for people who are working with **existing** programs and who are only modifying or enhancing existing program logic or code. However, for the development of **any new** program or application, these permissive-type words shall be understood to be interpreted in their absolute UNCONDITIONAL sense of "will", "DO not", or "shall".

1.1 Purpose

The primary purpose of this document is to provide a common basis for the development, design, construction, installation, and implementation of standardized application systems. The programs within these systems should then be uniform and consistent in structure, style, development, and content.

These standards and conventions should result in application programs that would be designed so that the resulting program language source codes are prose-like descriptive, consistent-in-form, self-documenting, readable, and easy to modify, maintain, revise or update by any maintenance application computer programming personnel.

1.2 Scope

This document, "COBOL FOR MVS Standards" is organized into three sections. The first, "COBOL Design Standards", is for data processing systems analysts (DPSA). The second, "COBOL Coding Standards", is for application computer

programmers (CP). And the third, "COBOL Environments", is to describe the available in-house COBOL utility operating system procedures that should be used to execute COBOL applications at the Executive Branch's central computer site, in either batch or on-line environments.

1.3 Applicability

The definitions for the terms such as "policy," "procedure," "standard," " The standards, conventions, procedures, and guidelines presented in this document must be followed by the Information and Communication Services Division (ICSD) computer programmers and data processing systems analysts, by any State agency with data processing personnel, and by any consultant, vendor, or contractor working for the State who will be using the State's computing resources at the State Executive Branch's central computing site.

1.4 Mainframe Computer Processing

Computer applications for the State of Hawaii Executive Branch should be designed for and developed with COBOL for MVS. The mainframe computer processing system for the State of Hawaii Executive Branch is an IBM 9672-RC5 system that runs in "LPAR" (logical partition) mode with separate designated logical OS/390 operating systems for each of the following host nodes:

a. Node-A is primarily used for online access to PRODUCTION data through CICS/VS and ADABAS regions. It is also used for production batch processing.

b. Node-B is primarily used for online access to TEST data through CICS/VS and ADABAS regions. It is also used for TSO (Time Sharing Option), and for test or production batch processing.

c. Node-E is primarily used for online access of both PRODUCTION and TEST data through CICS/VS and DB2, and also for test or production batch processing for the Department of Education.

1.5 Mainframe Production Job Management

Any application production job stream shall (as its last job step) execute a program like "PGM=L1PGG40L" so that a "message in a box" will appear on the job-log report. This message will allow mainframe production control personnel to know that the job named in the parameter for the last executed step has

"completed successfully."

Example:

```
//STEP99    EXEC  PGM=L1PGG40L,COND=(02,LT),
//            PARM='XASA1'
//STEPLIB   DD    DSN=EDPD.LINKLIBP,
//            DISP=SHR
```

1.6 Comments and Suggestions

Any State of Hawaii Information Technology Standards document, reference manual or users guide mentioned in this document are available through the departmental user agency data processing coordinator (DP Coordinator). Standards are also accessible on-line by clicking on Information Technology Standards on the ICSD home page at:

http://www.hawaii.gov/icsd/

Statewide Forms are accessible on-line by clicking on Forms Central on the Government in Hawaii home page at:

http://www.ehawaiigov.org/government/html/

Comments, recommendations, proposals, or suggestions regarding the contents of this document may be sent either via email to icsd.admin.ppmo@hawaii.gov or in writing to:

Information and Communication Services Division
Project Planning and Management Office
1151 Punchbowl Street, B10
Honolulu, Hawaii 96813-3024

2 COBOL DESIGN STANDARDS

The standard User Application programming language at the State's central computer site is IBM COBOL FOR MVS. Occasionally, a program, subroutine, or macro may be developed in other languages, but only if the other language is more suited for addressing specific or unique application design issues, problems or constraints.

These COBOL standards are meant to serve as a statewide convention and guideline for application systems design. Prime consideration is given to the issues of program structure, organization, ease of maintenance, and compiler efficiency.

These standards and guidelines will also establish a unified approach for developing the design of COBOL FOR MVS application programs. In any case, these design standards will be followed by contractors, vendors, and consultants who will be designing application systems for the State of Hawaii Executive Branch.

These design, development, and construction standards will be used to orient new Data Processing Systems Analysts hired by the State. Good program design practices and established standards should always be understood and followed.

This section presents specific conventions and guidelines for improving COBOL program performance. It is written primarily for IBM OS/390 COBOL data processing systems analysts who have a good understanding of COBOL language syntax and structure.

2.1 IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION provides the pertinent information concerning the COBOL application program design development.

2.1.1 PROGRAM-ID

The eight (8) character systems analyst assigned mnemonic name should conform to the Statewide Information Technology (IT) standard naming conventions discussed in the Program Naming Convention section of the statewide Information Technology OS/MVS JCL Standards.

A brief one-line comment statement summarizing the major purpose or function of the program follows this line. The detail program description will follow in the Remarks Section.

Example:

```
       PROGRAM-ID.  XLSA1A1L.
*      EXTRACT PREVIOUS 12 MONTHS XLSA TRANSACTIONS.
```

2.1.2 AUTHOR

The full name of both the designer Data Processing Systems Analyst's and the developer Computer Programmer's names will be included here.

Contractors, Vendors, and Consultants will indicate the firm's name, and

the names of their Designer and Developer names should also appear here.

Example-1:

```
AUTHOR.    ANALYST:  ANDREW A ANDERSON
           PROGRAMMER:  PHILIP P PETERS.
```

Example-2:

```
AUTHOR.    ANALYST:  ASHLEY A ATOZ
           PROGRAMMER:  PAT P POTTERS
       CREATIVE APPLICATIONS INTEGRATORS.
```

2.1.3 REMARKS

This section will provide a brief narrative of the program's purpose and functions; and a summary description and dates of all revisions made subsequent to the implementation. This section will contain the following subsections:

a. Program Abstract

A brief description of the main program function or purpose, usually 3 to 5 sentences. If the program is part of a system of programs, then also explain how the program fits into the system. The following should be included:

If subroutine must be called by the program, list the names of subroutines.

Example:

```
*      SUBROUTINES CALLED ARE:  "CANCEL", "GREGRN".
```

b. File Descriptions

Enter the name of each file used by the program indicating whether it is input, output or work file (I/O) and its file organization (PHYSICALLY SEQUENTIAL, VSAM)

Indicate position and length of sort key fields that were used to control the record order sequence, provide description of key word fields, and briefly describe the source and purpose of any internal or external files.

c. Included Text

Indicate Panvalet include names when ++INCLUDE will be used to copy record layouts, processing structures, members, or subroutines.

2.2 ENVIRONMENT DIVISION

The function of this division is to define and specify the hardware configuration and requirements of the program.

2.2.1 CONFIGURATION SECTION

In COBOL for MVS, the use of the Configuration Section is optional.

2.2.2 INPUT-OUTPUT SECTION

a. Programs will not use three (3) or more tape drives concurrently (for both INPUT and OUTPUT for a job).

Any exception, at any time, requires the approval of the ICSD Production Services Branch (PSB) computer operation's scheduler or the computer operator shift supervisor, and must be scheduled to assure the availability of the tape drives.

NOTE:

No program will use the computer console as a typewriter for I/O messages that would need any computer operator to key-enter a response.

Example:

'ACCEPT FROM CONSOLE'

b. The body of all batch programs will have the 'DISPLAY' statement:

'DISPLAY ..... UPON CONSOLE'

to send AT LEAST one message either at the beginning, or at the ending of the program to inform the ICSD PSB computer operator or control personnel of the status of the program.

Example:

```
DISPLAY     ' ***** PROGRAM-NAME:  ' PROG-ID,  '— ',
..... 'NORMAL TERMINATION ****' UPON CONSOLE.
```

c. Programs should be designed to be hierarchical, structured, modular, and perform only one function. Avoid the use of the 'GO TO' command. When the 'GO TO' command is needed, it should always transfer control in a top-down manner
NOTE:

Any exception to the above 'DISPLAY' statement must have the PRIOR approval of the DAGS-ICSD-PSB Branch Manager.

2.2.2.1 FILE-CONTROL

The FILE-CONTROL contains the SELECT Statement that is used to link the COBOL application program's I/O commands to the physical I/O data devices.

The following are considerations for the use of reserve words in the SELECT statement:

a. The "ASSIGN TO" DDNAME (DATA DEFINITION NAME) must follow the statewide Information Technology (IT) Program Naming Convention format.

b. For the "ASSIGN" clause, device independence is required. Programmers should not assign data sets to particular devices. Actual assignments should be made via Job Control Language (JCL) statements at execution time.

c. The "RESERVE" clause should not be specified in the program. The assignment should be made at execution time via JCL DD (DATA DEFINITION) statements.

2.2.2.2 I-O-CONTROL

The I-O-CONTROL statement contains two reserve words whose use should be restricted.

The RERUN paragraph must not be used (the job statement checkpoint subroutine may be used).

The "APPLY" clause should only be used with OCCURS... DEPENDING specified data. It may also be used with the approval of the project leader.

2.3 DATA DIVISION

The Data Division describes the attributes, features and functions of each file, record, and data element used in the program.

THE use of the "COPY" Statement is not allowed. The ++INCLUDE command from the ICSD acquired utility software product, Panvalet, will achieve the same result.

Data division members for all records used in more than one program should be created and placed IN the Panvalet test library, and included in application programs when required. The project manager is responsible for the creation of the Panvalet include modules.

2.3.1 FILE SECTION

The "FILE SECTION" Contains a description of all externally stored data as well as each Sort-Merge-File used in the program.

2.3.1.1 Record Description Entries

Each record description will contain the following entries:

BLOCK CONTAINS. Use the phrase "BLOCK CONTAINS 0 RECORDS" so that the number of physical records in a logical block will come from the JCL Data Control Block (DCB). The exception may be for keyed-sequenced VSAM files where the number of records per block could be specified.

RECORD CONTAINS. This entry must not conflict with the LOGICAL record size defined in the JCL DCB.

LABEL RECORD. Use the phrase “LABEL RECORDS ARE STANDARD” to eliminate an operating system warning statement.

RECORDING MODE. Use the phrase “RECORDING MODE IS F” to specify that there are fixed blocks of data in the file. The specification for this parameter may come from the job control language (JCL) data set Access Control Block (ACB) specification.

IMPORTANT NOTES:

a. The ICSD has a standard skeleton COBOL FOR MVS program in an IBM-TSO shared partitioned dataset named (COBOLMVS). This model template source program has the data layout storage descriptions for a program that will produce a printed control report.

b. In the FILE SECTION, each file data record picture description will have only one “FILLER” statement to reflect the total logical record length. The exception is when a keyed-definition record is being defined, then appropriate field names for the key data, and the RECORD STATUS should be specified in this SECTION.

c. Sequential files should have fairly large block sizes containing a number of logical records up to a maximum size of 32,768 bytes.

   The COBOL compiler’s de-blocking routine to get logical records from the larger physical block-size record area is very fast, and it does not require the massive number of housekeeping instructions that would be required to get a physical block-size logical record for each I/O data request.

   Large blocks save I/O channel time and improves peripheral storage capacity, but requires more Virtual Storage for Buffer Management.

   The large physical block-size records may substantially increase the total region size for the program execution.

d. For VSAM files, the recommended VSAM file Control Internal (CI) size

is 4,096 bytes.

e. The CI size value up to 8,192 must be a multiple of 512. The CI size value may be a number up to 32,768 BYTES, however, when the CI SIZE value that is greater than 8,192 BYTES, it must be a multiple of 2,048.

f. For VSAM files that are used mainly for random retrieval, use CI sizes of 4,096 or less.

g. For VSAM files used mainly for sequential processing, use larger CI sizes.

h. For VSAM files, be sure that you have requested the IMBED option. It is similar to the APPLY CORE-INDEX option, because it causes the index to be resident.

2.3.1.2 APPLY WRITE-ONLY

Specify APPLY WRITE-ONLY in the Environment Division for the output of varying blocked records. This cuts short output blocks when the remaining space would be too small for the maximum number of occurrences in an OCCURS record.

For APPLY WRITE-ONLY. In the output file layout, specify the data-name of another elementary level item as the DEPENDING object. Do not specify the DEPENDING field in the record itself that has the OCCURS.

2.3.1.3 PICTURE

The abbreviated “PIC” form is preferred. The multiplying factors for the picture length should be at least two digits. When a logical grouping of ten or more items is required, the PIC clause and its associated characteristics should be vertically aligned to facilitate summing.

2.3.2 WORKING STORAGE SECTION

The Working Storage section contains data description entries for non-contiguous data items and/or records.

2.3.2.1 WORKING-STORAGE CONVENTIONS

Rules for designing DATA RECORDS as stated in the FILE SECTION apply here unless specifically restricted.

a. "77 level" entries *will not* be used. Instead, elementary items such as counters, indicators, subscripts, switches, etc., will be grouped together under 01, 05, or 10 levels

b. "88 level" condition names should be used to describe the tested conditions. Names should be meaningful to make the application readable and "self-documenting".

c. All data items should be designed to be initialized with a VALUE clause wherever possible. The exception are variables whose values will come from the special registers.

d. Numerical data that will not be used in calculations, like ZIP-CODE, should be defined as Alphanumeric, PIC X(05).

e. Specify the same usage (COMP or COMP-3) for numeric items which interact in moves, compares, and arithmetic statements.

f. For added and subtracted items, try to specify the same number of decimal places.

2.3.2.2 WORKING-STORAGE Organization

a. Fixed length group items or tables are limited to 131,071 bytes.

b. Varying length tables must not exceed 32,767 bytes.

c. Working-Storage data entries will be categorized into major groupings (01, 05, or 10 levels). The names assigned for a similar function in the major grouping should be designed to begin with the same prefix.

For example:

all counters have prefix of *"CNT"*;
any accumulator have prefix of *"CUM"*;

any indices have prefix of *"INDX"*;
any message have prefix of *"MSG"*; etc.

2.3.2.3 ERROR-SWITCH

Use the following procedure to handle exception conditions in your program. Include an error switch labeled ERROR-SW of one byte and PROG-ID of eight bytes to the program. ERROR-SW will be initially set to zero. PROG-ID will contain the program's Program-ID Number.

2.3.2.4 COMPUTATIONAL

COMP is the best format for items used as subscripts, in extensive integer arithmetic, or as OCCURS...DEPENDING objects. Conversion to COMP will automatically occur for subscripts when COMP is not specified.

COMP results in the fastest arithmetic instructions.

a. Make COMP items less than five digits if possible. One to four digits require two bytes; five to nine digits require four bytes; more than nine require eight bytes and more complex machine instruction sequences.

b. Avoid using more than 15 digits. Machine instructions cannot handle over 15 digits, so expensive subroutines are needed to process larger items.

   Maximum decimal value for a PIC S9(04) COMP is 65,535.

   Maximum decimal value for a PIC S9(05) COMP-3 is 99,999.

c. For numeric fields larger than 6 digits, do not use COMP, instead use COMP-3 because COMP-3 values are easier to read in an error ABEND dump listing.

   For COMP-3, each storage byte holds two digits for efficient use of space in memory and on peripheral storage. Efficient machine instructions process COMP-3 data directly; other formats often require conversion to COMP-3.

d. Specify “S” in PICTURE for COMP and COMP-3 items, otherwise an extra instruction is needed to remove the sign whenever the value is modified. Specify an odd number of digits for the field length.

e. Except where COMP is best (see above), COMP-3 is the preferred format for all numeric items.

f. Specify SYNC for COMP items to align the elementary item on a proper storage boundary. This will ensure efficiency when performing arithmetic operations.

   The SYNC clause is encouraged for any computational items and may appear only at the elementary level. If used, performance is improved when performing arithmetic operations. But this may increase virtual storage and record size requirements.

2.3.2.5 OCCURS

The purpose of this facility is to conserve I/O channel time and peripheral space by carrying only the meaningful occurrences in a variable-length list of items, rather than always carrying the maximum number.

For sequential access of VARYING RECORD SIZES IN A file, the compiler must generate generalized code to handle all possible record and group lengths. Use OCCURS...DEPENDING with care.

IMPORTANT NOTE:

OCCURS...DEPENDING is extremely costly in CPU time.

Every time the DEPENDING variable is modified (via READ, WRITE, MOVE, or calculation), time-consuming routines are needed to compute the current length of groups and the location of variably located fields.

Variable-length records must be written in large blocks. Care is needed to assure efficient use of output buffers and DASD storage.

Use only one OCCURS...DEPENDING statement in the overall record description. The overhead from "nested" OCCURS...DEPENDING is much greater than in a simple use of the “OCCURS”.

Put the OCCURS...DEPENDING data at the end of the record, preceded by all fixed data. Extra overhead is entailed in accessing any fixed data following the variable data, since its relative position is also variable.

2.3.2.6 REDEFINES Clause

All redefining should refer to the originally defined statement regardless of the number of redefinitions.

2.3.2.7 RENAMES Clause

Do not use the “renames” clause. The REDEFINES clause can be used to get the same result.

2.3.2.8 USAGE Clause

Internal elementary numeric items should be defined with the appropriate computational form whenever the primary use is in arithmetic operations.

If arithmetic is not performed on a numeric item, it should be defined to have an alphanumeric “X” picture.

If frequent arithmetic is performed on a numeric item, it should have a usage of COMP-3 with an odd number of digit positions or other forms of COMP as appropriate.

2.3.2.9 VALUE Clause

The VALUE clause should not be stated within the FILE SECTION.

Numeric items should always be signed unless an absolute value is necessary.

2.3.3 LINKAGE SECTION

The LINKAGE SECTION is used to describe data made available from another program.

Data item description entries in the LINKAGE SECTION provide names and description, but storage within the program is not reserved since the data area actually exists elsewhere.

Heavily used data parameters that are passed via the LINKAGE SECTION should be moved to WORKING-STORAGE variables before these data-elements are referenced in the processing logic.

CICS/VS interface and storage control statements are found in the LINKAGE SECTION.

Example:

```
LINKAGE SECTION.

    01    DFHCOMMAREA    PIC X(010).
    01    DB-PCB1        PIC X(100).
    01    DB-PCB2        PIC X(100).
```

2.4 PROCEDURE DIVISION

The Systems Analyst responsible for the program project must follow structured programming design techniques. It is recommended that a 3-step process transition from a data-flow-diagram, to a functional hierarchical structured chart, and ultimately to a pseudo code of the logical algorithm. The pseudo code is the foundation of the program solution. The details for the program comes from expanding the information defined in the requested system's specifications, see Appendix-A. The needed functions for the application's solution are expanded into the appropriate program syntax structures.

3 COBOL CODING STANDARDS

The standard programming language for the Executive Branch's central computer site installation is IBM COBOL FOR MVS. Occasionally, a program, subroutine, or macro may be developed in other languages, but only if the other language is more suited to the problem.

These coding standards are meant to serve as a statewide guideline to improve

program quality and programmer productivity. Prime consideration is given to the issues of program readability, understanding, ease of maintenance, ease of debugging, and program efficiency.

These standards and guidelines will provide for uniformity in the development of reliable COBOL application programs.

Good programming practices and established standards should always be followed. Changes to improve efficiency should never obscure basic program logic.

This section has specific conventions and guidelines for improving COBOL program performance. It is written primarily for IBM OS/390 COBOL Computer Programmers who have a good working knowledge of COBOL language grammar, syntax, and structure.

The standard application development approach includes structured design and programming with the strongly recommended use of a basic standardized skeleton program like the one stored in EDPD.PANVTEST as XASA1A1HA1 (see Appendix-B).

As a Supplement, the IBM COBOL Compiler and Library Programmer's Guide's chapter on "PROGRAMMING TECHNIQUES" offers IBM's recommended techniques for increasing the efficiency of COBOL programs. These recommendations may be followed when they do not conflict with the State's standards.

IMPORTANT DEBUGGING FACILITIES NOTES:

> The STATE, FLOW, ENDJOB, DYNAM, COUNT, SYST, TEST, and SYMDMP parameter options and READY TRACE and EXHIBIT statements should be used only in debugging.
>
> These facilities add substantial overhead to the program and may be very costly in both memory space and CPU time. (All DEBUG OPTIONS, except STATE consume execution time while the program is running, even if no TRACE is taken, and even if the program does not ABEND.)
>
> Under no circumstances are any of these Debugging Facilities to be specified for CICS/VS programs. Consult the CICS/VS Application Programmer's Reference Manual for Translator Options for COBOL under CICS/VS, and the COBOL CONSIDERATIONS section of this document.

3.1 IDENTIFICATION DIVISION

The Identification Division provides for the capture of all pertinent information concerning the program.

3.1.1 PROGRAM-ID

The eight character assigned mnemonic conforming to the standard naming conventions will be provided by the Data Processing System Analyst.

A one-line comment statement summarizing the major purpose or function of the program follows this line.
Example:

```
 PROGRAM-ID. XASA1A1L.
*        EXTRACT LAST TWELVE MONTHS TRAINING
TRANSACTIONS.
```

3.1.2 AUTHOR

Both data processing systems analyst and computer programmer's names will appear here. Contractors and consultants will also indicate the firm's name here.

3.1.3 INSTALLATION

The use of this command is OPTIONAL.

3.1.4 DATE-WRITTEN

Use format of "February 22, 2005" to reflect the date when the coding was originally developed and coded.

3.1.5 DATE- COMPILED

Enter only paragraph name because the system will insert the current date at compile time.

3.1.6 REMARKS

This section will provide a brief narrative of the program and a summary of all revisions made subsequent to the initial implementation. This section will contain the following subsections:

a. Program Abstract;

A brief description of the main program logic, keep it short, from 3 to 5 sentences. If the program is part of a system of programs, then explain how the program fits into the system. The following should also be included:

i. If the SPECIAL-NAMES paragraph (except for top-of-page) or I-O control paragraph is used, it should be mentioned here to draw attention to its presence.
ii. If subroutine calls are used in the program, list the names and functions of subroutines.

Example:

EXTERNAL SUBROUTINES CALLED: "CANCEL", "GREGRN".

- Explain the routine's structure and use of any table.
- Explain when and how the routine should be used.

. Establish the minimum requirements for the protection of the State's physical assets as they relate to the misuse or loss of computer hardware or equipment;

b. File Descriptions

Enter the name of each file used by the program indicating whether it is input, output or work-file (I/O) and its file organization (PS, VSAM)

i. If the SPECIAL-NAMES paragraph (except for top-of-page) or I-O control paragraph is used, it should be mentioned here to draw attention to its presence.

ii. Indicate Panvalet include names if ++INCLUDE command is used.

c. Program Modification

Any revision and modification to the program should be noted. Data should include change request number, date change implemented, programmer's name, and summary description of the change.

EXAMPLE of IDENTIFICATION DIVISION:

```
IDENTIFICATION DIVISION.

PROGRAM-ID. XASA1C1L.

*          ICSD TRAINING EVALUATION REPORTS

INSTALLATION.  STATE OF HAWAII, DAGS-ICSD.

AUTHOR.        PROGRAMMER:    RONALD O. WHITE;
ANALYST:       HARRY I. SMITH.

DATE-WRITTEN. MARCH 3, 2001.

DATE-COMPILED.

*REMARKS.      1.     GENERATES ANNUAL TRAINING STATISTICS
*                            FROM MASTER ATTENDANCE FILE.

*              2.     PRODUCES  THE FOLLOWING REPORTS:
*                     a.     XASA1C1R - DETAIL COURSE STATISTICS.

*                     b.     XASA1C2R - SUMMARY BY COURSES.
*                     c.     XASA1C3R - DETAIL DEPT. STATISTICS.
*                     d.     XASA1C4R - DETAIL STUDENT STATISTICS.

*              3.     TERMINATE JOB IF INPUT IS OUT OF SEQUENCE.


*              4.     SORT ORDER:    01,41,CH,A   SORT KEY
*                                    66,02,CH,A   TRAN YEAR
*                                    62,04,CH,A   TRAN MONTH-
DAY
```

```
*          5.    COURSE CODE AND DESCRIPTIONS MAXIMUM
*                OF 90 COURSES.

*          6.    DEPT CODE AND NAME FOR 45 AGENCIES.

*        MODIFIED 09-11-2001 TO ADD SPECIAL DEPARTMENTAL
*                STAFF PORTFOLIO REPORT-XASA1C5R.
```

3.2 ENVIRONMENT DIVISION

The function of the Environment Division is to define and specify the hardware configuration and requirements of the program.

3.2.1 CONFIGURATION Section

The SOURCE-COMPUTER and OBJECT-COMPUTER paragraph are optional. If you want to identify the computer, specify IBM-OS-390.

The use of the SPECIAL-NAMES paragraph for any purpose other than for “TOP-OF-PAGE” or “NEW-PAGE”, is discouraged.

Do not use SPECIAL-NAMES to rename input/output devices, because the function names are already as explicit as any mnemonic.

IMPORTANT NOTE:

DO NOT USE COBOL REPORT WRITER COMMAND STATEMENTS. THE COBOL FOR MVS AND VS COBOL II DO NOT DIRECTLY SUPPORT THE REPORT WRITER FEATURE.

3.2.2 INPUT-OUTPUT Section

a. No program will use more than three (3) magnetic tape cartridge drives (I/O) concurrently for any length of time.

b. Any exception for more tape cartridge drives requires the approval of the ICSD PSB Computer Operations Scheduler or Shift Supervisor, and must be scheduled to ensure the general availability of the tape drives.

c. No program will use the console typewriter for any I/O message that would require a computer operator to enter a response at the computer console.

d. All batch programs will use DISPLAY statements to send an Operator Console status message, using:

DISPLAY ..... UPON CONSOLE

The ‘DISPLAY ..... UPON CONSOLE’ sentence is used to send messages to indicate the beginning and/or ending of a program execution.

Example:

```
DISPLAY ' *** ', PROG-ID, ' *** ',
'NORMAL TERMINATION ***' UPON CONSOLE.
```

***NOTE:**

For any exceptions to the above, the specifications and designs must be approved by the ICSD PRODUCTION Services Branch (PSB).

3.2.2.1 FILE-CONTROL Paragraph

a. SELECT Statements

For readability, and to allow the flexibility for name changes, each SELECT statement will begin on a new line with the ASSIGN and other options indented on the following line under the file-name.

Example:

```
SELECT    PAY-XTRACT-FILE
          ASSIGN TO XLSA1A1D
          KEY IS XTRT-ID-KEY
          PASSWORD IS OPEN-OKAY.
```

b. File names

All file names will be descriptive and must end with the suffix word "-FILE" and indicate the file's primary purpose or function, such as, input, extract, output, work, sort, or the project's PMS code.

Example:

```
MSTR-IN-FILE        (Sample Input File Name)
MSTR-OUT-FILE       (Sample Output File Name)
TRAN-IN-FILE        (Sample Input File Name)
TRAN-ERROR-FILE     (Sample Output File Name)
OAB-XTRACT-FILE     (Sample PMS File Name)
```

c. ASSIGN clause

Device independence is encouraged. Programmers will not assign

data sets to particular devices.

d. RESERVE clause

Reserve clause should not be specified but instead assigned at execution time via JCL (job control language) DD statements.

e. FILE STATUS clause

File status must be defined in the Data Division for all VSAM files. It is used to monitor the successful execution of each I/O request of the VSAM file.

3.2.2.2 I-O-CONTROL Paragraph

a. The RERUN paragraph must not be used (the checkpoint subroutine is available).

b. The APPLY clause is discouraged except for use with OCCURS....DEPENDING. It may only be used with the approval of the project leader.

3.3 DATA DIVISION

This division describes each file, record, and data element used in the program.

The use of the 'COPY' statement is not allowed. The Panvalet ++INCLUDE command will achieve the same result.

The COBOL Report Writer Facility will not be used.

Data division members for all records used in more than one program should be designed and created in the project's library, copied into the PANVALET test library, and included in application program logic when required. The design and creation of any shared PANVALET include module is the project systems analyst's responsibility.

3.3.1 FILE SECTION

The File Section contains a description of all externally stored data as well as each Sort-Merge-File used in the program.

3.3.1.1 File Description Entries

a. The file-name is the computer programmer's descriptive mnemonic name for the data file.

   The file-name identifies the major function and purpose for the file. *DO NOT link the file-name to a physical device*

b. The file-name should have at least three portions. The last suffix word must be "-FILE". The other words must describe the application system purpose, the primary function, activity, or data source.

   Example:

   MSTR-TRAINING-FILE
   PAYROLL-MASTER-FILE
   PMS-XTRACT-MSTR-FILE

3.3.1.2 File Description Entries

Each record description will contain the following entries:

BLOCK CONTAINS. Use BLOCK CONTAINS 0 RECORDS for sequential files. The value for this parameter will be determined by the JCL at execution. For keyed-sequenced files, the actual number of records per block may be specified when the file is accessed randomly. For VSAM files, this statement is treated as a comment.

RECORD CONTAINS. This entry is optional. If the actual record size is larger than the total specified record length description, only the 01-level specified data length is made available to the computer operating system.

LABEL RECORD. This statement definition comes through JCL parameter. The omission of this phrase currently results in a compiler warning message, so it is recommended to add the

phrase:
"LABEL RECORD IS STANDARD".

DATA RECORD. The record description names must be related to the given file name for ease of program maintenance. The only difference in the names is the suffix name, "-FILE" is changed to "-RECORD" OR "-RECD".

Example:

```
FD    MSTR-XTRT-FILE
      BLOCK CONTAINS 0 RECORDS
                .
                .
                .
      DATA RECORD IS MSTR-XTRT-RECD.
```

IMPORTANT NOTES:

a. The ICSD standard skeleton COBOL FOR MVS program has the DATA STORAGE date-element field description for a model template program that will produce a printed control report.

b. The data record picture description will use only one “FILLER” statement to reflect the total logical record length. The exception is when a KEYED-DEFINITION record is being defined, then appropriate field names for the KEY data and RECORD STATUS may be placed in this SECTION.

c. Sequential files should have fairly large block sizes containing a number of logical records up to a maximum size of 32,768 bytes.

   The COBOL COMPILER’S de-blocking routine to get logical records from the physical BLOCK-SIZE is fast and does not require the massive number of housekeeping instructions required to get a new physical block of data.

   Large blocks save I/O channel time and improves peripheral storage capacity, but requires more Virtual Storage for Buffer Management.

The large physical block-size record may substantially increase the total region size for the program execution.

d. FOR VSAM FILES, be sure that you have requested the IMBED option. It is similar to the APPLY CORE-INDEX option, it causes the index to be resident.

3.3.2 WORKING-STORAGE SECTION

This section contains data description entries for non-contiguous data items and/or records.

3.3.2.1 WORKING-STORAGE Conventions

Rules for recording data as stated in the FILE SECTION apply here unless specifically restricted.

a. 77 level entries must be removed. Elementary items such as counters, indicators, subscripts, switches, etc., should be grouped together under 01, 05, or 10 levels.

b. 88 level condition names should be used to describe and test conditions. Names should be meaningful. See example in Switches after WORKING-STORAGE ORGANIZATION.

c. All data items including constants, should be initialized with a VALUE clause wherever a value is needed.

d. Continuation of literals is not allowed. The entire literal should be coded on one line, or split into two or more value lines. Careful alignment helps reduce checkout time and produces preferable documentation.

e. All fields to be used in numerical calculations should be signed, (packed) COMP-3, and have an odd number of digits.

f. Numerical data that will not be used in calculations should be defined as Alphanumeric, for example: ZIP-CODE PIC X(05).

g. Arithmetic fields must not be in display mode unless it is part of an I/O record.

h. Eliminate any records or independent items in Working-Storage that are not referenced in the cross-reference listing.

i. Try to specify the same usage (COMP or COMP-3) for numeric items which interact in moves, compares, and arithmetic statements.

j. For added and subtracted items, specify the same number of decimal places when possible.

k. Use the variable FILLER to define any unreferenced data field areas.

3.3.2.2 WORKING-STORAGE Organization

a. The first and last entry for the Working-Storage Section must contain a descriptive literal message to define where WORKING-STORAGE Begins and Ends to provide the programmer with a debugging aid to locate data assigned to variables within the WORKING-STORAGE.

Example:

```
WORKING-STORAGE SECTION.
01      WS-MESSAGES.
        05      FILLER          PIC X(32)    VALUE
                        'PGM=XLSA1A1L, WS BEGINS HERE'.
                .
                .
        05      FILLER          PIC X(32)    VALUE
'PGM=XLSA1A1L, WS ENDS HERE'.
```

b. Place the high activity data at the beginning of each group of the WORKING-STORAGE SECTION variables.

c. If possible, start all PICTURE clauses in column 42. When a long VALUE clause of more than 12 characters or attribute continuations necessitated by a very long PICTURE expansion in column 32; and for short PICTURE attributes, place COMP-3, COMP, or short VALUES beginning in column 52.

Example:

```
05    ONE          PIC S9(05)   VALUE +00001 COMP-3.
05    FICA-RATE    PIC S9(01)V9(04)VALUE +0585
                                              COMP-3.
```

d. For the ease of program checkout and maintenance, it is suggested that WORKING-STORAGE data entries be categorized into major groupings (01, 05, or 10 levels) whenever possible or practical.

   The names assigned to the fields in a major grouping should all begin with the same functional descriptive prefix. The fields in each major group should be alphabetized. But fields within major groups that may be printed should be listed in the same order as the output.

EXAMPLE OF WORKING-STORAGE COUNTERS AND ACCUMULATORS:

```
01    ACCUMULATORS.
      05    CUM-AMT-PAID          PIC S9(7) COMP-3.
      05    CUM-INTEREST          PIC S9(7) COMP-3.
      05    CUM-SUBTOTAL          PIC S9(9) COMP-3.
      05    CUM-MON-TOTAL         PIC S9(9) COMP-3.

01    COUNTERS.
      05    CNT-CLIENT-TYPES      PIC S9(7) COMP-3.
      05    CNT-MASTER-READ       PIC S9(7) COMP-3.
      05    CNT-REJECT-RECD       PIC S9(7) COMP-3.
      05    CNT-SELECTED          PIC S9(7) COMP-3.
      05    CNT-TRANS-READ        PIC S9(7) COMP-3.
      05    CNT-WRITE-MSTR        PIC S9(7) COMP-3.
```

.

e. Some recommended major groupings are as follows:

   i. Input/Output Areas. This entry would include the Working-Storage copies of the record layouts of files being processed.

   ii. Control Fields. Areas used for data being sequenced or for processing logic controls.

   iii. Constants. Data names to minimize compiler

conversions of literals to internal temporary storage target attributes. Avoid the use of literals in the body of the program.

iv. Counters. To include all accumulators and audit counters used for numeric operations.

v. Literals. By assigning a variable name to a literal, the use of the literal in the procedure division is easily located by the cross-reference listing. Also, changes to a literal can be made in a central location (DATA DIVISION) and thus eliminates the need to change the literal in the PROCEDURE DIVISION.

vi. Messages. To simplify changes for descriptors; or to define all program logic error messages and their literal text. Assign a control number and a consistent label for all error messages and other status messages.

Example:

```
10    MSG4-ERR-IN-SEQUENCE        PIC X(42)  VALUE
        'TRANSACTION OUT OF SEQUENCE-ABORT'.
```

vii. Switches. The use of switches and flags will be kept at a minimum. Decision switches should have the value of "0" or “NAY” to indicate off-condition and "1" or “YES” for an on-condition.

Names should be meaningful, self-documenting, and related to the condition being tested.

The “88-level” condition data names will be used when defining all logical switches.

The "88" condition name will be prefixed with the elementary variable name that it is describing.

EXAMPLE-1:

```
01    TEST-SWITCHES.
      05    EOF-PAY-MSTR            PIC X(01)    VALUE
      ZERO.
            88    EOF-PAY-MSTR-ON                 VALUE '1'.
            88    EOF-PAY-MSTR-OFF                VALUE
      ZERO.
```

COBOL CODING:

```
IF    EOF-PAY-MSTR-ON
      PERFORM   800-CHK-LAST-PAY-MSTR
            THRU 800-CHK-LAST-PAY-MSTR-EXIT.
```

EXAMPLE-2:

```
01    TEST-SWITCHES.
      05    EMPLOYEE-STATUS         PIC X(01)    VALUE
      ZERO.
            88    EMPLOYEE-IS-NEW                 VALUE '1'.
            88    EMPLOYEE-IS-RETIRED             VALUE '2'.
            88    EMPLOYEE-UNCLASSIFIED           VALUE '9'.
```

COBOL CODING:

```
IF    EMPLOYEE-IS-NEW
      PERFORM   550-ADD-NEW-EMPLOYEE
            THRU 550-ADD-NEW-EMPLOYEE-EXIT.
```

viii. Pass-Areas. To contain global data elements that will be linked-to and passed-from another external program.

ix. Print Formats. Report headers, title lines, detail lines, total lines, etc.

x. Tables. To include standard tables, arrays, general definitions, etc. These may be included in the program logic via the ++INCLUDE Panvalet command statement.

3.3.2.3 Level Number

All level numbers must have two (2) digits.

Except for level 01 items, all level numbers should initially be assigned to values of set increments such as: 05, 10, 15. Additional level numbers may come about as a result of maintenance or requirement changes when deeply involved in checkout.

Level numbers should be indented in a consistent manner for each record. The COBOL coding form is designed in a manner which is convenient to indent four columns for each level, but to establish this indentation as a fixed standard may be unrealistic in a hierarchy that exceeds four or five levels.

Align all level specifications and attributes of the same numeric-level rank in the same column.

3.3.2.4 PICTURE

The PIC form is preferred. The PIC clause and the associated characteristics should be grouped and aligned in a method to facilitate summing. The following illustration requires a bit more clerical effort but has an excellent payoff during checkout and maintenance.

```
05    FLD-A          PIC   X(02).
05    FLD-B          PIC   9(10).
05    FLD-C          PIC   X(22).
05    NUM-FLD        PIC S9(05)V99.
```

For easy reference, PICTURE clauses will be aligned vertically whenever possible beginning in column 42.

When a REDEFINES clause is necessary, it should begin in column 32, and the PICTURE statements within the redefining area should

be under the REDEFINES.

But the key point is "*consistency*" to visually associate COBOL keywords. Use the following formats for attribute descriptions:

- Alphabetic display fields as X(nn);
- Integer numeric fields as 9(nn);
- Decimal fields as S9(nn)v9(nn).
  However, when the decimal fraction portion of a field is less than 3 positions, use "99" or "9" instead of V9(02) and V9(01) respectively.
- Output suppression fields should be fully expanded to include editing characters:

  Example: PIC ZZZ,ZZZ.99

3.3.2.5 OCCURS

The OCCURS command is used to define relational table data elements or repeating groups of data elements.

Fixed length group items or tables in the working Storage Section or Linkage Section may be as long as 131,071 bytes.

Variable length tables must not exceed 32,767 bytes.

When an OCCURS clause is required, the word OCCURS should start in the same column as the word PIC and PIC should start on the next line.

Example:

```
01    TABLE-ITEM        OCCURS  10 TIMES
      PIC  X(05).
```

Use no more than one OCCURS...DEPENDING statement in the overall record description. The overhead from "nested" OCCURS...DEPENDING is much greater than in a simple fixed occurrence use.

Put the OCCURS...DEPENDING data at the end of the record, preceded by all fixed data. Extra overhead is entailed in accessing

any fixed data following the variable data, since its relative position is also variable.

When a program must update a file that has an OCCURS...DEPENDING record definition, the following technique is recommended.

The State's computer hardware and equipment include, but are not limited to, vendor supplied information processing equipment; work stations and terminals; personal computers; mainframe computer systems and mini-computer systems; servers; supporting peripheral equipment such as tape drives, disk drives, CD ROM drives and printers; networking routers, switches, hubs and connectivity equipment; and processing facilities.

a. Use READ...INTO (or READ the record-as-a-block and MOVE this block of data to a working-storage 01-variable-name to specify data-elements needed for processes) to get the OCCURS...DEPENDING data-elements for the actual processing logic.

b. In the Working-Storage record description, specify a simple OCCURS (*do not use DEPENDING*) with the maximum number of occurrences.

c. In the Procedure Division, refer only to the simple OCCURS data-names defined in Working-Storage.

d. Specify a level 01 COMP item to indicate the number of occurrences. If the number of occurrences changes, modify this item but not the DEPENDING object.

e. Do not directly change the DEPENDING object, because that invokes expensive recalculation of record and group sizes and relative positions, every time the value is modified.

f. At WRITE time move the level 01 item value to the DEPENDING object. Use WRITE...FROM to transfer the OCCURS...DEPENDING record from Working-Storage to the output file.

g. Using APPLY WRITE-ONLY for an output file in the Environment Division will cause the access method routines to cut short each output block whenever the remaining space would not be adequate to handle the maximum number of occurrences.

h. When using APPLY WRITE-ONLY for the output file description, specify the data-name of another 01 level item as the DEPENDING object - not the corresponding field in the record itself.

i. Be sure to specify COMP and SYNC for the DEPENDING object; or else a conversion to COMP is needed whenever the record, group lengths, and positions must be calculated.

j. Do not move groups or records containing variable-length data to or from the record description, except at READ and WRITE time.

3.3.2.6 VALUE Clause

The VALUE clause will not be specified within the FILE SECTION.

Numeric data items should always be signed unless an absolute value is needed.

3.3.2.7 USAGE

Do not code the phrase words: "USAGE IS".

3.3.2.8 COMPUTATIONAL

Internal elementary numeric items should be tagged with the appropriate computational form whenever their primary use is in arithmetic operations.

If arithmetic operations are not performed on a numeric data-item, it should have an alphanumeric picture, “PIC X(09)”.

If arithmetic operations are performed on a numeric item, it should have a usage of COMP-3 with an odd number of digit positions or COMP as appropriate.

COMP-3 data is easy to read in a dump listing. Specify S in the numeric picture definition for COMP-3 items, and specify an odd number of digits.

With COMP-3, each byte holds two digits for efficient use of space in memory and on peripheral storage. If the signed data field “PIC S9(07)” were specified as “COMP-3,” the data field length would be only 4 bytes, and not 7 bytes.

Avoid using more than 15 digits. Machine instructions cannot handle over 15 digits, so expensive subroutines are needed to process larger items.

Keep COMP items to less than five digits. One to four digits require two bytes; five to nine digits require four bytes; more than nine require eight bytes and more complex machine instructions.

Do not specify COMP when an unsigned field is larger than 4 digits or when a single field is larger than 8 digits.

COMP is the best format for items used as subscripts, in extensive integer arithmetic, or as OCCURS...DEPENDING objects.

Conversion to COMP will occur for subscripts unless COMP is specified. COMP results in the fastest arithmetic instructions.

Specify S in PICTURE for COMP items, otherwise an extra instruction is needed to remove the sign whenever the value is modified.

Except where COMP is best, COMP-3 is the preferred format for all numeric items. Efficient machine instructions process COMP-3 data directly; other formats often require conversion to COMP-3.

The SYNC clause is encouraged for computational items and must appear only at the elementary level. Performance is improved when performing arithmetic operations.

It is not necessary to specify SYNC on the IBM mainframe, but SYNC items are processed more efficiently. Subscripts should be defined in the PICTURE with "S" and "COMP SYNC".

3.3.2.9 RENAMES Clause

Do not use the renames clause. The REDEFINES clause can give the same result.

3.3.2.10 REDEFINES Clause

All redefining should refer to the originally defined statement regardless of the number of redefinitions.

All redefines entries must be of equal size and must have the same level numbers.

See the IBM COBOL reference manual's chapter on "Data Descriptions" when elements involved in the REDEFINE have the SYNC option.

3.3.2.11 ERROR-SWITCH

Use the following procedure to handle validation or exception error test conditions in the application program.

Include an error switch variable named ERROR-SW that is one byte, and a constant variable named PROG-ID that is eight bytes.

ERROR-SW will be initially set to zero. PROG-ID will contain the value of the program's Program-ID code.

3.3.3 PROCEDURE: HANDLING EXCEPTION CONDITION

3.3.3.1 FOR EACH EXCEPTION CONDITION

a. Assign a number and constant variable name in WORKING-STORAGE to each possible error message for easy reference.

b. DISPLAY an appropriate error message to the Control Report.

c. DISPLAY the transaction FIELD, RECORD, or other useful information regarding the exception. (All error messages should be numbered and labeled for easy reference.)

3.3.3.2 FOR PROGRAM EXECUTION STATUS

a. If program ran okay and the application job stream can continue processing, set the ERROR-SW to "0".

b. If the program has an error in logic and the next program should not continue processing, set the ERROR-SW to "1".

c. If an audit trail accumulator is out of balance and the next program must not run, set the ERROR-SW to "2".

d. If the program should abnormally terminate and the next program must not run, set the ERROR-SW to "A".

3.3.3.3 For the end of job routine

Use the ERROR-SW and set a program user RETURN-CODE and display the PROD-ID and an appropriate message to the control report.

RETURN-CODE = 000	IF ERROR-SW = ZEROS.
RETURN-CODE = 111	IF ERROR-SW = ‘1’.
RETURN-CODE = 222	IF ERROR-SW = ‘2’.
RETURN-CODE = 888	IF ERROR-SW = ‘A’.

For each exception condition, print an appropriate error message similar to the messages in the Control Report found in the Panvalet test library COBOL structured template program member: XASA1A1HA1.

3.3.4 LINKAGE SECTION

The LINKAGE SECTION is used to describe data made available from another program.

Data item description entries in the LINKAGE SECTION provide names and description, but storage within the program is not reserved since the data area exists elsewhere.

CICS/VS interface and storage control statements are found in the LINKAGE SECTION.

Example:

```
LINKAGE SECTION.

01    DFHCOMMAREA          PIC X(010).
01    DB-PCB1              PIC X(100).
01    DB-PCB2              PIC X(100).
```

NOTE: Heavily used parameter values passed via Linkage Section should be moved to Working-Storage variables.

3.4 PROCEDURE DIVISION

Programs should be developed in logical functional modules by establishing module types such as initialization modules, modules for data manipulation, processing, I/O modules, etc.

The exact structure will be a matter of judgment and the only rigid standard will be inclusion of a main line module which is a series of perform statements that can be readily analyzed to determine the logical sequence of events within the program. Primary purpose of the Main Line is to link all the processing modules together.

3.4.1 Structured Organization

Once the main-line module structure has been determined, it should be coded at the beginning of the Procedure Division. Other modules should follow the main-line module in any manner that seems appropriate to the programmer.

It is recommended that a brief description of the logic be added in front of each module structure for better program documentation.

3.4.1.1 Structured Paragraph Names

The most important consideration to be given the processing modules (paragraphs) is that they should be self-indexing.

A consecutive numbering scheme is extremely efficient for this purpose. A unique number, incremented either by 10 or 100, for each paragraph-name should be designed so that the paragraphs and their functions, when read incrementally from top-to-bottom, would read almost like a pseudo-novel.

The paragraph names for the modules will be numbered according to their primary function following these ranges (either three or four digit length):

000-090	Exception Handling Declarative
100-190	Housekeeping and Initialization
200-290	Main-line to Perform Processing Logic
300-390	Error or Exception Processes
400-490	END-OF-JOB Logic
500-890	Processing Logic Function Details
900-990	Input and Output Operations

Paragraph names should be explicit and be assigned only when necessary for PERFORM or GO TO coding. For readability, either double space between paragraphs or sections; or else use the "EJECT" command so each processing paragraph-name could be at the top-of-page.

Remember, a paragraph name can consist of up to 30 characters. The use of meaningful and self-documenting names is always helpful for program maintenance. A sequential prefix will be given to each paragraph name and assigned in an ascending order with a minimum increment of 10.

The general format for paragraph names is:

```
999-VERB-ADJECTIVE-OBJECT
```

Example:

```
100-OPEN-FILES.
        OPEN
                INPUT
                OUTPUT
130-GET-TAX-TABLES.
                .....
170-GET-MAST-DATA.
        PERFORM 900-READ-MAST.
                .....
200-UPDT-MNTH-WAGES.
                .....
```

```
230-CALC-TAX.
                .....
900-READ-MSTR.
        READ .... AT END ....
                ....
950-WRITE-MSTR.
        WRITE record-name FROM MNTH-TRANS
        ....
```

In addition to data manipulation and processing modules, the READ and WRITE processing may also include PERFORM modules.

The program design should follow Structured Programming techniques.

Structured programming involves a systematic "TOP-TO-DOWN" approach for the design and coding of a program.

A program should follow a "TOP-TO-DOWN" design construction.

That is, the flow of control within a program (or paragraph) should be from top to bottom on a page. Refer to Appendix-A for a pseudo-code example of a developed structured program design.

Here we are attempting to reduce the indiscriminate unconditional jumping from one part of a program to another. Readability is the key here.

A structured program is subdivided into functional "modules" with each module consisting of one (1) paragraph.

A module is designed to perform one specific task. This allows future changes to the program to be localized to a few modules and possibly reduce modification time.

Overlapping functions between modules should be kept of the barest minimum.

Within a program, the functional modules occupy a certain position in a hierarchy such that high level modules control the activities of subordinate modules. Certain techniques are utilized to control the flow of activity within the program.

A hierarchical relational system structure chart built from a data flow diagram, which identified the necessary processing functions and steps, should be used to visualize how the separate modules (paragraphs) of the system will logically relate, interact, and/or fit together.

Each box on the structure chart should be associated with only one paragraph-name in the program; and then expanded to look like the pseudo-code structured program logic demonstrated in Appendix-A.

Each module must have only one point of entrance and one point of exit. This reduces the number of alternative paths in a program.

3.4.1.2 Structured Coding Techniques

To minimize the effects of computer memory paging, infrequently executed paragraphs such as: initializing data-elements, end-of-data wrap-up, error detectors, and/or exception routines should be grouped together and separated from frequently executed paragraphs.

Consistent indentation and high visibility of keywords must be used to identify logical sequences of instructions.

Do not use comma (,) or semi-colon (;).

The period (.) is used when it is needed to end the conditional, sentence, paragraph name, and section name.

a. Data areas should be initialized in a separate resetable area and processed just before they are needed to minimize the possibility of a S0C7, severe system error.

b. References to the DATE, DAY, or TIME register data should be obtained *once only* at the first initialization routine and their values placed in Working-Storage variables.

c. Code only one statement per line.

Example:

Instead of ambiguous coding, such as:

```
MOVE SPACES TO  FIELD-A  FIELD-B.
```

For improved readability, code the above as:

```
MOVE SPACES TO      FIELD-A
                    FIELD-B.
```

Or better yet, code the preceding as:

```
MOVE SPACES TO FIELD A.
MOVE SPACES TO FIELD B.
```

d. Indent ELSE and always align it with its associated IF.

   The reserved word ELSE must be the only text on that line. The processing statements associated with the ELSE should be indented four spaces under it.

e. Indent AT END at least four spaces past the READ.

Example:

```
READ THE-NEXT-RECORD INTO WORKING-FIELDS
     AT END-OF-FILE PERFORM 400-END-FILE-SUMMARY.
```

f. Indent UNTIL and VARYING eight spaces past the PERFORM.

Example:

```
PERFORM   700-VERIFY-DATA
      THRU 700-VERIFY-DATA-EXIT
           UNTIL BAD-DATA.
```

g. Indent the GIVING to line up past the verb ADD or SUBTRACT.

Example:

```
SUBTRACT ALLOTTED FROM APPROPRIATED
         GIVING ENCUMBRANCE.
```

h. For programs that require data-type summations, sum all control

level break accumulators at one place in the source program.

i. Avoid using APPLY WRITE-ONLY in the output file description. When required for coding logic, specify the data-name of another level 01 item as the DEPENDING object . Do not use the corresponding field defined in the data-record itself.

j. Be sure to specify COMP and SYNC for the DEPENDING object or else a complex conversion to COMP is needed every time the record and group lengths and positions must be calculated for each record.

k. Do not move groups or records containing variable- length data, except once at the READ and/or WRITE time.

l. Start all batch programs with a DISPLAY start-program message.

m. The STOP RUN or GO BACK must be coded only once in the mainline processing paragraph.

3.4.2 CALL Statement

CALL statements should be coded in paragraphs that are to be "performed" when needed.

The CALL subroutine names should be consistent with the standard naming conventions (do not use the ALIAS feature).

Only the main entry point name will be CALLed.

The parameter names should be aligned on a column.

Example:

```
CALL "EXTRACTOR"
        USING MAST-IN-DATA
              SELECT-CRITERIA
              ACCEPTED-DATA.
```

Use the DYNAM link-edit option when calling a subprogram.

3.4.3 Comment Statements

The paragraph names will reflect the sole function of the statements included in the paragraphs and as such make most programs self documenting in its purpose and scope.

Comment statements will be used to point out or emphasis relationships and objectives of processes, comparisons, variables, complex formulas, derivations of formulas, or uses of tables or arrays.
Each subroutine will contain comments summarizing the source and function of the subroutine when its purpose is not immediately obvious. Comments should be made by using the asterisk (*) in column 7.

Example:

```
 500-GET-VALID-DATA.
*       TRANSACTION'S DATE FIELDS ARE VERIFIED AND
*       THEIR RANGES ARE COMPARED.  KEY FIELDS OF
CUSTOMER
*       ID AND ORDER NUMBERS ARE CHECKED AGAINST
TABLES.
```

Use an asterisk in column 7 to denote comment lines.
Do not use the COBOL NOTE statement.

The use of meaningful comment statements in program is strongly recommended. Briefly describe the task or function of the module to expand on the wording used for the paragraph name.

3.4.4 IF-THEN-ELSE Condition Statements

Indent the COBOL statements to identify logical sequences of instructions. This is especially applicable when defining a sequence of statements to be executed as a result of an "IF" or "ELSE" statement.

a. Simple conditional tests will be used. Coding compound statements will be kept to a minimum.

Example of poor coding:

```
IF FIELD IS EQUAL TO ‘A’ or = ‘B’ OR = ‘C’ .....
```

Must be coded as:

```
IF      FIELD = 'A'   or
        FIELD = 'B'   or
FIELD = 'C'  ....
```

b. Each compare condition should be explicitly stated. Use parenthesis to assure logical groupings of operands.

Example:

```
IF      (A IS NOT GREATER THAN B) OR
(C IS EQUAL TO D),....
```

c. ***Do not*** use compound negative conditional tests.

Example:

```
IF      A IS NOT EQUAL TO NOT C
```

Example:

```
IF      NOT B IS NOT LESS THAN NON-D
```

Example:

```
IF NOT A IS NOT = B OR NOT C = D
```

3.4.5 COMPUTE Statement

The use of the COMPUTE statement is encouraged. It is easy to check out the formula and it is more self-documenting.

Items referenced by arithmetic verbs should all be COMP or COMP-3. Items added or subtracted should have the same number of decimal places.

For items not in ideal formats of same attributes and same decimal lengths, move them to ideal items in Working-Storage before processing.

For computations involving several arithmetic operations, the COMPUTE verb is more efficient than a sequence of separate arithmetic verbs.

However, the precision of the intermediate results cannot be controlled using COMPUTE, and may not generate the expected final report.

For computation operations, fixed-point arithmetic is calculated much faster than floating-point arithmetic, however, floating-point arithmetic is much more precise and accurate than fixed-point.

The relative speeds of the arithmetic operations are as follows:

ADD or +	fast
SUBTRACT or -	fast
MULTIPLY or *	slow
DIVIDE or /	slower
** (exponentiation)	very slow

Avoid costly errors from multiplication by 0 or 1, or division into 0 by careful arrangement of the logic, or by coding an extra IF to bypass the calculation.

Eliminate unnecessary use of the ROUNDED clause.

Consider rounding directly in a calculation for values which always have the same sign, but be sure the rounding formula is documented to eliminate possible misunderstanding or misinterpretation in later maintenance of the program.

Example for two decimal place accuracy after transaction:

```
COMPUTE PENALTY-PERCENT = RATE * TIME + 0.005.
```

The COMPUTE verb should always be used when multiple arithmetic operators are involved. Consider two sets of equivalent code:

Poor Code:

```
MULTIPLY    B BY B GIVING B-SQUARED.
MULTIPLY    4 BY A GIVING FOUR-A.
MULTIPLY    FOUR-A BY C GIVING FOUR-A-C.
SUBTRACT    FOUR-A-C FROM B-SQUARED GIVING RESULT-1.

COMPUTE     RESULT-2 = RESULT-1 ** 0.5.
```

```
SUBTRACT  B FROM RESULT-2 GIVING NUMERATOR.
MULTIPLY  2 BY A GIVING DENOMINATOR.
DIVIDE    NUMERATOR BY DENOMINATOR GIVING X.
```

Improved Code:

```
COMPUTE   X = (-B + ((B*B) - (4 * A * C)) ** 0.5) / (2 * A).
```

Both (of the above) sets of code apply to the quadratic formula,

$$X = \frac{-B + \text{square-root}(BB - 4AC)}{2A}.$$

It is easier to determine what is happening from the single COMPUTE statement. It is difficult to realize the cumulative effect of the 8 individual arithmetic statements. Interpretation of the unacceptable code is further clouded by the mandatory definition of data names for intermediate results, e.g., RESULT-1, RESULT-2, SQUARED, X, NUMERATOR, etc.

Parentheses are often required in COMPUTE statements to alter the normal hierarchy of arithmetic operations.

For example, the parentheses are required around “2 * A” in the denominator. If they had been omitted, the numerator would have first been divided by “2” and then the quotient would have been multiplied by “A”.

Sometimes parentheses are optional to the compiler, but should be used to clarify things for the programmer. The parentheses around “4*A*C” do not alter the normal order of operations and hence are optional.

3.4.6 Condition Names

Meaningful condition names in the DATA DIVISION make for excellent documentation and can be a great aid in revealing the program logic.

3.4.7 Conditional Tests

To minimize misunderstanding, the relational operators " > " and " < " should be coded 'GREATER THAN' and 'LESS THAN'. It makes the program listing more self documenting and more narrative and prose-like.

Do not use implied conditional subjects or operators.

Example:

```
IF      X=FIVE OR SIX OR Y OR TWO
```

3.4.8 DISPLAY Statement

The DISPLAY UPON CONSOLE is permitted only to designate the start of a batch program, or the result of the execution of a batch program.

Any other use requires prior approval from the ICSD-PSB Production Services Branch Chief. An important exception to this rule is a situation requiring operator action, such as, multiple uses of tapes for Read Only then Read/Write.

If the DISPLAY statement is used, a comment line should be noted in the FILE-CONTROL paragraph stating its use in the program.

DISPLAY messages should be identified by the PROGRAM-ID which issues it.

Example:

```
DISPLAY     'START PROGRAM –ID:   XLSA1A1S'.
DISPLAY     'ABNORMAL END OF PROGRAM:   XLSA1A1L'.
DISPLAY     'NORMAL END OF PROGRAM - XLSA1A1L'.
```

3.4.9 GO TO Statement

Do not use the DEPENDING ON format. Use separate nested "IF" statements instead.

Use of GO TO will be kept to a bare minimum. The permissible GO TO transfer control to an "EXIT" paragraph-name.

The "GO TO" statement when employed should direct control of the program to a point subordinate to it within the same paragraph.

Upward processing control movement is not allowed. The "GO TO" statement should not be used to transfer control to a point outside of the module in which it resides.

Example:

```
200-SAMPLE-GOTO.
        READ TRAN-REC INTO WS-TRAN-REC
        AT END      MOVE HIGH-VALUES TO PREV-ID.
        IF      PREV-ID = HIGH-VALUES
                GO TO 200-SAMPLE-GOTO-EXIT.
        MOVE NEW-ID   TO PREV-ID.
200-SAMPLE-GOTO-EXIT.
        EXIT.
```

When it is necessary to loop back to the beginning of a module, instead of using the “GO TO” statement, use “PERFORM paragragh-name” THRU “paragraph-name-exit”:

```
PERFORM         520-CHECK-OCCURS-LIMIT
        THRU        520-CHECK-OCCURS-LIMIT-EXIT
        VARYING ..... UNTIL.....
PERFORM   ... UNTIL
PERFORM   ... TIMES
```

By doing so, the condition(s) under which the looping is undertaken is well defined.

3.4.10 IF Statement

Nested IF statements are permitted. Nesting the "THEN-ELSE" condition should be limited to no more than 3 levels. When nesting is used, the proper consistent alignment of paired IF-THEN-ELSE statements is required for ease of maintenance. Consistent indentation shows the subordination relationships.

Example:

```
IF      condition-1
        IF      condition-2
                IF      condition-3
                        statement-3-1
                ELSE
                        statement-3-2
        ELSE
```

```
            IF condition-4
                    statement-4-1
            ELSE
                    statement-4-2
    ELSE
            IF condition-5
                    IF condition-6
                            statement-6-1
                    ELSE
                            statement-6-2
            ELSE
                    statement-5-2.
```

The ONLY allowable exception to these relational indentations is for more than three (3) mutually exclusive case selection criteria.

Example:

```
    IF      GET-A-RECORD
            PERFORM   210-GET-A-RECORD
                 THRU   210-GET-A-RECORD-EXIT
    ELSE
    IF      START-BROWSE
            PERFORM   220-START-BROWSE
                 THRU   220-START-BROWSE-EXIT
    ELSE
    IF      GET-NEXT-RECORD
            PERFORM   230-GET-NEXT-RECORD
                 THRU   230-GET-NEXT-RECORD-EXIT
    ELSE
    IF      GET-PREVIOUS-RECORD
            PERFORM   240-GET-PREVIOUS-RECORD
                 THRU   240-GET-PREVIOUS-RECORD-EXIT
    ELSE
            PERFORM   300-WRONG-GET-REQUEST
                 THRU   300-WRONG-GET-REQUEST-EXIT.
```

In a sequence of mutually exclusive IF-ELSE statements, try to order the conditional statements from most likely test-condition, down to the least likely, but do not sacrifice program readability or understandability.

Use the IF-ELSE conditional statement to isolate groups of code that do not require processing for every execution path.

Do not place any Input/Output statement within the conditional IF-ELSE conditional statements.

The THEN clause is optional. When coded, it must be on a line by itself.

Example:

```
IF      condition-1
THEN
        IF      condition-2
        THEN
                IF      condition-3
                THEN
                        statement-3-1
                ELSE
                        statement-3-2
        ELSE
                IF      condition-4
                THEN
                        statement-4-1
                ELSE
                        statement-4-2
ELSE
        IF      condition-5
        THEN
                statement-5-1
        ELSE
                statement-5-2.
```

Use the negative conditional IF only when the negative statement is more straight-forward, explicit, much clearer or eliminates the use of "THEN NEXT SENTENCE".

Example:

```
IF      VARIABLE-NAME NOT NUMERIC
        PERFORM   500-NOT-NUMERIC
             THRU   500-NOT-NUMERIC-EXIT.
```

3.4.11 LOGICAL Comparisons

For numeric comparisons, the items compared should both be COMP or COMP-3 be both signed or both unsigned, and have similar PICTUREs.

Numeric data must be either COMP or COMP-3 for the machine compare instructions.

Comparing numeric items necessitates very costly conversion and decimal point alignment steps, unless the compared items have identical formats.

COMP-3 is easier to trace in a dump, but COMP is slightly faster than COMP-3.

The tests “IF NUMERIC” and “IF ALPHABETIC” class conditional statements may be quite essential but should be used with care. They are very costly, and use should generally be limited to validating raw input data.

3.4.12 MOVE Statement

Do not use the MOVE CORRESPONDING. It does not document well for readability, and can be difficult to maintain if exceptions occur at some later date.

When using "MOVE ... TO ..." for a group of "Moves", the word "TO" should be aligned on the same column to improve readability.

Example:

```
MOVE PAY-SSN          TO   PRT-SSN.
MOVE PAY-NAME         TO   PRT-NAME.
MOVE PAY-ADDRESS      TO   PRT-ADDRESS.
```

a. When using an alphanumeric data-name with literal values (or with WORKING- STORAGE variable values enclosed in single quotes), before the required MOVE is processed, verify the literal will be moved to a target item that has the same number of characters. If the moved literal is shorter than the receiving area, the statement actually results in an efficient move of the literal followed by an inefficient move of spaces to clear the remaining character positions.

b. Literals should be filled out (padded) unless that would entail a large number of trailing spaces (say 12 or more). Literal moves are most efficient when the sending item is at least as large as the receiving item.

c. It is more efficient to define constant data-element item of a series of spaces or zeros in Working-Storage and MOVE from those variable-names, rather than by using the figurative constant SPACES or ZEROS in the MOVE statements.

d. The recommended technique to save CPU time and conserve memory is to define a figurative constants with a maximum size specification and let the COBOL compiler truncate it during the data MOVE.

Example:

```
01      INIT-CONSTANTS.
        05      ALL-SPACES              PIC X(80)   VALUE
SPACES.
        05      ALL-HIGH-VALUE          PIC X(100) VALUE
HIGH-VALUE.
        05      ALL-LOW-VALUE           PIC X(100) VALUE
LOW-VALUE.
```

Consider group-level moves rather than separate moves of elementary items when PICTURE definitions of sending and receiving items correspond.

The numeric move is always most efficient if both items are COMP or COMP-3, have the same number of decimal places (if any), and both (or neither) have S in PICTURE. However, if items are initially not aligned, a planned move to ideally formatted items in Working-Storage will save repeated conversions.

Moves to an item which is the object of an OCCURS ... DEPENDING clause should be avoided at all times.

3.4.13 ON Condition

Eliminate unnecessary use of ON SIZE ERROR clause. Eliminate by using such techniques as checking for zero denominator before dividing,

etc.

3.4.14 OPEN/CLOSE Statements

Use a single OPEN statement rather than separate OPEN statements for files opened at the same time. Use of a single OPEN statement reduces routine loading time. System routines required for OPEN are referenced once for each OPEN statement regardless of the number of files specified in the statement.

Sometimes a programmer uses a file over and over to hold temporary data, repeating OPEN-WRITE-CLOSE and OPEN-READ-CLOSE sequences many times. These applications should be studied carefully to see whether a Working-Storage table could be used instead.

OPEN and CLOSE are very costly statements and are designed to be used only once or a couple of times by each program.

The OPEN/CLOSE statement should be executed only once per program.

The OPEN/CLOSE statement *cannot be used* in CICS programs.

3.4.15 PERFORM Statement

To eliminate the possibility of processing logical "fall-thru", use the format:
PERFORM procedure-name-1 THRU procedure-name-2.
The procedure-name-2 will be an EXIT for the procedure-name-1 module.

Example:

```
PERFORM   510-GET-VALID-TRANS
    THRU  510-GET-VALID-TRANS-EXIT.
```

Perform paragraph-names only, *do not* perform section-names.

The PERFORM UNTIL option is encouraged. Its use is usually obvious and is easy to check out. If possible, the program logic must check the parameters affecting the routine to assure it will not remain in a loop.

The PERFORM VARYING format is permissible but the other forms are preferable.

When using the VARYING format, try to place numeric computations outside of the loop. When using the varied variable's value for computations that depend on the number of times the paragraph is performed, do the computations after the loop processing.

The VARYING or UNTIL clause should be indented on a line separate from the PERFORM verb.

Example:

```
PERFORM   510-GET-VALID-TXN
    THRU  510-GET-VALID-TXN-EXIT
    UNTIL TXN-LAST-RECORD.
```

When possible, use multiple separate PERFORM statements instead of NESTED PERFORM statements.

3.4.16 Program Switches

Program switches should be kept at a minimum. However program initialization switches and end-of-job switches are permitted since their use and need is usually apparent. Switches that are set only once per program run are also obvious and should be set via a control card. Such as selecting only certain Counties to be processed; or selecting only certain reports to be generated. In any event, detailed annotation of switches is required.

The recommended switch practice is to set a switch as a result of a logical controlling condition. A preferable method is to test the condition again. If the condition is no longer present, anticipate the situation and route the program through a different series of modules.

In place of a separate switch to identify an EOF condition, the non numeric literal KEY field of the record may be set to high-value. The KEY field can then be tested for the EOF condition. The high-value in the KEY field simplifies the overall logic to complete processing of other input files. But a separate switch offers more flexibility.

If switches are used, use the level-88 feature and variable names that explain the condition.

Example:

```
05      EYES-COLOR              PIC X(02)    VALUE SPACES.
        88      EYES-ARE-BLUE                VALUE "BL".
        88      EYES-ARE-BROWN               VALUE "BR".
        88      EYES-ARE-GREEN               VALUE "GR".
                .
                .
                .
IF      EYES-ARE-BLUE
        PERFORM   500-BLUE-EYES-RTN
             THRU   500-BLUE-EYES-RTN-EXIT
ELSE
IF      EYES-ARE-BROWN
        PERFORM   510-BROWN-EYES-RTN
              THRU   510-BROWN-EYES-RTN-EXIT
ELSE
IF      EYES-ARE-GREEN
        PERFORM   520-GREEN-EYES-RTN
             THRU   520-GREEN-EYES-RTN-EXIT.
```

3.4.17 Print Report Format

For traditional reporting every time you start to edit the detail data-fields in the line, move all-spaces to a redefined variable that has a length to cover the entire print line area.

Either define a variable-name defined as all-spaces, or specify VALUE SPACES in another record layout definition, and MOVE this WORKING-STORAGE variable to the entire print line area.

Consider moving spaces to each specific field position that has been recently modified, if blanking the whole line area via a group level name is not desired.

3.4.18 Program Audit Control

Each program must have an audit control report. This report must include the following when applicable:

a. Program identification name as found in IDENTIFICATION DIVISION.

b. Report headers that identify the State, Department, Division, Branch,

and when applicable, the Section of the entity to receive the report.

c. Centered Title containing a description like: "PROGRAM CONTROL REPORT" or “STATISTICAL/AUDIT REPORT”.

d. The operating system run date.

e. Any input control statements or parameter data.

f. Any message for any encountered error or exception condition.

g. Input-Reject-Output record counts for files.

h. Any calculated Batch control total.

3.4.19 Program Constants

Data names should be used to express constant values instead of literals. This promotes easier program maintenance and also provides for better readability and understanding.

Example:

```
05    VALUE-9    PIC  9(01)    VALUE ‘9’.
05    VALUE-X    PIC  X(01)    VALUE ‘X’.
05    ONE        PIC  S9(05)   VALUE +1.
05    TWO        PIC  S9(05)   VALUE +2.
```

3.4.20 Prohibited or Restricted Verbs

a. The ALTER statement is not permitted.

b. The ACCEPT FROM CONSOLE option is not permitted.

To use this option requires prior written approval from the ICSD PSB Chief. If used, it should be noted in the FILE-CONTROL paragraph in a comment.

c. The EXAMINE verb is not permitted.

d. For any CICS/VS program, reserved words ACCEPT, DATE, DAY,

CURRENT-DATE, DISPLAY, EXHIBIT, INSPECT, SIGN IS SEPARATE, STOP RUN, TIME, and UNSTRING are not allowed.

3.4.21 READ Verb

Data from records are to be read into Working-Storage defined area, and not to the FD record description field.

Use the READ...INTO...WORKING-STORAGE-INPUT-AREA format.

This form provides a readable trace in memory dumps.

Use the "AT END" or "INVALID KEY" clauses for any "READ" statement.

Code only one READ statement for each file in a paragraph to be PERFORMED when data is needed.

The advantages of one functional READ or WRITE paragraph for each file are:

a. Coding can be added to count the processed records.

b. The file records can be reformatted without changing the program's logic.

c. DEBUG statements can be easily added to DISPLAY records.

The file input area should be filled with all high values at end-of-file unless the end-of-file condition forces an end-of-job condition. This particular approach is in the interest of uniformity for EOF logic rather than for gaining any program performance or efficiency advantage.

Whenever possible, avoid having more that one record per file in core at any one time. For example, when master record (with KEY=121) is in core, do not read the next record (with KEY=122) into core until all processing on record (with KEY=121) has been completely processed.

3.4.22 Record Counts

Record Counts should be maintained for all input and output files. Definition of counter names should be meaningful and representative of the file's function or purpose. Counters serve as a good debugging tool

and should be printed or displayed as part of the end of job routine whether the run is successful or not.

3.4.23 Report Writer Feature

Do not use any COBOL Report Writer Facility. The COBOL FOR MVS and VS COBOL II compilers do not directly support REPORT WRITER FACILITIES.

3.4.24 Sequence Check

Sequence checking should be performed whenever specific sequence is required of an input file. Any sequence error message generated should include previous and current keys as well as current record count.
FACILITIES.

3.4.25 Sort Feature

The internal COBOL SORT verb should be avoided for these important reasons:

- The COBOL program becomes a subroutine to the SORT verb.
- If program abends, the resulting dump is almost useless.
- Programs using CICS cannot use SORT.

The CASORT SRAM, external utility is the recommended method for sorting.

The SORT feature, if it must be used, is for very short and very small files only. The record released to the SORT should be an exact image of the input record with a sort key tagged on to the left-most position.

SORT-CORE-SIZE will default to a value specified at the time the SORT program was installed. This MAXSIZE should always be used. Give SORT as much information as possible.

All file SORTS will be accomplished through the external procedures using the computer operating system utility SORT program.

Example:

```
PROC=SORT
```

PROC=SORTD
PROC=REGSORT
PROC=BIGSORT

Use simple key fields. Combine adjacent key fields to build super description keys.

The COBOL Optimizer will cause unpredictable results for the "USING/GIVING" clauses with the internal SORT feature.

3.4.26 STRING/UNSTRING Command

The STRING/UNSTRING commands should be used with care. A character-by-character MOVE loop is more efficient.

It is more efficient to REDEFINE the data as a table of one-character items and to unstring the data with IF and MOVE statements.

3.4.27 Subscripting and Indexing

The Indexed form is preferable over subscripting, but both forms are acceptable. The Indexed form is more efficient and produces better documentation.

Indexing is usually more efficient than subscripting for sequential table searching or when the same subscript is used several times in relation to the number of times its value is changed.

Always specify COMP SYNC and S in PICTURE for any item used as a subscript.

Index variables are not interchangeable between tables.

If a table item is processed extensively (that is, many references to the same item in the same logic path or loop) move the table item to a fixed data-element in Working-Storage, then direct all the references to the fixed item.

The command, SET to a literal value, is fast. However, SET to a value of a data name may require format conversion and may slow the program's performance.

Subscript with data names, do not use literals as subscripts.

3.4.28 Tables

A fixed length table or fixed length group items must never exceed 131,071 bytes in length.

The maximum storage size of varying tables is 32,767 bytes.

When coding Table Entries, separate line entries should be made for each item in the table. Separate entries will aid in the correction of entries and provide for better readability. xx

Example, Bad Code:

```
05      MONTH-LITERAL-TABLE.
        10      MONTH-TEST      PIC X(18)
                                VALUE 'JANUARY   FEBRUARY'.
```

Example, Good Code:

```
05      MONTH-LITERAL-TABLE.
        10      FILLER          PIC X(09)  VALUE 'JANUARY'.
        10      FILLER          PIC X(09)  VALUE 'FEBRUARY'.
```

When working with TABLES, always use TABLE-MAXIMUM checks to keep from exceeding the boundaries of the TABLE.

For efficiency, put frequently accessed items at the beginning of the TABLE for sequential searches.

Do not alter table entries in their table location. Move the entries to working storage areas and modify those area and then move the values back to the table locations.

Any table whose size or values may change during a program run must be designed to be loaded each time the program is executed.

3.4.29 Termination Processing

After all input and output records have been appropriately been read and acted upon, the program will have end-of-execution processes that summarize the number of records that were read, accepted, rejected,

written to electronic media, or displayed on a printed listing.

3.4.30 Normal Processing

Normal expected termination processing for the control report from batch programs should include the following:

a. Audit trail summarizing any cumulated fields.

b. Total count of each input and output data set.

c. Message sent to the operator's console to identify the program name and its normal ending status.

3.4.31 Abnormal Processing

Abnormal termination processing for control reports should have the following:

a. A message to explain the cause for the termination;

b. The record key and fields causing the termination;

c. The input/output counts for the data fields;

d. A return condition code as specified for the standard ERROR-SW;

e. An abnormal end of job message to the operator's console and to the printed control report.

3.4.32 Trace Verb

Do not use the TRACE verb for any CICS program.

The RESET TRACE Verb consumes time even if TRACE is never activated. Therefore, *it is important to remove* these statements when they are no longer needed for debugging.

3.4.33 WRITE or REWRITE Statement

The "WRITE ..... FROM" format will be used.

Example:

```
WRITE PRINT-DETAIL-RECD FROM WORKING-OUTPUT-AREA.
```

The "AFTER ADVANCING" format is to be used instead of "AFTER POSITIONING".

WRITE statements will be kept to a minimum. The most CPU time is consumed to process COBOL I/O verbs. Put WRITEs in paragraphs to be PERFORMED.

Line counting is required for overflow printed report testing.

The REWRITE verb is a very time-consuming verb. Often several successive transactions apply to the same record. Be sure your program waits to REWRITE a record until after no further use can be made of the record, and all changes have been made to it.

4 COBOL ENVIRONMENTS

COBOL environments are related to conventions and techniques for “batch” and “real-time” programs. Batch programs will use VSAM (virtual sequence access methods) to process permanently stored data on direct access storage devices. Real-time programs will use CICS (command information controlled sequence) in systems areas defined and controlled by the ICSD Systems Services Branch (SSB).

4.1 VSAM PROCESSING IN COBOL/VS

The RECORD KEY must be coded for keyed sequence data sets. If ALTERNATE RECORD KEY is used, there must be a job control JCL DD Statement defined for the path associated with the alternate index.

The FILE STATUS must be checked after each OPEN/CLOSE or READ/WRITE command to assure successful completion.

For the SELECT...ASSIGN Statement, the "ASSIGN TO" will not have the "UT-S" code.

Before each direct-access READ, check the last record read. Data clustering is quite common, and you may very often already have the record needed.

4.2 CICS/VS PROCESSING IN COBOL/VS

The format for CICS Statements will have only one activity on one line. Any parameters will be indented five spaces under the "EXEC CICS" command. The "END-EXEC" will be aligned with the corresponding "EXEC CICS".

Example:

```
EXEC CICS  READ NEXT
        DATASET    ('NAMEIDX')
        INTO       (STUDENT-RECORD)
        RIDFLD     (STUDENT-NAME)
        LENGTH     (REC-LENGTH)
END-EXEC.
```

4.3 CICS Programming Techniques and Restrictions

a. OBJECT PROGRAM SIZE must not exceed 256K.

b. WORKING-STORAGE plus Task Global Table (TGT) must not exceed 64K.

c. Always code "GOBACK" as the last statement in a program to act as a "CICS RETURN". The default COBOL's "STOP RUN" will terminate CICS.

d. Do not code FILE SECTION (FD) nor any SELECTs.

e. Do not use any Input/Output verbs such as:

OPEN/CLOSE	REPORT WRITER
READ/WRITE	SEGMENTATION
SORT	CURRENT-DATE
ACCEPT/DISPLAY	DAY/DATE/TIME
EXHIBIT/TRACE	STOP RUN
INSPECT/UNSTRING	

f. Do not use the following COBOL compiler options:

COUNT	DYNAM	SYST
FLOW	SYMDMP	TEST
STATE	ENDJOB	STXIT

g. Move LOW-VALUES to the map area in WORKING- STORAGE before

moving data to the map area.

4.4 OPERATING SYSTEM PROCESSING PROCEDURES

The COBOL programmer communicates with the operating system, job scheduler, and programs with operating system (JCL) job control statements.

ICSD has established seven utility cataloged procedures to be used during the development and testing phases of an application in a COBOL or CICS environment. Five procedures are used to support COBOL syntax and logical checking; and two are used to support the CICS environment.

The default parameters selected for each step within these cataloged utility procedures were chosen to provide effective control aids for the job scheduler to regulate the execution of steps, to retrieve and determine the disposition of data allocating resources, and to communicate effectively with the operators and programmers. The following is an example of an acceptable override for the COBOL execution statement:

```
//      PARM.COBOL='APOST,NOADV,LIB,FLAG(W)'
//      PARM.LKED='LET,LIST'
```

The IBM OS/VS COBOL Compiler and Library Programmer's Guide has detailed information on the selected parameters and options in the chapter: "Using the Catalogued Procedures".

4.4.1 COBOL-FOR-MVS Procedures

There are five utility catalogued procedures to be used during the development and testing of a COBOL application program. These catalogued COBOL procedures incorporating Computer Associates CA-OPTIMIZER that will be used to debug the logic of the programs or test the system job flow controls, they are:

a. XCOBWC - Compile with Optimizer to test program statement syntax.

b. XCOBWCL - Compile with Optimizer and produce load module to be stored in ICSD.LINKLIBT.

c. XCOBWCLX - Compile with Optimizer and produce load module to be stored in ICSD.TESTLIBT.

d. XCOBWCLR - Compile with Optimizer and generate REPORT-WRITER executable load module to be stored in ICSD.LINKLIBT.

e. XCOBOCLG - Compile with Optimizer, store generated load module in a temporary data set, and execute the program.

For further information on the utility catalogued COBOL procedures incorporating the OPTIMIZER, see the COBOL/VS CA-OPTIMIZER Procedure users guide distributed by the Technical Standards and Methods Section of the EDP Division.

4.4.2 COBOL With CICS/VS Procedures

COBOL programs developed for real-time interactive application transactions that use the IBM OS/CICS/VS may be processed via the following catalogued utility procedures.

a. CICOBWC

This procedure is used to compile COBOL source stored in a partitioned data set like the programmer's or department's TSO data sets. No load module will be generated.

b. CICOBWCL

This procedure is used to compile COBOL source statements stored in a partitioned data set like the programmer's or department's TSO data sets. The load module generated must be defined in the symbolic parameter "NAME=", and this name will be stored in "CICSVS.LINKLIBT".

c. CIVCCOB

This procedure is used to compile COBOL source statements stored in the PANVALET library "EDPD.PANVTEST". This procedure will not generate an object load module. The PANVALET control statements are specified after a job control statement:

d. CIVCCOBL

This procedure is used to compile COBOL source statements stored in the PANVALET library, "EDPD.PANVTEST". This PANVALET control statements are specified after the job control statement:

The load module generated must be defined in the procedure symbolic parameter "NAME", and this name will be stored as a member of "CICSVS.LINKLIBT".

4.5 Test-To-Production Procedures

COBOL applications that have been tested to the satisfaction of the system requestor, and whose operations will be transferred from the ICSD Client Services Branch to the ICSD-PSB should have all tested object load modules transferred from the test library, "EDPD.LINKLIBT" to the production library "EDPD.LINKLIBP".

Any application to be handled by the ICSD-PSB should have load modules stored in "EDPD.LINKLIBP". For security and control, the application project manager must submit a “LOAD MODULE REQUEST” to ICSD-PSB Control Unit.

The physical transfer of the application's load module from “EDPD.LINKLIBT” occurs immediately when the transfer job executes. The actual delete of the load modules from "EDPD.LINKLIBT" is done once a day during the evening scheduled System Backups.

5 APPENDIX A: Structured Program Design

This example illustrates the 3 step transition from a data-flow pictorial diagram, to a functional hierarchical structured chart, and then to a pseudo code of the logical algorithm.

The structure and syntax for pseudo code is independent of any formal computer programming language. The resulting pseudo code serves as the basis for the COBOL program solution.

The details for the program comes from expanding the information defined in the requested system's specifications. The needed instructions that conform to the solution are expanded into the appropriate program syntax structures.

Example of Pseudo-Code:

```
REPEAT BEST-SOLUTON
        WHILE THERE-IS-MORE-RAW-DATA

BEST-SOLUTION
        PROCESS GET-GOOD-INPUT
        PROCESS  PROCEDURES-FOR-BEST-SOLUTION
        PROCESS PUT-OUT-SOLUTION
BEST-SOLUTION-ENDED

GET-GOOD-INPUT
        PROCESS GET-RAW-INPUT-DATA
        PROCESS EDIT-RAW-INPUT-DATA
        IF      BAD-RAW-INPUT
        THEN
                PROCESS SHOW-BAD-INPUT
        ELSE
                PROCESS GOOD-DATA-RECORD
        PROCESS DISPLAY-GOOD-INPUT
GET-GOOD-INPUT-ENDED

PUT-OUT-SOLUTION
        PROCESS FORMAT-SOLUTION
        PROCESS WRITE-SOLUTION-TRANSACTION
        PROCESS DISPLAY-SOLUTION
PUT-OUT-SOLUTION-ENDED
```

6 APPENDIX B: Structured COBOL Skeleton

A Copy of COBOL-FOR-MVS Model source listing follows:

```
000051  01  TABLE-STRUCTURES.                                            00810000
000052      03  FILLER                      PIC X(13)  VALUE "TABLES  START".00820000
000054*****************************************************************00830000
000055*                                                               *00840000
000056*                    CONSTANTS AND LITERALS                     *00850000
000057*                                                               *00860000
000058*****************************************************************00870000
000059  01  CONSTANTS-AND-LITERALS.                                     00880000
000060      03  FILLER                      PIC X(13)  VALUE "CONSTNT START" 00890000
000062      03  ABORT-CODE                  PIC X(01)  VALUE "A".       00900000
            03  NO-ERROR                    PIC X(01)  VALUE ZERO       00910000
000063      03  ONE                         PIC X(01)  VALUE "1"        00920000
000064      03  PROG-ID                     PIC X(04)  VALUE "A2GMUC".  00930000
000069      03  TWO                         PIC X(01)  VALUE "2".       00940000
000071                                                                   00950000
000065                                                                   00960000
000066*****************************************************************00970000
000067*                                                               *00980000
000068*                        SUBSCRIPTS                             *00990000
000069*                                                               *01000000
000070*****************************************************************01010000
000071  01  SUBSCRIPT-VARIABLES.                                        01020000
000072      03  FILLER                      PIC X(13)  VALUE "SUBSCRP START" 01030000
000073                                                                   01040000
000074                                                                   01050000
000075*****************************************************************01060000
000076*                                                               *01070000
000077*                     REPORTED MESSAGES                         *01080000
000078*                                                               *01090000
000079*****************************************************************01100000
000080  01  REPORTED-MESSAGES.                                          01110000
000081      03  FILLER                      PIC X(13)  VALUE "MESSAGE START" 01120000
000082      03  CNTL-RPT-STARTED            PIC X(132)                  01130000
000083                  VALUE "PROGNAME - PROGRAM STARTED".              01140000
000084                                                                   01150000
000085                                                                   01160000
000086*****************************************************************01170000
000087*                                                               *01180000
000088*                     WORKING VARIABLES                         *01190000
000089*                                                               *01200000
000090*****************************************************************01210000
000091  01  WORK-AREA-DATA-NAMES                                        01220000
000092      03  FILLER                      PIC X(13)  VALUE "WK-AREA START". 01230000
000093      03  CNTL-RPT-LINE               PIC X(132) VALUE SPACES.    01240000
            03  TODAYS-DATE                 PIC X(06).                  01250000
                                                                        01260000
000095                                                                   01270000
000096*****************************************************************01280000
000097*                                                               *01290000
000098*                     SWITCHES AND FLAGS                        *01300000
000099*                                                               *01310000
000100*****************************************************************01320000
000101  01  SWITCHES-AND-FLAGS.                                         01330000
000102      03  FILLER                      PIC X(13)  VALUE "SWITCH  START". 01340000
000103      03  ERROR-SW                    PIC X(01)  VALUE ZERO       01350000
            03  END-OF-PROGRAM              PIC X(01).                  01360000
                88  END-OF-PROGRAM-ON                VALUE "1".         01370000
                88  END-OF-PROGRAM-OFF               VALUE "0".         01380000
000105*****************************************************************01390000
000106*                                                               *01400000
```

```
000051  01  TABLE-STRUCTURES.                                            000010000
000052      03  FILLER                  PIC X(13)  VALUE "TABLES  START".000020000
000054 ******************************************************************000030000
000055 *                                                                *000040000
000056 *                    CONSTANTS AND LITERALS                      *000050000
000057 *                                                                *000060000
000058 ******************************************************************000070000
000059  01  CONTANTS-AND-LITERALS.                                       000080000
000060      03  FILLER                  PIC X(13)  VALUE "CONSTNT.START" 000090000
000062      03  ABORT-CODE              PIC X(01)  VALUE "A".            000100000
            03  NO-ERROR                PIC X(01)  VALUE ZERO            000110000
000063      03  ONE                     PIC X(01)  VALUE "1".            000120000
000064      03  PROG-ID                 PIC X(06)  VALUE "A2CMLC".       000130000
000065      03  TWO                     PIC X(01)  VALUE "2".            000140000
000066                                                                   000150000
000067 ******************************************************************000160000
000068 *                       SUBSCRIPTS                               *000170000
000069 *                                                                *000180000
000070 ******************************************************************000190000
000071  01  SUBSCRIPT-VARIABLES.                                         000200000
000072      03  FILLER                  PIC X(13)  VALUE "SUBSCRP START" 000210000
000073                                                                   000220000
000074                                                                   000230000
000075 ******************************************************************000240000
000076 *                                                                *000250000
000077 *                    REPORTED MESSAGES                           *000260000
000078 *                                                                *000270000
000079 ******************************************************************000280000
000080  01  REPORTED-MESSAGES.                                           000290000
000081      03  FILLER                  PIC X(13)  VALUE "MESSAGE START".000300000
000082      03  CNTL-"RP"-STARTED       PIC X(132)                       000310000
000083                 VALUE "PGMNAME - PROGRAM STARTED".                000320000
000084                                                                   000330000
000085                                                                   000340000
000086 ******************************************************************000350000
000087 *                                                                *000360000
000088 *                    WORKING VARIABLES                           *000370000
000089 *                                                                *000380000
000090 ******************************************************************000390000
000091  01  WORK-AREA-VARIABLES                                          000400000
000092      03  FILLER                  PIC X(13)  VALUE "WK-AREA START".000410000
000093      03  CNTL"RP"LINE            PIC X(132) VALUE SPACES.         000420000
            03  TODAYS DATE             PIC X(06).                       000430000
                                                                         000440000
000095                                                                   000450000
000096 ******************************************************************000460000
000097 *                                                                *000470000
000098 *                    SWITCHES AND FLAGS                          *000480000
000099 *                                                                *000490000
000100 ******************************************************************000500000
000101  01  SWITCHES-AND-FLAGS.                                          000510000
000102      03  FILLER                  PIC X(13)  VALUE "SWITCH  START".000520000
000103      03  ERROR-SW                PIC X(01)  VALUE ZERO            000530000
            03  END-OF-PROGRAM          PIC X(01).                       000540000
                88  END-OF-PROGRAM-ON               VALUE "1".          000550000
                88  END-OF-PROGRAM-OFF              VALUE "0".          000560000
000105 ******************************************************************000570000
000106 *                                                                *000580000
```

COBOL VS Standards and Conventions

```
00107 *                        COUNTERS AND CUMULATORS                  *01210000
00108 *                                                                 *01220000
00109 *-----------------------------------------------------------------*01230000
00110  01  COUNTERS-CUMULATORS.                                         01240000
00111      05  FILLER                  PIC X(13)  VALUE "COUNTER START" 01250000
00112      05  CTR-FILE1               PIC S9(05) COMP-3 VALUE ZERO.    01260000
00113      05  CTR-FILE2               PIC S9(05) COMP-3 VALUE ZERO.    01270000
00114      05  CTR-FILE3               PIC S9(05) COMP-3 VALUE ZERO.    01280000
00115      05  CTR-FILE4               PIC S9(05) COMP-3 VALUE ZERO.    01290000
00116      05  CTR-FILE5               PIC S9(05) COMP-3 VALUE ZERO.    01300000
00117      05  CTR-FILE6               PIC S9(05) COMP-3 VALUE ZERO.    01310000
00118      05  CTR-FILE7               PIC S9(05) COMP-3 VALUE ZERO     01320000
00119      05  CTR-FILE8               PIC S9(05) COMP-3 VALUE ZERO.    01330000
00120      05  CTR-FILE9               PIC S9(05) COMP-3 VALUE ZERO.    01340000
00121      05  CTR-FILE10              PIC S9(05) COMP-3 VALUE ZERO.    01350000
00122                                                                   01360000
00123                                                                   01370000
00124  01  FILLER                      PIC X(20)                        01380000
00125                          VALUE "END WORKING STORAGE   PGMNAME".   01390000
00126 *-----------------------------------------------------------------*01400000
00127 *                                                                 *01410000
00128 *                        LINKAGE SECTION                          *01420000
00129 *                                                                 *01430000
00130 *-----------------------------------------------------------------*01440000
                                                                        01450000
00131      EJECT                                                        01460000
00132 *-----------------------------------------------------------------*01470000
00133 *                                                                 *01480000
00134 *                        REPORT SECTION                           *01490000
00135 *                                                                 *01500000
00135 *            (MUST BE REMOVED FOR CICS/VS PROGRAMS.)              *01510000
00135 *                                                                 *01520000
00136 *-----------------------------------------------------------------*01530000
00137  REPORT SECTION.                                                  01540000
00138  RD  CONTROL REPORT                                               01550000
00139      CONTROLS ARE FINAL                                           01560000
00140      PAGE LIMIT IS 60 LINES                                       01570000
00141      HEADING 1                                                    01580000
00142      FIRST DETAIL 10                                              01590000
00143      LAST DETAIL 60.                                              01600000
00144  01  PAGE-HEAD TYPE IS PAGE HEADING                              01610000
00145      03  LINE 1.                                                 01620000
00146          05  COLUMN 1            PIC X(09)  VALUE "REPORT # ".   01630000
00147          05  COLUMN 15           PIC X(08)  SOURCE PROG-ID.      01640000
00148          05  COLUMN 20           PIC X(04)  VALUE "-CNTL-RPT"    01650000
00149          05  COLUMN 54           PIC X(28)                       01660000
00150                          VALUE "C O N T R O L    R E P O R T"    01670000
00151          05  COLUMN 114          PIC X(09)  VALUE "RUN DATE:".   01680000
00152          05  COLUMN 124          PIC X(08)  SOURCE TODAYS-DATE.  01690000
00153      03  LINE PLUS 1.                                            01700000
00154          05  COLUMN 60           PIC X(15)                       01710000
00155                          VALUE "STATE OF HAWAII".                01720000
00156      03  LINE PLUS 1                                             01730000
00157          05  COLUMN 1            PIC X(40)                       01740000
00158                          VALUE "DEPARTMENT OF BUDGET AND FINANCE". 01750000
00159 ***      MODIFY HEADER DESCRIPTIONS TO IDENTIFY USER'S AGENCY. *** 01760000
00159      03  LINE PLUS 1.                                            01770000
00160          05  COLUMN 1            PIC X(40)                       01780000
00161                          VALUE "EPM DIVISION"                    01790000
00162      03  LINE PLUS 2.                                            01800000
```

```
00160                05  COLUMN 1                PIC X(132)
00161                    VALUE ALL "-".
00162
00163            03  LINE PLUS 1.
00164                05  COLUMN 1                PIC X(31)
00165                    VALUE "C O N T R O L  M E S S A G E S:".
00166
00167            03  LINE PLUS 1.
00168                05  COLUMN 1                PIC X(132)
00169                    VALUE ALL "-".
00170
00171        01  PGM-MESSAGE TYPE DETAIL
00172            03  LINE PLUS 1.
00173                05  COLUMN 1                PIC X(132) SOURCE CNTL-RPT-LINE.
00174            OBJECT
00175        01  IO-STATISTICS TYPE IS CONTROL FOOTING FINAL
00176            03  LINE PLUS 1.
00177                05  COLUMN 1                PIC X(25)
00178                    VALUE "INPUT / OUTPUT STATISTICS".
00179            03  LINE PLUS 2.
00180                05  COLUMN 1                PIC X(25)
00181                    VALUE ALL "-".
00182            03  LINE PLUS 1.
00183                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00184                    SOURCE CTR-FILE1.
00185                05  COLUMN 14               PIC X(43)
00186                    VALUE "<--FILE1 RECORD COUNT".
00187            03  LINE PLUS 1.
00188                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00189                    SOURCE CTR-FILE2.
00190                05  COLUMN 14               PIC X(43)
00191                    VALUE "<--FILE2 RECORD COUNT".
00192            03  LINE PLUS 1.
00193                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00194                    SOURCE CTR-FILE3
00195                05  COLUMN 14               PIC X(43)
00196                    VALUE "<--FILE3 RECORD COUNT".
00197            03  LINE PLUS 1.
00198                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00199                    SOURCE CTR-FILE4
00200                05  COLUMN 14               PIC X(43)
00201                    VALUE "<--FILE4 RECORD COUNT".
00202            03  LINE PLUS 1.
00203                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00204                    SOURCE CTR-FILE5
00205                05  COLUMN 14               PIC X(43)
00206                    VALUE "<--FILE5 RECORD COUNT".
00207            03  LINE PLUS 1.
00208                05  COLUMN 1                PIC ZZZ,ZZZ,Z19
00209                    SOURCE CTR-FILE6
00210                05  COLUMN 14               PIC X(43)
00211                    VALUE "<--FILE6 RECORD COUNT".
00212            03  LINE PLUS 1.
```

```
013          05  COLUMN 1                PIC ZZZ,ZZZ,ZZ9                  02410000
014                                  SOURCE CTR-FILE9.                   02420000
015          05  COLUMN 14               PIC X(43)                         02430000
016                                  VALUE '<--FILE9 RECORD COUNT',        02440000
017      03  LINE PLUS 1.                                                  02450000
018          05  COLUMN 1                PIC ZZZ,ZZZ,ZZ9                   02460000
019                                  SOURCE CTR-FILE10.                  02470000
020          05  COLUMN 14               PIC X(43)                         02480000
021                                  VALUE '<--FILE10 RECORD COUNT'.       02490000
022      EJECT                                                           02500000
023  PROCEDURE DIVISION.                                                 02510000
024      MAIN-LOGIC-DRIVER                                               02520000
025          PERFORM 1100-INITIALIZE-TERM                                02530000
026                  1100-INITIALIZE-EXIT.                               02540000
027          PERFORM 2000-MAINLINE-LOGIC THRU                            02550000
028                  2000-MAINLINE-LOGIC-EXIT                            02560000
029                  UNTIL END-OF-PROGRAM ON.                            02570000
030          PERFORM 8000-DSP-CONSOLE-MESSAGE THRU                       02580000
031                  8000-DSP-CONSOLE-MESSAGE-EXIT.                      02590000
032          PERFORM 8500-PROGRAM-WRAPUP THRU                            02600000
033                  8500-PROGRAM-WRAPUP-EXIT.                           02610000
034          STOP RUN.                                                   02620000
035      1000-LOGIC-DRIVER-EXIT.                                         02630000
036          EXIT                                                        02640000
037      SKIP3                                                           02650000
038  1100-INITIALIZE.                                                    02660000
039          OPEN                                                        02670000
040              OUTPUT                                                  02680000
041                  CONTROL-RPT FILE.                                   02690000
042          MOVE ZEROS                   TO END-OF-PROGRAM.             02700000
043          MOVE CURRENT-DATE            TO TODAYS-DATE.                02710000
044          INITIATE CONTROL-REPORT.                                    02720000
045          MOVE CNTL-RPT-STARTED  TO CNTL-RPT-LINE.                    02730000
046          GENERATE PGM-MESSAGE.                                       02740000
047  1100-INITIALIZE-EXIT.                                               02750000
048          EXIT                                                        02760000
049      SKIP3                                                           02770000
050  2000-MAINLINE-LOGIC.                                                02780000
051                                                                      02790000
052      *****************************************************************02800000
053      *                                                               *02810000
054      *              INSERT PROCESSING CONTROLLING LOGIC HERE         *02820000
055      *                                                               *02830000
056      *              DESCRIBE MAIN-LINE PROCESSING MODULES.           *02840000
057      *                                                               *02850000
058      *****************************************************************02860000
059                                                                      02870000
060          DISPLAY '***** TEST OF MAIN LINE LOGIC *****'.              02880000
061          MOVE 'Y' TO END-OF-PROGRAM.                                 02890000
062                                                                      02900000
063  2000-MAINLINE-LOGIC-EXIT.                                           02910000
064          EXIT.                                                       02920000
065                                                                      02930000
066      EJECT                                                           02940000
067      *****************************************************************02950000
068      *              PARAGRAPHS OF FUNCTIONS THAT ARE                 *02960000
069      *              "PERFORM"ED BY THE MAINLINE CONTROLLING          *02970000
070      *              PROCESSING LOGIC ARE INSERTED HERE.              *02980000
071      *****************************************************************02990000
072      SKIP2                                                           03000000
```

```
00234  880-ECHO-CONSOLE-MESSAGE.                                        03010000
00001  ********************************************************************03020000
00002  *    INCLUDE STANDARD END OF JOB MESSAGE/RETURN-CODE ROUTINE      *03030000
00003  ********************************************************************03040000
       ++INCLUDE   ZZSMTEXA1                                            03050000
00234  880-ECHO-CONSOLE-MESSAGE-EXIT.                                   03060000
00237      EXIT.                                                        03070000
00238                                                                   03080000
00239                                                                   03090000
00240  890-PROGRAM-WRAPUP.                                              03100000
00241      TERMINATE CONTROL-REPORT.                                    03110000
00242      CLOSE                                                        03120000
00242              CONTROL-RPT-FILE.                                    03130000
00240  890-PROGRAM-WRAPUP-EXIT.                                         03140000
00245      EXIT.                                                        03150000
       ********************************************************************03160000
```