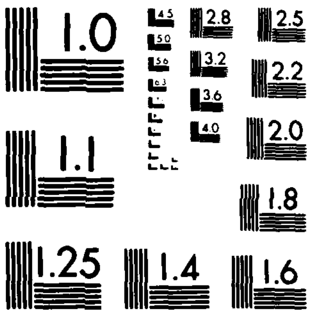
1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(1)

AD-A144 126

# CODAP80 RELEASE 83.1 SYSTEMS MANUAL

EXECUTIVE AGENT
FOR
JOINT TASK
ANALYSIS SUPPORT

DTIC FILE COPY

DECEMBER 1983

84 08 06 015

50272-101

| | | | |
|---|---|---|---|
| **REPORT DOCUMENTATION PAGE** | 1. REPORT NO. DOD/DF-84/006b | 2. BIN-602 | 3. Recipient's Accession No. |

4. Title and Subtitle
COMPREHENSIVE OCCUPATIONAL DATA ANALYSIS PROGRAMS 80 (CODAP80)
Systems Manual

5. Report Date
JANUARY 1984

6.

7. Author(s)
N/A

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N/A
(G) N/A

12. Sponsoring Organization Name and Address
NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

13. Type of Report & Period Covered
FINAL RELEASE 83.1

14.

15. Supplementary Notes

SOURCE CODE FOR CODAP80 PROGRAMS.
for magnetic tape see

16. Abstract (Limit: 200 words)

CODAP80 is an enhanced IBM version of the Comprehensive Occupational Data Analysis Programs. The software system is used to process occupational information and includes programs that range from data entry to statistical analysis. CODAP80 is based on a database management concept which allows the job analyst more versatility in analysis than its predecessor. Included with the system are four manuals: the CODAP80 User's Manual, Job Analysis Manual, Systems Manual, and Executive Summary.

[illegible]

17. Document Analysis a. Descriptors

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

| | | |
|---|---|---|
| 18. Availability Statement: RELEASE UNLIMITED | 19. Security Class (This Report) UNCLASSIFIED | 21. No. of Pages 35 |
| | 20. Security Class (This Page) UNCLASSIFIED | 22. Price |

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

## FOREWORD

The Comprehensive Occupational Data Analysis Programs (CODAP), a software package developed by the United States Air Force, is in use by all the United States military services and numerous other agencies throughout the world. Of the two predominant versions of CODAP, the IBM version has not kept pace with the continuing development of the UNIVAC version.

In 1978 the Navy Occupational Development and Analysis Center, a detachment of the Naval Military Personnel Command, and serving as Executive Agent for Joint Task Analysis Support for the Department of Defense, initiated a project to develop an enhanced IBM version of CODAP which would be less machine dependent than the existing IBM version, easy for non-programmers to learn and use, and which would provide the capability to implement new analysis approaches for analyzing occupational data. The funding for this project was provided by the United States Navy, Marine Corps, and Coast Guard.

As a result of this project, CODAP80, an enhanced version of IBM CODAP, was developed by Texas A&M University. This manual is one of four CODAP80 manuals which were developed to accompany the CODAP80 system. The four manuals are the CODAP80 Executive Summary, the Job Analysis Manual, the User's Manual, and the Systems Manual.

A-1

2

## TABLE OF CONTENTS

# CODAP80

## INTRODUCTION

The CODAP80 occupational analysis computer system consists of two major sets of software: the software that comprises database creation, and the software that comprises the interpreter. The entire CODAP80 system is written in FORTRAN according to the specifications outlined in the document IBM System/360 and System/370. FORTRAN IV Language, order number GC28-6515-10.

CODAP80 was developed using the IBM FORTRAN IV G1 compiler (Release 2.0) operating under MVS/JES3 on an Amdahl 470/V8 computer, and on an IBM 370/148 computer operating under VM/SP. The system should compile correctly on IBM's H compiler. Users of the VS FORTRAN compiler will find that CODAP80's interpreter subroutine SQUASH produces an underflow abend condition.

## ORGANIZATION OF THE SYSTEMS MANUAL

The Systems Manual is organized into two major sections: database creation and the interpreter. The characteristics and attributes of the two system sections is discussed, with subroutines being identified and file layouts displayed. All file layouts pertain to the data to be found in the sample set of data introduced in the User's Manual. The operation of the CODAP80 interpreter is outlined, as well as token recognition and the access and retrieval of database information. File space calculation equations for the different routines in the CODAP80 system can be found in the User's Manual. JCL execution setups for the CODAP80 system may also be found in the User's Manual.

## CODAP80 RELEASE

The specifications outlined in the Systems Manual pertain to the 83.1 release of CODAP80.

## THE DATABASE CREATION ROUTINES

### INTRODUCTION

The database creatioh routines consist of the programs INPSTD, OGROUP and REARNG. All three of these programs execute in under 512K bytes of memory.

### INPSTD

The INPSTD database creation routine is the primary input routine for all the data of an occupational study. Files unique to the INPSTD routine are:

| | | |
|---|---|---|
| CONTROL | - | FT03F001 |
| INPFILE | - | FT02F001 |
| DATA | - | FT04F001 |
| VARCOM | - | FT10F001 |
| SYMTAB1 | - | FT12F001 |
| DECODE | - | FT17F001 |

### CONTROL

Data set CONTROL is a card image sequential file containing remarks for the history, task and secondary variables, as well as decode titles and format fields specifications. On CODAP80's host computer, it was blocked FB/80/6160. The exact layout of the CONTROL file can be found in Appendix B.

### INPFILE

Data set INPFILE is a direct access file with 3600 byte records. Following execution of the INPSTD database creation routine, it contains the incumbent history, task (relativized) and secondary data. Following the incumbent data are two words containing the sum of the row time spent responses from the incumbent and the number of nonzero responses to tasks. The layout of INPFILE after SAMPLEDATA80 INPSTD execution may be found in Appendix B.

### DATA

Data set DATA is a card image sequential file containing the raw incumbent responses to history, task and secondary indicies. The information in this data set is processed by INPSTD and written to data set INPFILE. On CODAP80's host computer, it was blocked FB/80/6160. The exact layout of the DATA file can be found in Appendix B.

## VARCOM

Data set VARCOM is a direct access file with 244 byte records. Following execution of the INPSTD database creation routine, it contains the history, task and secondary remarks that were processed from the CONTROL file. The first word of the VARCOM file contains the length in characters of the remark, and words 2-61 contain the variable remark stored in packed format (A4). The layout of VARCOM after SAMPLEDATA80 INPSTD execution can be found in Appendix B.

## SYMTAB1

Data set SYMTAB1 is a direct access file with 48 byte records. It is the inital symbol table and contains pointer values and data counts. It serves primarily as input to the OGROUP and REARNG database creation routines. The layout for SYMTAB1 can be found in Appendix B.

## DECODE

Data set DECODE is a direct access file with 120 byte records. It contains the decode titles that were processed from CONTROL by INPSTD. The file can be thought of as a pseudo-indexed sequential file. See Appendix B for the layout of the DECODE file.

## INPSTD SUBROUTINES

A listing of the INPSTD subroutines may be found on page 5.

## OGROUP

The OGROUP database creation routine is the main clustering routine in the COAP80 system. Files unique to OGROUP are:

GRPFILE - FT15F001
GRPHSN - FT16F001

## GRPFILE

Following execution of the OGROUP database creation routine, the GRPFILE contains all the resultant information from the incumbent clustering. GRPFILE is a direct access file with records of 12960 bytes. Between and within overlap values reside in GRPFILE, as well as the colapsing process for the diagram. See Appendix B for the configuration of GRPFILE.

## GRPHSN

Data set GRPHSN is a direct access file with records of 40 bytes. Each word of a record corresponds to an incumbent, and contains, positionally, the HSN location an incumbent should have. *GRPHSN acts as input to the* REARNG database creation routine. See Appendix B for the layout of file GRPHSN.

## OGROUP SUBROUTINES

A listing of the OGROUP database creation subroutines can be found on page 6.

## REARNG

The REARNG database creation routine transposes the incumbent data in INPFILE so that, instead of a record holding all the variables for an incumbent, a record holds all the incumbents for a variable. If OGROUP was executed, then the incumbents in a record are in HSN order. Files unique to REARNG are:

| | |
|---|---|
| DATABASE | - FT01F001 |
| SYMTAB2 | - FT13F001 |

## DATABASE

Data set DATABASE is a direct access file with records of 4000 bytes. The layout of the DATABASE file is as follows:

History variables will be written first. They will be unpacked (i.e., each incumbent has a value) and will be number of incumbents (NINC) words long.

Task variables will be written in the first word of the record that follows immediately after the end of the history variables. Task variables will be written in packed format (i.e., missing values are to be removed). Secondary variables will start in the word that immediately follows the last word of the last task variable. Secondary variables will be written in packed format.

The task variable membership vectors will begin in the first word of the record that follows the last secondary row. The task membership vectors will contain real numbers ranging from 1 to 9. The secondary variable membership vectors will begin in the word that immediately follows the last word of the last task membership vector.

Following the task and secondary membership vectors are the tables, that define the locations of the task or secondary row vectors and their associated membership vectors. Each row has two entries in the table. The first entry for a row consist of the number of the record (relative to the

## CODAP80 INPSTD SUBROUTINES
## ENTRY POINT IS MAIN1

```
ATODIG
COMPRS
DASHCK
DECCVT
DECTIT
ERRRTN
HVAR
INCDAT
MAIN1
NEXTID
PCTTIM
PREPCM
SVAR
SYMTAB
TVAR
VALDAT
VALDEC
VALID1
VALID2
VARCOM
ZR
```

## CODAP80 OGROUP SUBROUTINES
## ENTRY POINT IS MAIN2

```
MAIN2
MASIGN
MBULDM
MCALCW
MCALSP
MDGRAM
MDOROW
MFILLX
MFILST
MFILWC
MFMAX
MFNDXP
MFXMAX
MGETW
MGRABH
MGRABR
MGRABS
MGRABT
MGRABX
MGRBPS
MGRPIT
MGRUP6
MGTRAW
MHSNFO
MIGET
MINITD
MINITK
MINPC
MKOUNT
MNDIGT
MNUMF
MNUMI
MOGET1
MOPUT1
MOUTA
MOUTD
MOUTFO
MOVLGT
MOVLPT
MPBLOK
MPICSG
MRDSK
MRDSKL
MSHOWA
MSHOWM
MVCNT
MWDSK
MWDSKL
MWTHSN
SMLARM
```

9

start of the task rows) that the row begins in times 1000, plus the number of words to skip from the start of the record to reach the first word of the row. The second entry in the table consists of the length of the row in words.

The table for the task rows will begin in the first word of the next record after the secondary membership vectors. The table will be Ntask * 2 words long and will span (Ntask*2-1)/1000+1 physical records. The table for the secondary rows will begin in the first word of the record that immediately follows the end of the task table. The secondary table will be NSEC*2 words long and will span (NSEC*2-1)/1000+1 physical records.

Following the tables the system rows Rawsum and Nonzero will appear. Each of these rows will be NINC words long and will span (NINC-1)/1000+1 physical words. Each rows will start in the first word of a physical record.

The layout of file DATABASE following execution of the REARNG database creation routine on the SAMPLEDATA80 data can be found in Appendix B.

## REARNG SUBROUTINES

A listing of the REARNG database creation subroutines can be found on the following page.

## CODAP80 REARNG SUBROUTINES
## ENTRY POINT IS MAIN 3

```
BLANKP
CPYIN
GETWOG
GRBPS
GTD
INPUT
MAIN3
PRNPST
PTD
PUTGR
PUTPS
RDISKO
SORTIN
SYMCVT
TRNSPA
TRNSPB
```

11

## THE CODAP80 INTERPRETER

The CODAP80 INTERPRETER is used to process and display information residing on a CODAP80 database. The amount of memory required to execute the interpreter is 820K bytes (non-overlayed).

### CODAP80 ERRORFIL

The first step in installing the interpreter is to generate the error file. On the following pages are the CODAP80 error messages and a FORTRAN program (ERRORCVT) that will take the error messages and generate the errorfil.

### CODAP80 INTERPRETER SUBROUTINES

On page 13 can be found a listing of the CODAP80 interpreter subroutines.

### INTERPRETER OVERVIEW

The function of the interpreter is to input CODAP80 source code statements, ensure that they are syntactically correct, and then perform the operations indicated by the statements. The first phase of the interpreter, the syntax analyzer and symbol table builder (SYNSYM), involves analyzing the syntax of the statements to ensure that they are correct, and translating the source code into an intermediate form more suitable for computer processing. The inputs to the interpreter used by this phase are the CODAP80 source code, and the permanent symbol table (PST, which tells where all permanently stored information is located; the outputs of this phase are an echo print of the source code and any appropriate error messages, intermediate printing for use by maintenance programmers, an intermediate form of the source code which is a stack with a numeric representation of the source code and tables for symbols, strings, and constants. The second phase (EXECUT) carries out the operations on the data specified by the CODAP80 commands. Its inputs are the stack with the numeric representation of the source code, the three tables, the PST, the permanent data files, scratch files, and data (if any) for the INPUT command. Its output are additions to the PST and permanent data files and printed reports.

Additional inpu*s to SYNSYM that come from the main. routine are an array to build the stack of numeric tokens, storage space to build the working symbol, string, and constant tables in, and two variables SUCCES, and PRTLVL (print level). The storage space for tables is used to produce the Working Symbol Table (WST), the string table, and the constant table. The WST is used to keep track of all variables used within a given run. If the variable exists on the PST it is copied into the WST, if it is not on the PST a new entry for it is made in the WST. In this way all variables that are accessed within a run can be located via the core resident WST. The

## ERROR MESSAGES
## CODAP80 INTERPRETER ERROR MESSAGES

```
001 UNRECOGNIZABLE CHARACTER.
002 STRING EXCEEDS 240 CHARACTERS.
003 UNRECOGNIZABLE TOKEN.
004 INVALID RELATIONAL OPERATOR.
005 INVALID BOOLEAN OPERATOR.
006 INVALID SYSTEM VARIABLE.
007 TOO MANY SYMBOLS USED.
008 NUMBER OF TOKENS IN SOURCE CODE EXCEEDS STACK SIZE.
009 INTEGER PORTION OF SYSTEM VARIABLE TOO LARGE.
010 RESERVED FOR
011 MORE MESSAGES
012 FROM GTOKEN
013 EXPECTING A COLUMN.
014 ID NOT PREVIOUSLY DEFINED.
015 EXPECTING A "ROWS" OR "COLUMNS" KEYWORD.
016 EXPECTING A "FOR" KEYWORD.
017 A RESERVED WORD HAS BEEN USED FOR A VARIABLE NAME.
018 EXPECTING ASSIGNMENT OPERATOR.
019 AN INVALID FUNCTION HAS BEEN SPECIFIED.
020 REMARK NOT FOUND.
021 LEFT PAREN MISSING.
022 UNBALANCED PARENTHESES.
023 INVALID VARIABLE NAME.
024 AN ILLEGAL RANGE STATEMENT HAS BEEN SPECIFIED.
025 A GROUP NAME HAS NOT BEEN SPECIFIED.
026 A MODULE NAME HAS NOT BEEN SPECIFIED.
027 EXPECTING AN "ON" KEYWORD.
028 EXPECTING UNDEFINED ID.
029 EXPECTING A ROW.
030 EXPECTING AN "IN" KEYWORD.
031 EXPRESSION NOT ENCLOSED IN PARENTHESES.
032 INVALID SYNTAX.
033 VARIABLES OUT OF SEQUENCE.
034 "BEGIN" NOT FIRST STATEMENT.
035 SYSTEM VARIABLE NOT VALID HERE.
036 EXPECTING "USING" KEYWORD.
037 A SINGLE VALUED VARIABLE OR CONSTANT IS EXPECTED.
038 RELATIONAL OPERATOR NOT VALID HERE.
039 EXPECTING A SYSTEM COLUMN.
040 EXPECTING HISTORY VARIABLE IN SEQUENCE.
041 EXPECTING TASK VARIABLE IN SEQUENCE.
042 EXPECTING SECONDARY VARIABLE IN SEQUENCE.
043 EXPECTING "FORMAT" KEYWORD.
044 EXPECTING FORMAT STATEMENT.
045 THRU NOT VALID HERE.
046 EXPECTING SYSTEM ROW.
047 EXPECTING "THEN".
048 MULTIPLE CREATES NOT ALLOWED IN "IF".
049 EXPECTING "ELSE".
050 EXPECTING A PERIOD.
051 EXPECTING A COMMAND KEYWORD.
052 EXPECTING RELATIONAL OPERATOR.
053 "+" ONLY OPERATION ALLOWED HERE.
054 NOT A VALID KEYWORD FOR THIS COMMAND.
055 STUDY ID DOES NOT AGREE WITH DATA BEING ACCESSED.
056 A COLUMN DESIGNATION NOT VALID HERE.
057 A GROUP DESIGNATION NOT VALID HERE.
058 UNRECOGNIZED ID.
059 EXPECTING SINGLE VALUE VARIABLE.
060 EXPECTING SYSTEM COLUMN IN SEQUENCE.
061 EXPECTING SYSTEM GROUP IN SEQUENCE.
062 EXPECTING SYSTEM COLUMN OR SYSTEM GROUP.
063 INVALID SYNTAX FOR A MODULE LIST.
064 EXPECTING RIGHT PAREN.
065 EXPECTING "HEADING" KEYWORD.
```

## ERROR MESSAGES
## CODAP80 INTERPRETER ERROR MESSAGES
## (continued)

```
066 HEADING STRING CANNOT EXCEED 131 CHARACTERS.
067 THIS FUNCTION HAS BEEN SPECIFIED MORE THAN ONCE.
068 EXPECTING FUNCTION SPECIFICATION.
069 BOTH THE VERTICAL AND HORIZONTAL AXES CONSIST OF THE SAME DATA TYPE.
070 EXPECTING "BY" KEYWORD.
071 EXPECTING CHARACTER STRING.
072 NO MORE THAN 10 TITLE LINES MAY BE REQUESTED.
073 EXPECTING "COLUMNS" OR "COLS" KEYWORD.
074 EXPECTING "ROWS" KEYWORD.
075 EXPECTING KEYWORD TO SPECIFY OVERLAPPING ALGORITHM.
076 EXPECTING "MAXIMIZE" SPECIFICATION.
077 EXPECTING "HSN" KEYWORD.
078 EXPECTING "LOHSN" KEYWORD.
079 EXPECTING "HIHSN" KEYWORD.
080 EXPECTING HEADING STRING OR PERIOD.
081 EXPECTING HEADING STRING.
082 EXPECTING "SIMCOF" KEYWORD.
083 EXPECTING "WITHIN" KEYWORD.
084 EXPECTING NEW ID.
099 CHECK SYNTAX FOR A CORRECT KEYWORD (EITHER "ROWS" OR "COLUMNS").
152 NO MEAN OR STANDARD DEVIATION FOUND.
153 ASSIGNMENT OPERATOR IS MISSING.
154 EXPECTING A CONSTANT.
155 REPEATED "MEAN".
156 REPEATED "STD".
157 "STD" IS MISSING.
158 "MEAN" IS MISSING.
159 AN ASSIGNED VALUE OF STD MUST NOT BE LESS THAN ZERO.
160 TOKEN IS NOT A CREATED/SYSTEM ROW/COLUMN/MODULE/GROUP.
161 MUST BE TVARS/HVARS/SVARS/TASKS ONLY.
162 LENGTH OF CREATED ROW/COLUMN/MODULE/GROUP MUST BE GREATER THAN ZERO.
163 NTASK = 0. CAN'T GENERATE ROWS.
164 NHIST = 0. CAN'T GENERATE ROWS.
165 NSEC  = 0. CAN'T GENERATE ROWS.
166 NINCS = 0. CAN'T GENERATE ROWS.
170 "TAPE" OR "CARD" KEYWORD IS MISSING IN COPY COMMAND.
171 "B","BINARY","D","DISTANCE","D2","DSQUARE","OVL",OR "OVERLAP" ONLY
172 "MINMEM" OR "HEADING" IS EXPECT HERE.
173 "RESET" MUST BE PRECEEDED BY "NOSKIP".
174 "CUM" OR "COUNT" MUST BE PRECEEDED BY "NOSKIP"&"RESET" RESPECTIVELY
175 MINMEM MUST BE EQUAL OR GREATER THAN 2.
180 EXPECTING "N" IN ADDATA COMMAND.
181 EXPECTING "N=" OR "N:=" BEFORE A CONSTANT.
182 REPEAT FACTOR MISSING IN ADDATA COMMAND.
183 EXPECTING "=" OR ":="
184 A CONSTANT IS MISSING IN ADDATA COMMAND.
185 "(" MISSING.
186 UNRECOGNIZED TOKEN IN ADDATA COMMAND.
187 A CONSTANT MISSING FOLLOWING A DASH (-).
188 ")" MISSING FOLLOWING A CONSTANT LIST.
189 THE VALUE OF N AND NUMBER OF CREATED IDS ARE NOT THE SAME.
190 NUMBER OF NUMERIC FIELDS IN FORMAT MUST BE 1000 OR LESS.
```

# ERRORCVT
## PROGRAM TO GENERATE THE INTERPRETER ERROR FILE

```
//  EXEC  FG.REGION=256K
//FT01F001 DD DSN=ERRORS.DISP=OLD
//FT02F001 DD DSN=ERRORFIL.UNIT=SYSDA.DISP=(NEW.CATLG).
//             DCB=(DSORG=DA).SPACE=(320.(150.1))
//SOURCE   DD *
C-----------------------------------------------------C
C ERROR FILE PROGRAM (ERRORCVT).                      C
C PROGRAM TO GENERATE THE ERROR FILE FOR THE          C
C CODAP80 INTERPRETER.  THE ERRORS (IN CARD IMAGE     C
C FORM) ARE READ FROM FILE 1 AND WRITTEN TO           C
C FILE 2 (DSORG=DA).                                  C
C-----------------------------------------------------C
      DEFINE FILE 2 (150,80,U,IREC)
      INTEGER IERR(80)
      DATA J/1/
    1 J=J+1
      DO 2 I=1,80
    2 IERR(I)=0
      DO 3 I=1,80
      READ(1,101,END=100) IERR(I)
  101 FORMAT(I3.1X.76A1)
    3 CONTINUE
      IREC=J
      WRITE(2'IREC) IERR
      GO TO 1
  100 IF(I.EQ.1) GO TO 200
      IREC=J
      WRITE(2'IREC) IERR
  200 NUMERR=(J-2)*80+I-1
      IRECST=J-1
      IF(I.EQ.1) IRECST=IRECST-1
      WRITE(2'1) NUMERR.IRECST
      IREC=IRECST
      REWIND 1
  201 READ(1.101.END=300) (IERR(I).I=1.77)
      WRITE(2'IREC) (IERR(I).I=1.77)
      GO TO 201
  300 READ(2'1) NUMERR.IRECST
      WRITE(6.601) NUMERR.IRECST
      K=IRECST-1
      DO 401 I=2.K
      IREC=I
      READ(2'IREC) IERR
      WRITE(6.601) IERR
  601 FORMAT(1X.(1X.30I4))
  401 CONTINUE
      ITREC=NUMERR+IRECST-1
      DO 402 I=IRECST.ITREC
        IREC=I
        READ(2'IREC) (IERR(K),K=1.77)
        WRITE(6.602) (IERR(K).K=1.77)
  602   FORMAT(1X,I4.1X.76A1)
  402 CONTINUE
      STOP
      END
//SYSIN    DD *
```

**CODAP80 INTERPRETER SUBROUTINES**
**ENTRY POINT IS MAIN4**

| | | | | | |
|---|---|---|---|---|---|
| ACCMOD | CORRPM | GROUP | PCNTAV | RELYCO | SRELEX |
| ADDATE | CORRPR | GROUPS | PCNTC | RELYE | STDA |
| ADDATS | CORRRM | GRPCHK | PFUNC | RELYS | STDAAV |
| ADDCON | CORRRT | GRPIT | PFUNCA | RELYSM | STDAC |
| ADDID | CORRS | GRPLST | PFUNCC | RELYSS | STDE |
| ADDREM | CREATE | GTCHAR | PFUNCR | REPCRE | STDE1 |
| ADDSTR | CREATS | GTOKEN | PICKSG | REPORE | STDE3 |
| ALLCOR | CRELEX | HASH | POP | REPORS | STDP |
| ALPHA | CTBMAN | HEADNG | POPIT | RETCCR | STDPAV |
| ARITHE | CTRIPS | HSNFO | POSTFX | RETCNS | STDPC |
| ASSIGN | CUMLST | ID | PRALL | RETGM | STDS |
| AVALUE | DATINT | IDCR | PRALLS | RETLT1 | STRING |
| AVALUS | DECODE | IDENTE | PRFORM | RETLT2 | STRIPS |
| AVGA | DELIM | IDENTS | PRHEAD | RETMEM | STRLEN |
| AVGAAV | DESCRE | IGET | PRHORZ | RETPOS | STRRET |
| AVGAC | DESCRS | INFMT | PRINTE | RETRC | SUM |
| AVGP | DIAGRS | INITD | PRINTS | RETSVV | SUMAV |
| AVGPAV | DIGIT | INITK | PRLOAD | RETTRP | SUMC |
| AVGPC | DIGRAM | INITLZ | PRSORT | RLIST | SYNANL |
| BACKUP | DOROW | INPUTE | PRSVV | ROW | SYNSYM |
| BEGINE | ECODE | INPUTS | PRTCT | ROWCHK | SYNTAX |
| BEGINS | ENDE | INRPRN | PRTMSG | ROWNUM | SYSVAR |
| BINSRC | ERRPRT | INUNPK | PRTPST | RPCRHD | TCOUNT |
| BLDTRP | EXECOM | ITOA | PRTRVE | RPCRNR | TRANSA |
| BLDWST | EXECTR | KEYWRD | PRTST | RPCRRM | TRANSB |
| BOOLOP | EXECUT | KOUNT | PRTV12 | RPSYCT | TRIPLS |
| BUILDM | FILLST | MAIN4 | PRTWST | RPSYGP | TRNSPS |
| CALCSP | FILLWC | MODCHK | PRVTRK | RPSYMD | VARHD |
| CALCW | FILLX | MODLST | PSETUP | RPSYRW | VARSUE |
| CBOOL | FINDXP | MODULE | PSTADD | RRELEX | VARSUS |
| CLIST | FIXMAX | MROWLT | PSTEND | RTOA | VARSU2 |
| CLUST | FMAX | N | PTOKEN | RTOPND | VECWRT |
| CLUSTE | FPA | NAV | PUSH | RTRWRC | WDISK |
| CLUSTS | FRMATR | NC | PUSHBS | RTSCOL | WDISKL |
| CNEWVC | FRODST | NDIGIT | PUSHIT | RTTRIP | WRTRVE |
| CNSTNT | FSERCH | NEWID | PUSHPT | RTVTOK | WRTV12 |
| COLCHK | FSORT | NUMF | PUSHRP | SAVHSN | WSETUP |
| COLERR | FULLAS | NUMI | PUSHRT | SELECE | WSTADD |
| COLNUM | FUNC | OGET1 | PUTD | SELECS | WSTREP |
| COLUMN | GCOLST | OPUT1 | RANDK | SETSTK | WSTR12 |
| COMENT | GETD | OUTA | RANDMK | SHIFTR | WSTSUB |
| CONTOK | GETRAW | OUTD | RANDOE | SHOWA | |
| COPYE | GETW | OUTFO | RANDOS | SHOWM | |
| COPYE1 | GMLEN | OVLGET | RANDU | SIMLAR | |
| COPYE2 | GMRCCT | OVLPUT | RBOOL | SINVAL | |
| COPYIN | GRABH | PA | RDISK | SKIP | |
| COPYS | GRABR | PAIO | RDISKL | SNVAL2 | |
| COP2PM | GRABS | PASORT | RELATE | SORTID | |
| CORRE | GRABT | PBLOCK | RELOP | SORTLS | |
| CORRLT | GRABX | PCNT | RELYAC | SQUASH | |

16

string and constant tables are used to collect strings and constants. SUCCES is a variable that indicates if any errors are found. It is set to 1 when passed to SYNSYM. If an error is found it is set to 0. This means that if SUCCES is 0 on return from SYNSYM an error was found; so EXECUT is not called. PRTLVL is a variable that will allow a maintenance programmer to get extra information printed out that should help in modifying the system. If set to 2 only the source code and error messages are printed out. PRTLVL will be set to 2 for normal execution. If PRTLVL is 1 the PST is printed first, then the echo print of the source code and error messages, followed by the WST, the string table, and the constant table. If PRTLVL is 0 all the previous information will be printed plus the numeric value of each parameter will be printed as it is recognized.

SYNSYM first calls IDENTS to identify the next command and returns an associated number. Based on this number SYNSYM calls one of the syntax analysis routines. The routine called will analyze the syntax of a particular command. If the number indicates the END command was found, SYNSYM returns to the main routine. Each of the syntax analysis routines calls a subroutine, which gets the next parameter of the source code and returns its numeric representation, several times checking for invalid syntax or until a period is found and then returns. When errors are found ERRPRT is called to print the message and set SUCCES to 0. ERRPRT will print a dollar sign under the last character of the parameter in error and the message on the next line. Whenever possible syntax analysis is resumed after finding an error, but, if not possible, parameters are skipped until a period is found, then a return is made.

The basic unit that the syntax analysis routines work with is the token. The parameters of commands are tokens. A token is the smallest meaningful aggregate of characters. For example: CREATE ROW FOR G3 FRED:=A+3 'remark'. has 11 tokens. They are CREATE, ROW, FOR, G3, FRED, :=, A, +, 3, 'remark', and '.'. GTOKEN (Get Next Token) is the routine that picks these out of the source code and assigns a numeric value. For ids, constants, and strings some information is put into tables, the numeric code points to the proper position in the tables. Tokens from 150000 to 159999 represent ids and point to positions in the WST. The ids are added using a hashing scheme. Tokens from 20000 to 29999 point to the constant table. Tokens from 10000 to 19999 point to the string table.

Rows, columns, groups, and modules will all be stored as records on the same file.

The EXECUT phase is broken down very similarly to SYNSYM. IDENTE is first called to identify a command. Based on the number IDENTE returns, one of the execution routines is called. The execution routines do whatever data manipulation is required then return. The execution routines do not update the permanent files or PST. Any information they generate is stored on scratch files. When the END command is reached the data and entries of the WST that are to be added to the permanent information are copied over. This means that if a power failure or some problem occurs in the middle of EXECUT, the CODAP80 program can be rerun with no problem, since the permanent information has not been changed. If something happens after the END command is found, while the scratch files are being copied over, a separate program that does just the copying over can be run, since the scratch files should remain intact for a while. This means there is never any reason to have to restore the database to a previous state so that a rerun can be made.

## TOKEN RECOGNITION

The tokens of the language are recognized by GTOKEN (Get Next Token). GTOKEN recognizes tokens without regard to context. Because the language requires a blank, comma, or delimiter between tokens, there is never a case where GTOKEN has to be aware of what the previous or succeeding token is, in order to recognize a token. It is never necessary to scan more than one character ahead of a token in order to recognize that token. Because the cards with the source code on them are syntax analyzed as they are read in, it is not possible to back up from the beginning of a card to the end of the previous card. This means that strings, comments, and constants are the only tokens that may cross card boundaries, since they can be identified from the first character.

The inputs to GTOKEN are the source code, the variables PRTLVL, SUCCES, ACTUAL, and LENGTH, the PST, and storage space for the WST, string table, constant table, and stack of numeric tokens. TOKEN is used to return the numeric code for the token. ACTUAL is used to return the source code token when the token is an id. LENGTH is used to return the number of characters in the id. The outputs are an echo print of the source code and any error messages about invalid tokens, a printout of the numeric token if called for by PRTLVL, another string, constant, or id in the appropriate table, the numeric token in the stack, TOKEN, SUCCES, LENGTH, and ACTUAL.

GTOKEN is broken down into several subroutines. The first series of routines called, attempt to recognize a token from the source code. The parameter FOUND is passed to each of these routines initialized to 0. On return from each routine FOUND is checked for 0 or 1. If it is 0 the next routine is called. If it is 1 this means the routine recognized a token, so GTOKEN returns to its calling routine. If the recognition routine does not recognize a token, it calls a routine called BACKUP so that the next recognition routine will begin its scan of the source code at the same position that the previous routine did.

The first routine called to attempt to recognize a token is STRING. STRING first skips past any blanks or commas. STRING looks for a string which is anything enclosed in quotes. If a string is found the subroutine ADDSTR is called to add the string to the string table. ADDSTR returns the index of where the string was added into the table. STRING adds this index to 10000 to generate the numeric token. FOUND is set to 1 and a return is made.

COMMENT is called next, which attempts to recognize a comment which is anything enclosed in number signs (#). If a comment is found, FOUND is set to 1 and a return made. GTOKEN does not return when a comment is found. Because comments are intended only for the writer of the CODAP code they are ignored by the interpreter. GTOKEN will start the series of calls to the recognition routines again beginning with STRING after finding a comment. No numeric token is generated for comments.

CNSTNT is called next to attempt to identify a constant which is a string of digits that may include a decimal point. If a constant is found it is converted into numeric form and ADDCON is called to add it to the constant table. ADDCON returns to the index of where the constant was added into the table. CNSTNT adds this index to 20000 to generate the numeric token. FOUND is set to 1 and a return made.

RELOP is called next which attempts to recognize a relational operator. Relational operators are things such as .EQ., .NE., .LT., and =. If a relational operator is recognized it is assigned a numeric token of 30000 plus. The exact numeric token assignments can be seen in appendix.

BOOLOP is called to recognize boolean operators. Boolean operators are .AND., &, .OR., and ¦. If a boolean operator is recognized it is assigned a numeric token of 40000 plus as indicated in appendix.

DELIM attempts to recognize the next token as a delimiter. Delimiters are: .,+,-,/,*,**,:=,(,), and ;. If a delimiter is found it is assigned a numeric token of 50000 plus.

FUNC attempts to recognize functions. The functions are; LOG, SQRT, ACUM, DCUM and so forth. If a function is found it is assigned a numeric token of 60000 plus.

SYSVAR attempts to recognize system variables which are: the system groups produced by the clustering program such as G1 and G5, history variables that are input by INPSTD such as H1 and H4, task variables such as T1 and T4, secondary variables such as S1 and S3, incumbents such as I1 and I2, and computed variables such as RAWSUM and SIMCOF. If one of these is recognized it is assigned a numeric value of 70000 plus for computed variables, 80000 plus the integer portion of the history variable for history variables, 90000 plus the integer portion of the task variable for task variables, 100000 plus the integer portion of the secondary variable for secondary variables, 110000 plus the integer portion of the group variable for group variables, and 160000 plus the integer portion of the incumbent for incumbents. Groups can take on the range 110000-139999 and incumbents are anything 160000 or greater.

KEYWRD is called next to look for a key word. Key words are words that have special meaning to the CODAP language such as CREATE, AVALUE, IF, and THEN.

ID is called to check for an id. Since this is the last routine called the next token has to be an id unless it is a character that is not meaningful to CODAP80. If an id is found BLDWST is called to add the id to the WST. BLDWST is further explained in the next chapter. BLDWST returns the index of where the id was added into the table. ID adds this index to 150000 to generate the numeric token.

If none of these routines can recognize the token it means an invalid character was entered in the source code. GTOKEN will then skip that character after printing an error message and begin the calls to the recognition routines again.

## THE SYMBOL TABLE AND ITS ACCESS ROUTINES

The two symbol tables used in the interpreter are used to keep track of where all information is located. The PST keeps track of all information on the permanent files and is kept on a disk file. The WST keeps track of where any information needed within a single CODAP80 program is located. The WST may point to information on the permanent files or to information on the scratch files. The WST is created in a table in core.

The PST is initially created by INPSTD and contains 11 entries each with 12 elements.

The headings on each of the 12 elements do not really apply to the first 11 entries, only to the 12th and following entries, which keep track of data that is added to the files through the use of the CODAP80 language.

Element 4 of entries 1-5 is used to keep track of where the next free labels and records are located. Entries 1-3 give information about the history, task and secondary records established by INPSTD. For each of the 3 entries, element 6 tells the number of variables in the database, element 10 gives the record number of the first record of that type variable, and element 11 gives the record number of where the first remark is stored. Each variable in the system has a 240 character string called a remark associated with it. Entry 4 gives information about the groups produced by the clustering program; element 6 gives the number of groups produced and element 10 gives the record number of where the first of the two vectors, LOWHSN and HIGHHSN, is stored. The fifth entry has the number of incumbents for the study, in element 6, the sixth entry gives in elements 1-3 the Hollerith value in 3A4 format of the study id associated with a database. The seventh through ninth entries give the record number of each of the three computed variables.

For the 12th and following entries of the PST is where information added to the files is kept track of. Elements 1-3 hold the source code id in 3A4 format. A routine called SQUASH is called to convert 4 A1 words to one A4 word to get the id into 3A4 format. This is the only routine in the system that is machine dependent; it assumes a 32 bit word. The fourth element holds a link in the chain or linked list and is used will be as unique as possible for each different id. The hash code points to a position in an array and the number in that array points to a proper entry in the symbol table. To add an id to the table its hash code is generated; the code points to an element of the array. This element is set to point to the next free space in the symbol table. A pointer that keeps track of where the next free space is incremented. To retrieve an id from the symbol table its hash code is generated. The code will be the same as was generated when the id was added to the symbol table. The code then points to the position in the array of pointers that points to the proper entry in the symbol table.

## CODAP80 INTREPRETER TOKENS AND THEIR NUMERIC VALUES SORTED ALPHABETICALLY

| | | | | | |
|---|---|---|---|---|---|
| . | 50001 | DIAGRAM | 144816 | OVL | 141609 |
| < | 30004 | DISTANCE | 145602 | OVLGRP | 144008 |
| <> | 30002 | DSQUARE | 144815 | PCNT | 142401 |
| <= | 30006 | D2 | 140807 | PERCENT | 144812 |
| ( | 50003 | ELSE | 142409 | PRINT | 143205 |
| + | 50005 | END | 141605 | RANDOM | 144019 |
| * | 50008 | EXECUTE | 144811 | RAWSUM | 75001 |
| ** | 50009 | FOR | 141601 | REL | 141611 |
| ) | 50004 | FORMAT | 144006 | RELY | 142414 |
| : | 50002 | FROM | 142407 | REPLACE | 144802 |
| - | 50006 | GROUPS | 144005 | REPORT | 144009 |
| / | 50007 | GVARS | 143209 | RESET | 143220 |
| > | 30003 | HEADING | 144801 | ROW | 141608 |
| >= | 30005 | HIHSN | 74005 | ROWS | 142415 |
| := | 50010 | HROWS | 143215 | SAVE | 142412 |
| = | 30001 | HSN | 72001 | SELECT | 144011 |
| ACUM | 60007 | HVARS | 143202 | SIMCOF | 75002 |
| ADDATA | 144014 | IF | 140805 | SORT | 142410 |
| ALL | 141604 | IN | 140806 | SORT | 60009 |
| AVALUE | 144001 | INCS | 142413 | SROWS | 143216 |
| AVE | 141602 | INCUMBENTS | 147202 | STD | 141603 |
| AVGA | 142417 | INPUT | 143203 | STD | 60006 |
| AVGP | 142416 | L | 140002 | STDA | 142419 |
| B | 140004 | LIST | 142420 | STDP | 142418 |
| BEGIN | 143211 | LOG | 60002 | SUM | 60003 |
| BINARY | 144010 | LOHSN | 74004 | SUMONLY | 144813 |
| BY | 140804 | MAX | 60005 | SVARS | 143208 |
| CCNST | 143219 | MAXIMIZE | 145604 | SYSCNST | 144806 |
| CCOLS | 143218 | MEAN | 60010 | SYSCOLS | 144807 |
| CGRPS | 143207 | MIN | 60004 | SYSGROUPS | 146404 |
| CMODS | 143214 | MINMEM | 144015 | SYSMODS | 144804 |
| COL | 141610 | MISSING | 144810 | SYSROWS | 144805 |
| COLS | 142402 | MODS | 142405 | TAPE | 142411 |
| COLUMN | 144013 | MODULES | 144803 | TASKS | 143206 |
| COLUMNS | 144809 | N | 140001 | THEN | 142408 |
| CONSTANTS | 146405 | NHIST | 74002 | TROWS | 143212 |
| COPY | 142403 | NINCS | 74001 | TVARS | 143201 |
| CORR | 142404 | NONZERO | 76001 | USING | 143213 |
| COUNT | 143204 | NOPAGE | 144017 | VARSUM | 144012 |
| CREATE | 144003 | NOREM | 143210 | WITHIN | 75003 |
| CROWS | 143217 | NOREMARKS | 146401 | | |
| CUM | 60001 | NOSAVE | 144002 | | |
| D | 140003 | NOSKIP | 144016 | | |
| DCUM | 60008 | NOSTID | 144018 | | |
| DECODE | 144007 | NOSUMMARY | 146403 | | |
| DELETE | 144004 | NOT | 141607 | | |
| DES | 141606 | NSEC | 73001 | | |
| DESCEND | 144808 | NTASK | 74003 | | |
| DESCENDING | 147201 | ON | 140801 | | |
| DESCRIBE | 145601 | OVERLAP | 144814 | | |

21

## CODAP80 INTERPRETER TOKENS AND THEIR NUMERIC VALUES SORTED NUMERICALLY

| | | | | | |
|---|---|---|---|---|---|
| = | 30001 | END | 141605 | GROUPS | 144005 |
| <> | 30002 | DES | 141606 | FORMAT | 144006 |
| > | 30003 | NOT | 141607 | DECODE. | 144007 |
| < | 30004 | ROW | 141608 | CVLGRP | 144008 |
| >= | 30005 | OVL | 141609 | REPORT | 144009 |
| <= | 30006 | COL | 141610 | BINARY | 144010 |
| . | 50001 | REL | 141611 | SELECT | 144011 |
| : | 50002 | PCNT | 142401 | VARSUM | 144012 |
| ( | 50003 | COLS | 142402 | COLUMN | 144013 |
| ) | 50004 | COPY | 142403 | ADDATA | 144014 |
| + | 50005 | CORR | 142404 | MINMEM | 144015 |
| - | 50006 | MODS | 142405 | NOSKIP | 144016 |
| / | 50007 | FROM | 142407 | NOPAGE | 144017 |
| * | 50008 | THEN | 142408 | NOSTID | 144018 |
| ** | 50009 | ELSE | 142409 | RANDOM | 144019 |
| := | 50010 | SORT | 142410 | HEADING | 144801 |
| CUM | 60001 | TAPE | 142411 | REPLACE | 144802 |
| LOG | 60002 | SAVE | 142412 | MODULES | 144803 |
| SUM | 60003 | INCS | 142413 | SYSMODS | 144804 |
| MIN | 60004 | RELY | 142414 | SYSROWS | 144805 |
| MAX | 60005 | ROWS | 142415 | SYSCNST | 144806 |
| STD | 60006 | AVGP | 142416 | SYSCOLS | 144807 |
| ACUM | 60007 | AVGA | 142417 | DESCEND | 144808 |
| DCUM | 60008 | STDP | 142418 | COLUMNS | 144809 |
| SQRT | 60009 | STDA | 142419 | MISSING | 144810 |
| MEAN | 60010 | LIST | 142420 | EXECUTE | 144811 |
| HSN | 72001 | TVARS | 143201 | PERCENT | 144812 |
| NSEC | 73001 | HVARS | 143202 | SUMONLY | 144813 |
| NINCS | 74001 | INPUT | 143203 | OVERLAP | 144814 |
| NHIST | 74002 | COUNT | 143204 | DSQUARE | 144815 |
| NTASK | 74003 | PRINT | 143205 | DIAGRAM | 144816 |
| LOHSN | 74004 | TASKS | 143206 | DESCRIBE | 145601 |
| HIHSN | 74005 | CGRPS | 143207 | DISTANCE | 145602 |
| RAWSUM | 75001 | SVARS | 143208 | MAXIMIZE | 145604 |
| SIMCOF | 75002 | GVARS | 143209 | NOREMARKS | 146401 |
| WITHIN | 75003 | NOREM | 143210 | NOSUMMARY | 146403 |
| NONZERO | 76001 | BEGIN | 143211 | SYSGROUPS | 146404 |
| N | 140001 | TROWS | 143212 | CONSTANTS | 146405 |
| L | 140002 | USING | 143213 | DESCENDING | 147201 |
| D | 140003 | CMODS | 143214 | INCUMBENTS | 147202 |
| B | 140004 | HROWS | 143215 | | |
| ON | 140801 | SROWS | 143216 | | |
| BY | 140804 | CROWS | 143217 | | |
| IF | 140805 | CCOLS | 143218 | | |
| IN | 140806 | CCNST | 143219 | | |
| D2 | 140807 | RESET | 143220 | | |
| FOR | 141601 | AVALUE | 144001 | | |
| AVE | 141602 | NOSAVE | 144002 | | |
| STD | 141603 | CREATE | 144003 | | |
| ALL | 141604 | DELETE | 144004 | | |

Routines that ares used to access the symbol tables are WRTRVE (WST retrieve), PRTRVE (PST retrieve), WRTV12 (WST retrieve length 12), WSTADD (add to the WST), WSTREP (replace an entry of the WST), and WSTSUB (subtract and entry of the WST). WRTRVE will return the first 11 elements of an entry based on an index number passed to it. Since the numeric tokens for ids contain the index number it is a simple matter to compute the index number from the token when the token's entry in the WST needs to be retrieved. WRTV12 does the same thing that WRTRVE does except that it retrieves all 12 elements of each entry. WSTADD will add a new entry to the WST based on an input index number. WSTREP will replace an entry of the WST based on an input index number. RPTRVE will retrieve an entry of the PST given the source code id in 3A4 format and its hash code. PRTRVE has to search through the linked list to find an id if there is a collision, but WRTRVE and WRTV12 do not have to because the index they get points directly to the proper entry in the symbol table. This is one of the advantages of the numeric tokens because they make it possible to get to resolve collisions in hash values. The fifth element gives the type of data represented, a 1 means a row, a 2 a group, a 3 a module, and a 4 a column. The sixth element contains the length of the vector. Elements 7-9 contain the module or group that a column or row was created for, or is blank if the entry represents a module or group. Element 10 gives the record number of the vector. Element 11 gives the record number of its associated remark. Element 12 is the label number that points to the symbol table.

The WST is created during each run by first calling the subroutine WSETUP (WST set up) to copy the first 11 entries from the PST to the WST. As ids are encoutnered by GTOKEN they are added to the WST by calling the subroutine BLDWST. BLDWST first checks to see if the id is already present in the PST, if it is that entry from the PST is copied into the WST. If it is not the id is added to the WST by filling in the id (elements 1-3), and the chain (element 4), and the membership (elements 7-9). The syntax analysis routine that called GTOKEN will set the type, membership, and remark record number elements. The length, involves taking the Hollerith value of the source code id and treating it as a number, then computing a hash code that will be as unique as possible for each different id. This involves taking the Hollerith value of the source code id information from the WST without any search.

The access routines have been set up so that any time a module needs information from a symbol table it does not deal directly with the symbol table. This should prevent problems that might develop if a module made an incorrect change to a symbol table; which would then affect other modules.

## APPENDIX A

## FORTRAN FG PROC
## COMPILE, LINK EDIT AND GO PROCEDURE
## FOR THE G1 FORTRAN COMPILER

## FORTRAN FG PROC
## COMPILE, LINK EDIT AND GO PROCEDURE
## FOR THE G1 FORTRAN COMPILER

```
//FG     EXEC  PGM=IEYFORT,REGION=192K
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
/*
//LKED   EXEC  PGM=IEWL,REGION=128K,PARM=(XREF,LET,LIST)
//SYSLIB   DD  DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD  DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//             SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),
//             DCB=BLKSIZE=1024
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
/*
//GO     EXEC PGM=*.LKED.SYSLMOD
//FT05F001 DD  DDNAME=SYSIN
//FT06F001 DD  SYSOUT=A
//FT07F001 DD  SYSOUT=B
```

## APPENDIX B

## CODAP80 FILE LAYOUTS

## CONTROL AND DATA FILE LAYOUTS

```
SAMPLEDATA8000007001000400050005Y
HH.H.HTSTSTSTSTS
@@
H1   =SEX;
H2   =AGE;
H3   =YEARS ON JOB;
H4   =INCUMBENT ID;
T1   =SUBDUE VIOLENT INMATES;
T2   =SHAKE DOWN INMATES;
T3   =SHAKE DOWN VISITORS;
T4   =ESCORT INMATES;
T5   =TESTIFY IN COURT;
S1   =SECONDARY - SUBDUE VIOLENT INMATES;
S2   =SECONDARY - SHAKE DOWN INMATES;
S3   =SECONDARY - SHAKE DOWN VISITORS;
S4   =SECONDARY - ESCORT INMATES;
S5   =SECONDARY - TESTIFY IN COURT;
@@
H1  1=MALE; 2=FEMALE;
S1-S5 1=DO; 2=ASSIST; 3=SUPERVISE;
@@
```

```
219 117 1111220
14119212420 2221
1  1630 33430 0
127 344 41310 63
123 251 1122510
15330642710 0 0
2  1170 0 225231
```

27

# INPFILE LAYOUT

| RECORD # | RECORD FIELD CONTENTS 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 19 | 1 | 1 | 63.64 | 9.09 | 9.09 | 18.18 | 0.00 | . | 1 | 1 | 2 | . | 11 | 4 | ... | 0 |
| 2 | 1 | 41 | 19 | 2 | 11.11 | 44.44 | 0.00 | 22.22 | 22.22 | 2 | 2 | . | 2 | 1 | 9 | 4 | ... | 0 |
| 3 | 1 | . | 16 | 3 | 0.00 | 42.86 | 57.14 | 0.00 | 0.00 | . | 3 | 3 | . | . | 7 | 2 | ... | 0 |
| 4 | 1 | 27 | 3 | 4 | 23.53 | 23.53 | 17.65 | 0.00 | 35.29 | . | 1 | 1 | . | 3 | 17 | 4 | ... | 0 |
| 5 | 1 | 23 | 2 | 5 | 11.11 | 11.11 | 22.22 | 55.56 | 0.00 | . | 1 | 2 | 1 | . | 9 | 4 | ... | 0 |
| 6 | 1 | 53 | 30 | 6 | 36.36 | 63.64 | 0.00 | 0.00 | 0.00 | 2 | 1 | . | . | . | 11 | 2 | ... | 0 |
| 7 | 2 | . | 11 | 7 | 0.00 | 0.00 | 20.00 | 50.00 | 30.00 | . | . | 2 | 2 | 1 | 10 | 3 | ... | 0 |

# VARCOM LAYOUT

```
                              RECORD CONTENTS
RECORD #                                                   WORD 61 ------|
                                                                         |
                                                                         V
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   1     |   3|SEX |    |    |    |    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   2     |   3|AGE |    |    |    |    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   3     |  12|YEAR|S ON| JOB|    |    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   4     |  12|INCU|MBEN|T ID|    |    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   5     |  22|SUBD|UE V|IOLE|NT I|NMAT|ES  |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   6     |  18|SHAK|E DO|WN I|NMAT|ES  |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   7     |  19|SHAK|E DO|WN V|ISIT|ORS |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   8     |  14|ESCO|RT I|NMAT|ES  |    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
   9     |  16|TEST|IFY |IN C|OURT|    |    |    |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
  10     |  34|SECO|NDAR|Y - |SUBD|UE V|IOLE|NT I|NMAT|ES  |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
  11     |  30|SECO|NDAR|Y - |SHAK|E DO|WN I|NMAT|ES  |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
  12     |  31|SECO|NDAR|Y - |SHAK|E DO|WN V|ISIT|ORS |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
  13     |  26|SECO|NDAR|Y - |ESCO|RT I|NMAT|ES  |    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----

          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
  14     |  28|SECO|NDAR|Y - |TEST|IFY |IN C|OURT|    |    |...|    |
          ---- ---- ---- ---- ---- ---- ---- ---- ---- ----      ----
```

29

## SYNTAB1 LAYOUT

```
:    :    :    :     1:   43:   4:   0:   0:   0:   0:   1:   0:   0
:    :    :    :     1:    0:   5:   0:   0:   0:   0:   5:   0:   0
:    :    :    :     1:    0:   5:   0:   0:   0:   0:  10:   0:   0
:    :    :    :     1:    0:   0:   0:   0:   0:   0:   0:   0:   0
:    :    :    :  5000:    0:   7:   0:   0:   0:   0:   0:   0:   0
:SAMP:LEDA:TABO:    15:    0:   0:   0:   0:   0:   0:   0:   0:   0
:    :    :    :     0:    0:   5:   0:   0:   0:   0:   0:   0:   0
:    :    :    :     0:    0:   0:   0:   0:   0:   0:   0:   0:   0
:    :    :    :     0:    0:   0:   0:   0:   0:   0:   0:   0:   0
:    :    :    :     0:    0:   0:   0:   0:   0:   0:   0:   0:   0
```

# DECODE LAYOUT

```
RECORD #
|
|                        RECORD FIELD CONTENTS
|
V     1        2        3        4        5        6     ...     10

1    1.0      0.0      2.0      3.0      3.0      7.0    ...     0.0

2   80001   100001     0.0      0.0      0.0      0.0    ...     0.0
    80001   100005     0.0      0.0      0.0      0.0    ...     0.0
      3.0      5.0

3     H      80001    80001     1.0     MALE

4     H      80001    80001     2.0     FEMALE

5     S     100001   100005     1.0     DO

6     S     100001   100005     2.0     ASSIST

7     S     100001   100005     3.0     SUPERVISE
```

## GRPFILE LAYOUT

RECORD FIELD CONTENTS

| RECORD # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 2.00 | 3.00 | 4.00 | 51.31 | 2.00 | 6.00 | 2.00 | 1.00 | 53.16 | 47.47 |
| 2 | 70.00 | 5.00 | 7.00 | 1.00 | 1.00 | 70.00 | 56.86 | 2.00 | 4.00 | 1.00 | 1.00 | 56.86 | 51.31 | 2.00 | 6.00 | 2.00 | 1.00 | 53.16 | 47.47 |
| 3 | 2.00 | 3.00 | 4.00 | 5.00 | 4.00 | 6.00 | 1.00 | 3.00 | 4.00 | 7.00 | 1.00 | 7.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 4.00 | 3.00 | 5.00 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 | 0.00 | 0.00 | 1.00 | 3.00 | 0.00 | 4.00 | 5.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

| RECORD # | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | ... | 3240 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7.00 | 5.00 | 1.00 | 2.00 | 54.98 | 42.30 | 2.00 | 3.00 | 3.00 | 1.00 | 47.73 | 38.38 | 1.00 | 2.00 | 3.00 | 4.00 | 43.42 | ... | 0.00 |
| 2 | 1.00 | 5.00 | 1.00 | 2.00 | 54.98 | 42.30 | 2.00 | 3.00 | 3.00 | 1.00 | 47.73 | 38.38 | 1.00 | 2.00 | 3.00 | 4.00 | 43.42 | ... | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ... | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ... | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ... | 0.00 |

## GRPHSN LAYOUT

```
RECORD #
|
|                 RECORD FIELD CONTENTS
|
V       1       2       3       4       5       6       7     . . .     10

1       1       5       7       2       4       6       3     . . .     0
```

ATE

LME